รายงาน

Circular Queue

จัดทำโดย

นายธนรัก ชุ่มสวัสดิ์

รหัสศึกษา 68543210018-6

อาจารย์ผู้สอน

นายปิยพล ยืนยงสถาวร

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structures and Algorithms

สาขาวิชาวิศวกรรมซอฟต์แวร์ มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ภาคเรียนที่ 2 ปีการศึกษา 2568

Code Program Stack

```c
#include <stdio.h>
#include <stdlib.h>

#define N 5   // array size; usable queue capacity is N-1 (indices 1..N-1)

int Q[N];           // queue storage (we use indices 1..N-1)
int x, Qnumber = 0;  // Qnumber: total enqueued count (for display)
int F = 0, R = 0;    // Front and Rear (0 means empty)
char status = 'N';   // 'N' = NORMAL, 'O' = OVERFLOW, 'U' = UNDERFLOW
char ch;             // command input

void insertCQ(int y) {
    // Queue is full when R == F-1 (in modular sense) or special case when F==1 and R==N-1
    if ((F != 0 && R == F - 1) || (R == N - 1 && F == 1)) {
        printf("!!! OVER FLOW !!!\n");
        status = 'O';
        return;
    }

    // advance R (circular 1..N-1)
    if (R == N - 1) {
        R = 1;
    } else {
        R++;
        if (F == 0)   // queue was empty, set front to 1
            F = 1;
    }

    Qnumber++;
    printf("You are queue number : %d\n", Qnumber);
    Q[R] = y;
    status = 'N';
}

int deleteCQ() {
    int y;
    if (F == 0) { // empty
        printf("\n!!! UNDER FLOW !!!\n");
        status = 'U';
```

```
        return -1; // sentinel to indicate no data
    }

    y = Q[F];
    if (F == R) { // last element removed -> become empty
        F = 0;
        R = 0;
    } else {
        if (F == N - 1)
            F = 1; // wrap
        else
            F++;
    }
    status = 'N';
    return y;
}

int DataInQueue() {
    int y = 0;
    if (F != 0 && R != 0) {
        if (F <= R)
            y = R - F + 1;              // normal case
        else
            y = (N - 1) - F + 1 + R;    // wrapped case
    }
    return y;
}

void ShowAllQueue() {
    int i;
    printf("\n--- QUEUE STATUS ---\n");
    printf("Capacity (N-1) : %d\n", N - 1);
    printf("Status = %c\n", status);
    printf("Data waiting in queue = %d\n", DataInQueue());
    printf("F = %d / R = %d\n", F, R);
    printf("All slots (index:value):\n");
    for (i = 1; i < N; i++) {
        printf("%d:%d  ", i, Q[i]);
    }
```

```
    printf("\n--------------------\n");
}

int main() {
    printf("CIRCULAR QUEUE PROGRAM...\n");
    printf("=========================\n");

    // initialize Q array to 0 (optional)
    for (int i = 0; i < N; i++) Q[i] = 0;

    ch = ' ';
    while (ch != 'E' && ch != 'e') {
        printf("\n[1=INSERT : 2=DELETE : E=Exit] : ");
        if (scanf(" %c", &ch) != 1) {
            // input failure
            break;
        }

        switch (ch) {
        case '1':
            printf("Insert Number : ");
            if (scanf("%d", &x) == 1) {
                insertCQ(x);
                ShowAllQueue();
            } else {
                printf("Invalid number input.\n");
                // clear invalid token
                int c;
                while ((c = getchar()) != '\n' && c != EOF) { }
            }
            break;

        case '2':
            x = deleteCQ();
            if (x != -1)
                printf("\nData from Queue = %d\n", x);
            ShowAllQueue();
            break;
```

```
        case 'E':
        case 'e':
            printf("Exiting...\n");
            break;

        default:
            printf("Unknown command. Use 1, 2 or E.\n");
            break;
        } // end switch
    } // end while

    return 0;
}
```

