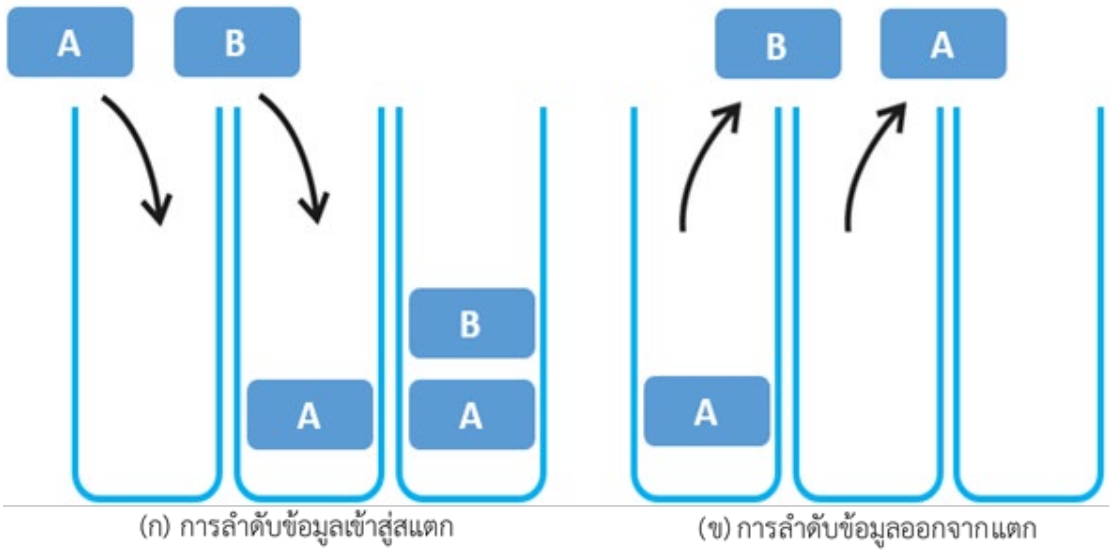
บทที่ 2 การประยุกต์ใช้โครงสร้างอาเรย์ (Array Application)

2.1 สแตก (Stack)

2.1.1 ลักษณะของสแตก

เป็นการประยุกต์ใช้โครงสร้างข้อมูลอาเรย์อย่างหนึ่ง ซึ่งมีกลไกการทำงานที่เป็น ลักษณะจำเพาะ กล่าวคือ

1. จะเก็บข้อมูลชนิดและขนาดเดียวกัน
2. มีการนำข้อมูลเข้าซึ่งเรียกว่า พุช (Push)
3. มีการนำข้อมูลออกซึ่งเรียกว่า ป๊อบ (Pop)
4. หากข้อมูลจะล้น ไม่สามารถนำข้อมูลเข้า (Push) ได้ จะแจ้งให้ผู้ใช้ทราบ เรียกว่าเกิดการล้น (Over Flow)
5. หากทำการนำข้อมูลออก (Pop) แต่ข้อมูลไม่มีในสแตก จะแจ้งให้ผู้ใช้ทราบเรียกว่าเกิดการหมด (Under Flow)
6. ข้อมูลที่เข้าทีหลังจะถูกนำออกมาก่อน เรียกพฤติกรรมนี้ว่า เข้าทีหลังออกก่อน (Last-In First-Out : **LIFO**)



