

แบบ ascending

ผลลัพธ์

```
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\66>cd Desktop\BCC55\Bin

C:\Users\66\Desktop\BCC55\Bin>bcc32 queue.cpp
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
queue.cpp:
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland

C:\Users\66\Desktop\BCC55\Bin>queue
Queue 1:
Front=> 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 <=Rear
Queue 2:
Front=> 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 <=Rear
```

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

```
Command Prompt
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\66>cd Desktop\BCC55\Bin

C:\Users\66\Desktop\BCC55\Bin>bcc32 queue.cpp
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
queue.cpp:
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland

C:\Users\66\Desktop\BCC55\Bin>queue
Queue 1:
Front=> 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 <=Rear
Queue 2:
Front=> 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 <=Rear

C:\Users\66\Desktop\BCC55\Bin>
```

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* queue);

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;
}

```