อธิบายการทำงานของโปรแกรม

ส่วนการประกาศไลบรารีและตัวแปรกำหนดขนาดคิว

โปรแกรมเริ่มต้นด้วยการเรียกใช้ไลบรารี <stdio.h> และ <stdlib.h> เพื่อรองรับการแสดงผลและจัดการหน่วยความจำ จากนั้นกำหนดค่าคงที่ N เพื่อใช้เป็นขนาดของอาร์เรย์สำหรับเก็บข้อมูลในคิว โดยตั้งใจให้ความจุใช้งานจริงเป็น N-1 เพื่อรองรับหลักการของ circular queue ที่ต้องเว้นหนึ่งช่องไว้สำหรับตรวจสอบ

สถานะเต็มหรือว่าง ซึ่งส่วนนี้เป็นพื้นฐานที่ทำให้ทุกฟังก์ชันถัดไปสามารถอ้างอิงขนาดคิวเดียวกันได้อย่างถูกต้อง

```
#include <stdio.h>
#include <stdlib.h>

#define N 5   // array size; usable queue capacity is N-1 (indices 1..N-1)
```

ส่วนการประกาศตัวแปรของคิว

หลังจากกำหนดขนาดแล้ว โปรแกรมประกาศอาร์เรย์ Q[N] สำหรับเก็บข้อมูลในคิว พร้อมตัวแปร F (Front) และ R (Rear) เพื่อชี้ตำแหน่งหัวและท้ายของคิว โดยกำหนดค่าเริ่มต้นเป็น 0 เพื่อแสดงว่าคิวยังว่างอยู่ นอกจากนี้ยังมีตัวแปร Qnumber สำหรับนับจำนวนการเข้าแถวทั้งหมด และตัวแปร status เพื่อบอกสถานะของคิวว่าปกติ ล้น หรือว่าง ซึ่งตัวแปรเหล่านี้จะถูกใช้และเปลี่ยนค่าโดยฟังก์ชันจัดการคิวในลำดับถัดไป

```
int Q[N];           // queue storage (we use indices 1..N-1)
int x, Qnumber = 0;  // Qnumber: total enqueued count (for display)
int F = 0, R = 0;    // Front and Rear (0 means empty)
char status = 'N';   // 'N' = NORMAL, 'O' = OVERFLOW, 'U' = UNDERFLOW
char ch;            // command input
```

ฟังก์ชัน insertCQ() สำหรับเพิ่มข้อมูลลงคิว

เมื่อผู้ใช้เลือกคำสั่งเพิ่มข้อมูล โปรแกรมจะเรียกฟังก์ชัน insertCQ() ซึ่งเริ่มจากการตรวจสอบว่าคิวเต็มหรือไม่ หากพบว่าเต็มจะตั้งค่า status เป็น Overflow และหยุดการทำงานทันที แต่ถ้าคิวยังไม่เต็ม ฟังก์ชันจะเลื่อนตำแหน่ง R ไปยังช่องถัดไปตามหลัก circular queue และหากคิวเดิมว่างอยู่ก็จะกำหนด F ให้

ชี้ตำแหน่งแรก จากนั้นข้อมูลใหม่จะถูกเก็บลงในอาร์เรย์ และฟังก์ชันนี้จะส่งผลลัพธ์เป็นการอัปเดตตำแหน่งคิว เพื่อให้ฟังก์ชันอื่นสามารถนำไปใช้งานต่อได้

```
void insertCQ(int y) {
void insertCQ(int y) {
    // Queue is full when R == F-1 (in modular sense) or special case when F==1 and R==N-1
    if ((F != 0 && R == F - 1) || (R == N - 1 && F == 1)) {
        printf("!!! OVER FLOW !!!\n");
        status = 'O';
        return;
    }

    // advance R (circular 1..N-1)
    if (R == N - 1) {
        R = 1;
    } else {
        R++;
        if (F == 0)   // queue was empty, set front to 1
            F = 1;
    }

    Qnumber++;
    printf("You are queue number : %d\n", Qnumber);
    Q[R] = y;
    status = 'N';
}
```