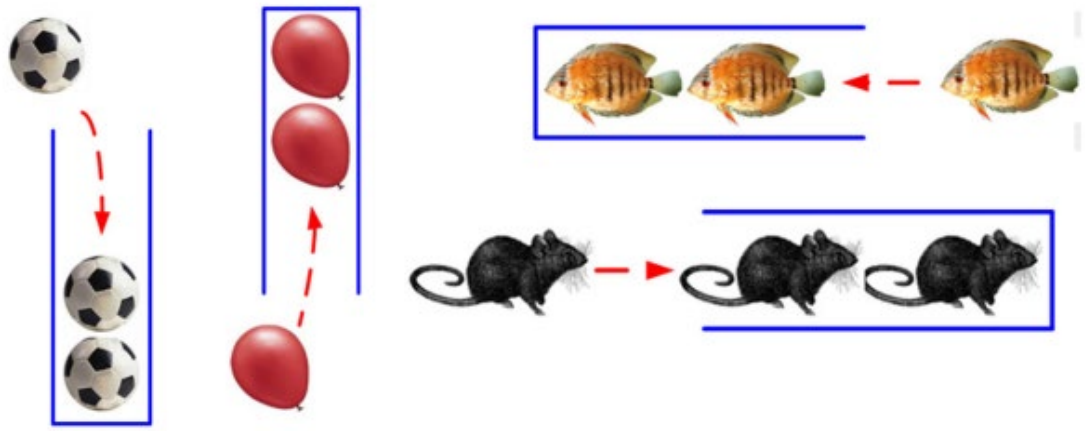
รูปที่ 2.1 แสดงลักษณะของสแตก

รูปที่ 2.2 สิ่งของรอบตัวที่มีการเก็บสแตก

สแตกเป็นกลไกที่เรียบง่ายแต่มีประโยชน์มากในหลาย ๆ ด้าน เช่น

1. ช่วยจัดการข้อมูลที่เป็นไปตามเทคนิค เข้าทีหลังออกก่อน LIFO
2. ใช้เก็บตำแหน่งหน่วยความจำกรณีที่มีการเรียกโปรแกรมย่อย(Sub Program) เพื่อให้สามารถกลับมาทำงานต่อได้
3. ใช้ในการอินเตอร์รัพท์ (Interrupt) ของระบบปฏิบัติการ
4. ใช้สำหรับการจัดการหน่วยความจำและการจัดสรรคืนอย่างเป็นระบบ
5. ใช้เก็บผลลัพธ์บางส่วนของการทำงานในกลไกการเรียกตนเองของโปรแกรม เรียกตนเอง (Recursive)
6. ใช้ในกลไกการแปลงนิพจน์อินฟิกไปเป็นโพสฟิก (Infix/Postfix) หรือหาค่าของนิพจน์โพสฟิก
7. ใช้เก็บตำแหน่งโหนดเพื่อช่วยในการเดินทางย้อนกลับของโครงสร้างลิงค์ ลิสต์ ต้นไม้ กราฟ (Linked List, Tree, Graph)
8. ใช้ในการเก็บค่าชั่วคราวของการจัดเรียงข้อมูล (Sorting)
9. อื่น ๆ อีกมากมาย

โดยทั่วไปแล้ว สแตกจะเป็นโครงสร้างข้อมูลสำหรับใช้เก็บค่าชั่วคราวเพื่อรอการ ประมวลผลหาโปรแกรมทำงานแล้วเสร็จ สแตกจะไม่มีข้อมูลใด ๆ หลงเหลืออยู่ และโดย ธรรมชาติแล้ว การจัดวางทิศทางของสแตกจะอยู่ในแนวไหนก็ได้ (แนวนอน แนวตั้ง แนวคว่ำ) ขึ้นอยู่กับความเหมาะสมในการใช้งาน ดังรูป 2.3



