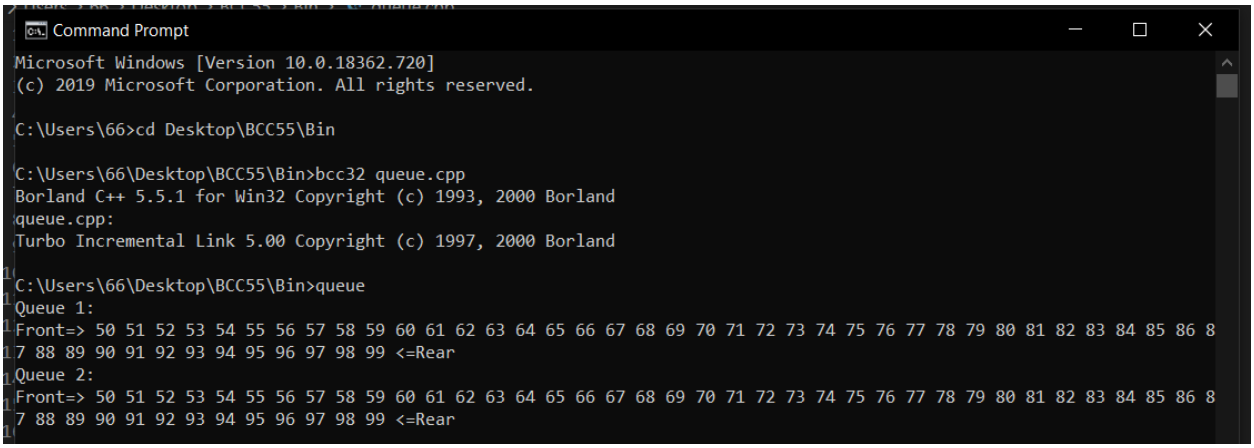
แบบ ascending

ผลลัพธ์



Source Code

```
#include <stdio.h>
#include <stdlib.h>
    typedef struct node
      {
       void*        dataPtr;
       struct node* next;
      } QUEUE_NODE;
    typedef struct
      {
       QUEUE_NODE* front;
       QUEUE_NODE* rear;
       int       count;
      } QUEUE;


    QUEUE* createQueue  (void);
    QUEUE* destroyQueue (QUEUE* queue);

    bool  dequeue   (QUEUE* queue, void** itemPtr);
    bool  enqueue   (QUEUE* queue, void*  itemPtr);
    bool  queueFront (QUEUE* queue, void** itemPtr);
    bool  queueRear  (QUEUE* queue, void** itemPtr);
    int   queueCount (QUEUE* queue);
```

```
    bool  emptyQueue (QUEUE* queue);
    bool  fullQueue  (QUEUE* queue);


void printQueue   (QUEUE* stack);

int main (void)
{

    QUEUE* Nickqueue1;
    QUEUE* Nickqueue2;
    int*   numPtr;
    int** itemPtr;


    Nickqueue1 = createQueue();
    Nickqueue2 = createQueue();
    for (int NickCount = 50; NickCount <= 99; NickCount++)
       {
         numPtr  = (int*)malloc(sizeof(NickCount));
         *numPtr = NickCount;
         enqueue(Nickqueue1, numPtr);
         if (!enqueue(Nickqueue2, numPtr))
            {
             printf ("\n\a**Queue overflow\n\n");
             exit (100);
            } // if !enqueue
       } // for
    printf ("Queue 1:\n");
    printQueue (Nickqueue1);
    printf ("Queue 2:\n");
    printQueue (Nickqueue2);
    return 0;
}


QUEUE* createQueue (void)
{

    QUEUE* queue;

    queue = (QUEUE*) malloc (sizeof (QUEUE));
    if (queue)
       {
        queue->front  = NULL;
```