ฟังก์ชัน deleteCQ() สำหรับนำข้อมูลออกจากคิว

หลังจากมีการเพิ่มข้อมูลแล้ว เมื่อผู้ใช้เลือกนำข้อมูลออก โปรแกรมจะเรียกฟังก์ชัน deleteCQ() ซึ่งเริ่มจากการตรวจสอบว่าคิวว่างหรือไม่ หากว่างจะรายงาน Underflow และส่งค่าพิเศษกลับไป แต่ถ้ามีข้อมูลอยู่ ฟังก์ชันจะดึงข้อมูลที่ตำแหน่ง F ออกมา และเลื่อนตำแหน่ง F ไปยังช่องถัดไป หรือหากข้อมูลที่ถูกนำออกเป็นตัวสุดท้ายก็จะรีเซ็ตทั้ง F และ R เป็น 0 ผลลัพธ์จากฟังก์ชันนี้จะถูกส่งกลับไปยังโปรแกรมหลักเพื่อแสดงผลต่อไป

```
int deleteCQ() {
    int y;
    if (F == 0) { // empty
        printf("\n!!! UNDER FLOW !!!\n");
        status = 'U';
        return -1; // sentinel to indicate no data
    }

    y = Q[F];
    if (F == R) { // last element removed -> become empty
        F = 0;
        R = 0;
    } else {
        if (F == N - 1)
            F = 1; // wrap
        else
            F++;
    }
    status = 'N';
    return y;
}
```

ฟังก์ชัน DataInQueue() สำหรับนับจำนวนข้อมูลในคิว

หลังจากมีการเพิ่มหรือลบข้อมูล โปรแกรมจะเรียกฟังก์ชัน DataInQueue() เพื่อคำนวณจำนวนข้อมูลที่ค้างอยู่ในคิว โดยฟังก์ชันนี้จะตรวจสอบตำแหน่ง F และ R ว่าอยู่ในลักษณะปกติหรือมีการวนรอบ แล้วคำนวณจำนวนข้อมูลตามสูตรที่เหมาะสม ผลลัพธ์จากฟังก์ชันนี้จะถูกส่งต่อไปยังฟังก์ชันแสดงผลเพื่อให้ผู้ใช้ทราบสถานะปัจจุบันของคิว

```
int DataInQueue() {
    int y = 0;
    if (F != 0 && R != 0) {
        if (F <= R)
            y = R - F + 1;              // normal case
        else
            y = (N - 1) - F + 1 + R;    // wrapped case
    }
    return y;
}
```

ฟังก์ชัน ShowAllQueue() สำหรับแสดงสถานะคิว

เมื่อได้ข้อมูลครบถ้วนจากฟังก์ชันก่อนหน้าแล้ว ShowAllQueue() จะถูกเรียกเพื่อแสดงภาพรวมของคิวทั้งหมด โดยจะแสดงความจุของคิว สถานะปัจจุบัน จำนวนข้อมูลที่รออยู่ รวมถึงตำแหน่ง F และ R และค่าภายในอาร์เรย์ทุกช่อง ฟังก์ชันนี้ทำหน้าที่เชื่อมผลลัพธ์จากทุกฟังก์ชันเข้าด้วยกัน เพื่อให้ผู้ใช้เข้าใจการเปลี่ยนแปลงของคิวในแต่ละขั้นตอน

```
void ShowAllQueue() {
    int i;
    printf("\n--- QUEUE STATUS ---\n");
    printf("Capacity (N-1) : %d\n", N - 1);
    printf("Status = %c\n", status);
    printf("Data waiting in queue = %d\n", DataInQueue());
    printf("F = %d / R = %d\n", F, R);
    printf("All slots (index:value):\n");
```

```
    for (i = 1; i < N; i++) {
        printf("%d:%d  ", i, Q[i]);
    }
    printf("\n---------------------\n");
}
```

ฟังก์ชัน main() และลำดับการทำงานของโปรแกรม

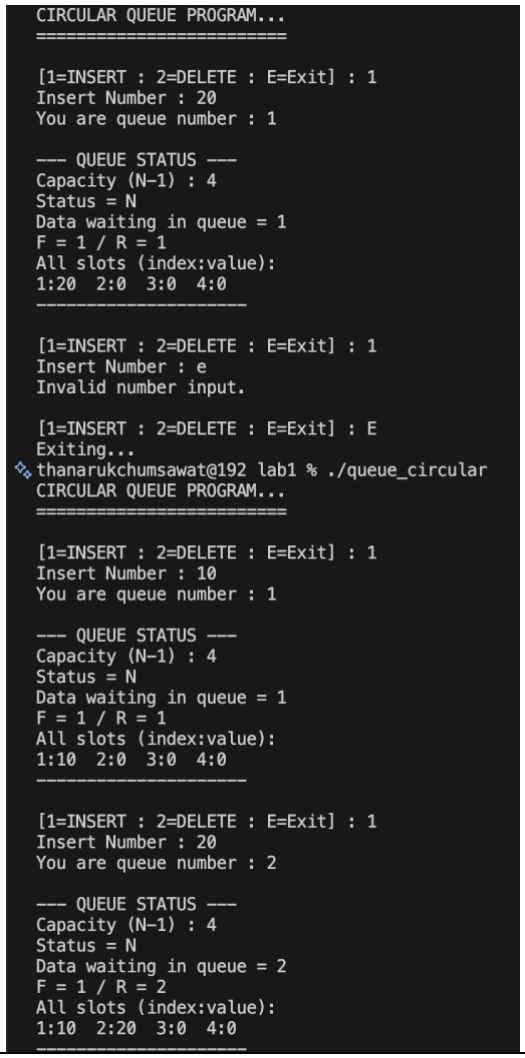
สุดท้ายโปรแกรมจะเริ่มทำงานที่ฟังก์ชัน main() ซึ่งทำหน้าที่เตรียมค่าเริ่มต้นของคิวและรับคำสั่งจากผู้ใช้ เมื่อผู้ใช้เลือกเพิ่มข้อมูล main() จะส่งค่าที่รับมาไปยัง insertCQ() จากนั้นเรียก ShowAllQueue() เพื่อแสดงผล และเมื่อผู้ใช้เลือกนำข้อมูลออก main() จะเรียก deleteCQ() แล้วส่งค่าที่ได้ไปแสดงผลเช่นกัน ลำดับการทำงานนี้จะวนซ้ำไปเรื่อย ๆ จนกว่าผู้ใช้จะเลือกออกจากโปรแกรม ทำให้ทุกฟังก์ชันทำงานเชื่อมโยงกันเป็นระบบคิวแบบ FIFO อย่างสมบูรณ์

```
int main() {
    printf("CIRCULAR QUEUE PROGRAM...\n");
    printf("==========================\n");

    // initialize Q array to 0 (optional)
    for (int i = 0; i < N; i++) Q[i] = 0;

    ch = ' ';
    while (ch != 'E' && ch != 'e') {
        printf("\n[1=INSERT : 2=DELETE : E=Exit] : ");
        if (scanf(" %c", &ch) != 1) {
            // input failure
            break;
        }

        switch (ch) {
        case '1':
            printf("Insert Number : ");
            if (scanf("%d", &x) == 1) {
                insertCQ(x);
                ShowAllQueue();
```

```
            } else {
                printf("Invalid number input.\n");
                // clear invalid token
                int c;
                while ((c = getchar()) != '\n' && c != EOF) { }
            }
            break;

        case '2':
            x = deleteCQ();
            if (x != -1)
                printf("\nData from Queue = %d\n", x);
            ShowAllQueue();
            break;

        case 'E':
        case 'e':
            printf("Exiting...\n");
            break;

        default:
            printf("Unknown command. Use 1, 2 or E.\n");
            break;
        } // end switch
    } // end while

    return 0;
}
```