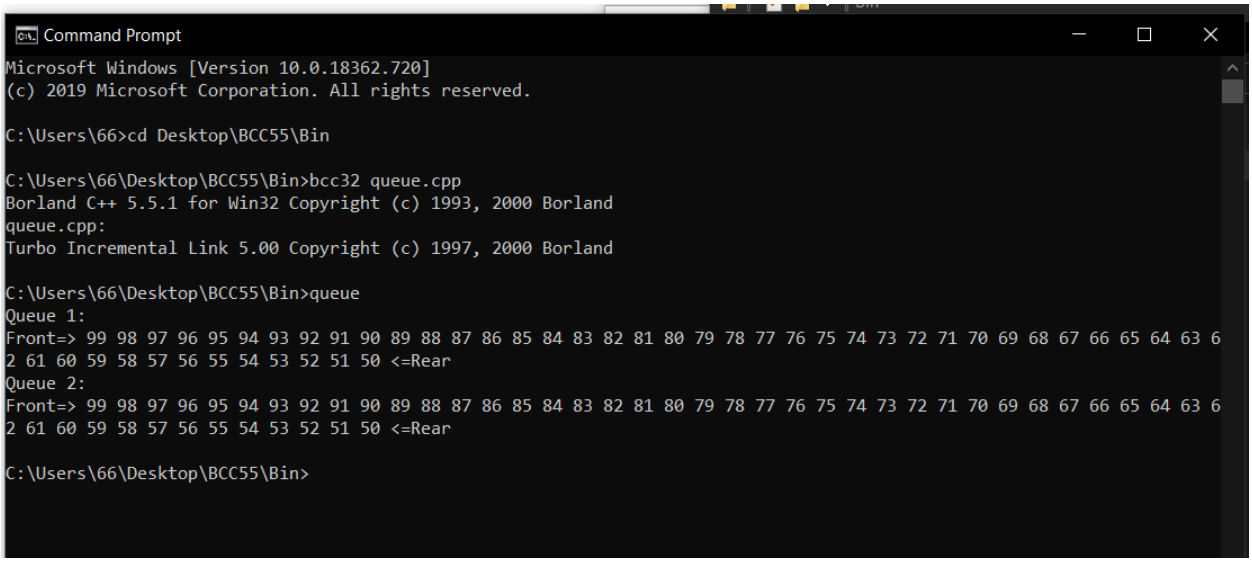
```
        queue->rear   = NULL;
        queue->count  = 0;
       }
    return queue;
}


bool enqueue (QUEUE* queue, void* itemPtr)
{

        QUEUE_NODE* newPtr = (QUEUE_NODE*)malloc(sizeof(QUEUE_NODE));
    newPtr->dataPtr = itemPtr;
    newPtr->next = NULL;
    if (queue->count == 0)

       queue->front  = newPtr;
    else
       queue->rear->next = newPtr;
    (queue->count)++;
    queue->rear = newPtr;
    return true;
}
bool dequeue (QUEUE* queue, void** itemPtr)
{

    QUEUE_NODE* deleteLoc;

    if (!queue->count)
        return false;
    *itemPtr  = queue->front->dataPtr;
    deleteLoc = queue->front;
    if (queue->count == 1)

       queue->rear  = queue->front = NULL;
    else
       queue->front = queue->front->next;
    (queue->count)--;
    free (deleteLoc);
    return true;
}
bool queueFront (QUEUE* queue, void** itemPtr)
{

    if (!queue->count)
        return false;
```

```
    else
       {
        *itemPtr = queue->front->dataPtr;
         return true;
       }
}


bool queueRear (QUEUE* queue, void** itemPtr)
{

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


bool emptyQueue (QUEUE* queue)
{

    return (queue->count == 0);
}


bool fullQueue (QUEUE* queue)
{

if(emptyQueue(queue)) return false; // Not check in heap

QUEUE_NODE* temp;

    temp = (QUEUE_NODE*)malloc(sizeof(*(queue->rear)));
    if (temp)
       {
        free (temp);
        return false;
       }
    return true;
}
```

```
int queueCount(QUEUE* queue)
{

    return queue->count;
}


QUEUE* destroyQueue (QUEUE* queue)
{

    QUEUE_NODE* deletePtr;

    if (queue)
       {
        while (queue->front != NULL)
           {
            free (queue->front->dataPtr);
            deletePtr   = queue->front;
            queue->front = queue->front->next;
            free (deletePtr);
           }
        free (queue);
       }
    return NULL;
}

void printQueue(QUEUE* queue)
{

    QUEUE_NODE* node = queue->front;

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

แบบ descending

ผลลัพธ์



Source Code