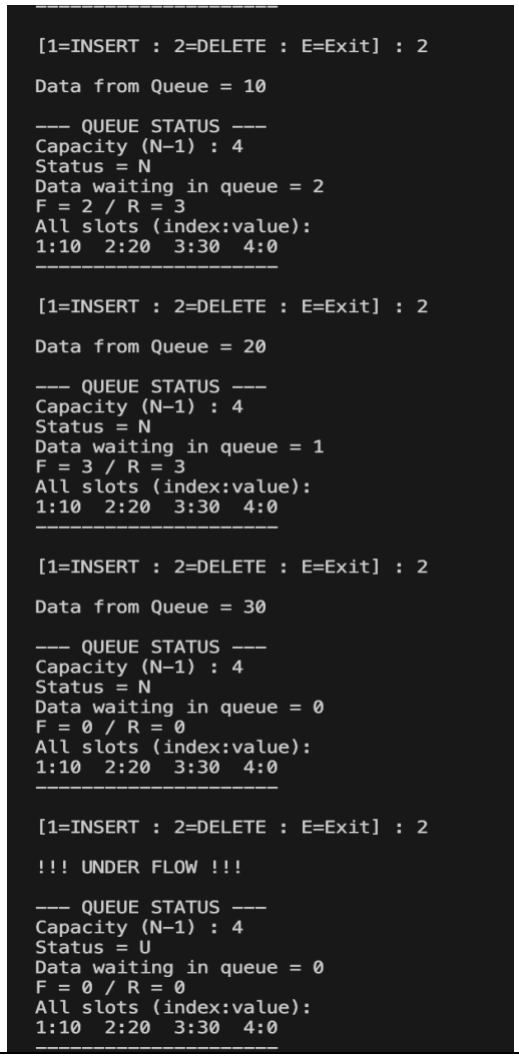
ผลลัพธ์ของโปรแกรม

การ Insert



```
CIRCULAR QUEUE PROGRAM...
========================

[1=INSERT : 2=DELETE : E=Exit] : 1
Insert Number : 20
You are queue number : 1

--- QUEUE STATUS ---
Capacity (N-1) : 4
Status = N
Data waiting in queue = 1
F = 1 / R = 1
All slots (index:value):
1:20  2:0  3:0  4:0
---------------------

[1=INSERT : 2=DELETE : E=Exit] : 1
Insert Number : e
Invalid number input.

[1=INSERT : 2=DELETE : E=Exit] : E
Exiting...
thanarukchumsawat@192 lab1 % ./queue_circular
CIRCULAR QUEUE PROGRAM...
========================

[1=INSERT : 2=DELETE : E=Exit] : 1
Insert Number : 10
You are queue number : 1

--- QUEUE STATUS ---
Capacity (N-1) : 4
Status = N
Data waiting in queue = 1
F = 1 / R = 1
All slots (index:value):
1:10  2:0  3:0  4:0
---------------------

[1=INSERT : 2=DELETE : E=Exit] : 1
Insert Number : 20
You are queue number : 2

--- QUEUE STATUS ---
Capacity (N-1) : 4
Status = N
Data waiting in queue = 2
F = 1 / R = 2
All slots (index:value):
1:10  2:20  3:0  4:0
---------------------
```

การ Delete



```
---------------------

[1=INSERT : 2=DELETE : E=Exit] : 2

Data from Queue = 10

--- QUEUE STATUS ---
Capacity (N-1) : 4
Status = N
Data waiting in queue = 2
F = 2 / R = 3
All slots (index:value):
1:10  2:20  3:30  4:0
---------------------

[1=INSERT : 2=DELETE : E=Exit] : 2

Data from Queue = 20

--- QUEUE STATUS ---
Capacity (N-1) : 4
Status = N
Data waiting in queue = 1
F = 3 / R = 3
All slots (index:value):
1:10  2:20  3:30  4:0
---------------------

[1=INSERT : 2=DELETE : E=Exit] : 2

Data from Queue = 30

--- QUEUE STATUS ---
Capacity (N-1) : 4
Status = N
Data waiting in queue = 0
F = 0 / R = 0
All slots (index:value):
1:10  2:20  3:30  4:0
---------------------

[1=INSERT : 2=DELETE : E=Exit] : 2

!!! UNDER FLOW !!!

--- QUEUE STATUS ---
Capacity (N-1) : 4
Status = U
Data waiting in queue = 0
F = 0 / R = 0
All slots (index:value):
1:10  2:20  3:30  4:0
---------------------
```