```
#include <stdio.h>
#include <stdlib.h>
    typedef struct node
      {
       void*        dataPtr;
       struct node* next;
      } QUEUE_NODE;
    typedef struct
      {
       QUEUE_NODE* front;
       QUEUE_NODE* rear;
       int       count;
      } QUEUE;


    QUEUE* createQueue  (void);
    QUEUE* destroyQueue (QUEUE* queue);
```

```
    bool  dequeue   (QUEUE* queue, void** itemPtr);
    bool  enqueue   (QUEUE* queue, void*  itemPtr);
    bool  queueFront (QUEUE* queue, void** itemPtr);
    bool  queueRear  (QUEUE* queue, void** itemPtr);
    int   queueCount (QUEUE* queue);
    bool  emptyQueue (QUEUE* queue);
    bool  fullQueue  (QUEUE* queue);


void printQueue   (QUEUE* stack);

int main (void)
{

    QUEUE* Nickqueue1;
    QUEUE* Nickqueue2;
    int*   numPtr;
    int** itemPtr;


    Nickqueue1 = createQueue();
    Nickqueue2 = createQueue();
    for (int NickCount = 99; NickCount >= 50; NickCount--)
       {
         numPtr  = (int*)malloc(sizeof(NickCount));
         *numPtr = NickCount;
         enqueue(Nickqueue1, numPtr);
         if (!enqueue(Nickqueue2, numPtr))
            {
             printf ("\n\a**Queue overflow\n\n");
             exit (100);
            } // if !enqueue
       } // for
    printf ("Queue 1:\n");
    printQueue (Nickqueue1);
    printf ("Queue 2:\n");
    printQueue (Nickqueue2);
    return 0;
}


QUEUE* createQueue (void)
{

    QUEUE* queue;
```

```
    queue = (QUEUE*) malloc (sizeof (QUEUE));
    if (queue)
       {
        queue->front  = NULL;
        queue->rear   = NULL;
        queue->count  = 0;
       }
    return queue;
}


bool enqueue (QUEUE* queue, void* itemPtr)
{

        QUEUE_NODE* newPtr = (QUEUE_NODE*)malloc(sizeof(QUEUE_NODE));
    newPtr->dataPtr = itemPtr;
    newPtr->next = NULL;
    if (queue->count == 0)

       queue->front  = newPtr;
    else
       queue->rear->next = newPtr;
    (queue->count)++;
    queue->rear = newPtr;
    return true;
}
bool dequeue (QUEUE* queue, void** itemPtr)
{

    QUEUE_NODE* deleteLoc;

    if (!queue->count)
        return false;
    *itemPtr  = queue->front->dataPtr;
    deleteLoc = queue->front;
    if (queue->count == 1)

       queue->rear  = queue->front = NULL;
    else
       queue->front = queue->front->next;
    (queue->count)--;
    free (deleteLoc);
    return true;
}
```

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

    if (!queue->count)
        return false;
    else
       {
        *itemPtr = queue->front->dataPtr;
         return true;
       }
}


bool queueRear (QUEUE* queue, void** itemPtr)
{

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


bool emptyQueue (QUEUE* queue)
{

    return (queue->count == 0);
}


bool fullQueue (QUEUE* queue)
{

if(emptyQueue(queue)) return false; // Not check in heap

QUEUE_NODE* temp;

    temp = (QUEUE_NODE*)malloc(sizeof(*(queue->rear)));
    if (temp)
       {
        free (temp);
        return false;
```

```
       }
    return true;
}


int queueCount(QUEUE* queue)
{

    return queue->count;
}


QUEUE* destroyQueue (QUEUE* queue)
{

    QUEUE_NODE* deletePtr;

    if (queue)
       {
        while (queue->front != NULL)
           {
            free (queue->front->dataPtr);
            deletePtr   = queue->front;
            queue->front = queue->front->next;
            free (deletePtr);
           }
        free (queue);
       }
    return NULL;
}

void printQueue(QUEUE* queue)
{

    QUEUE_NODE* node = queue->front;

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