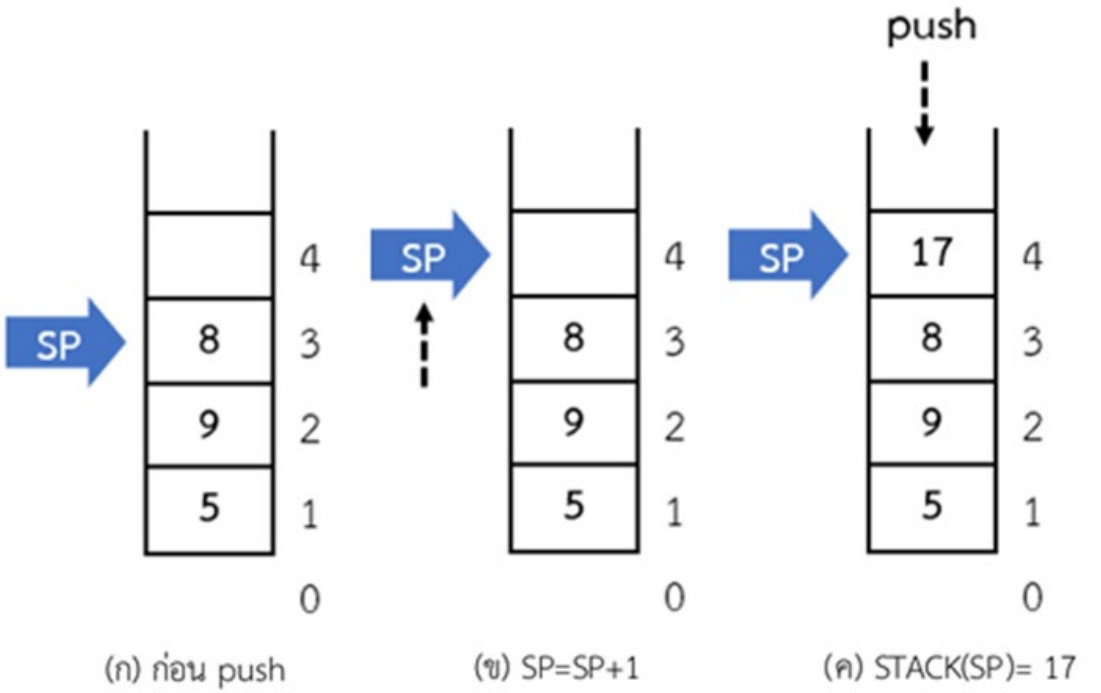
รูปที่ 2.3 แสดงทิศทางของโครงสร้างสแตก

2.1.2 การสร้างสแตก

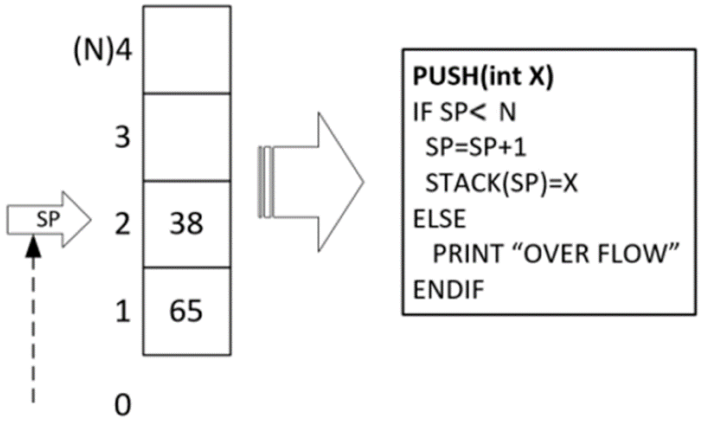
โครงสร้างสแตกประกอบด้วยตัวสแตกซึ่งสามารถใช้อาเรย์แทนได้ และตัวที่ระบุว่า ค่าที่อยู่ส่วนบนสุดของสแตกคือค่าใด คือสแตกพอยน์เตอร์ (Stack Pointer : **SP**) นอกจากนี้ ยังต้องการโปรแกรมย่อยอีก 2 โปรแกรม เพื่อนิยามกิริยา **"พุช"** (Push) และ **"ป๊อบ"** (Pop) สำหรับสแตก

1. สแตกพอยเตอร์ : จะถูกเรียกย่อๆว่า SP โครงสร้างสแตกซึ่งมีปลายเปิดข้างเดียว จำเป็นต้องมี SP ชี้ไปยังค่าที่อยู่บนสุดของสแตก

2. PUSH : เป็นการกระทำที่นำข้อมูลเข้าสู่สแตก ก่อนที่จะนำข้อมูลสู่สแตกได้ต้องให้ SP ชี้ไปยังช่องว่างถัดไปก่อน จากนั้นจึงสามารถนำข่าวสารเข้าสู่สแตกที่ตำแหน่ง SP ได้ ตามขั้นตอนและรูปแบบคำสั่งดังรูป 2.4

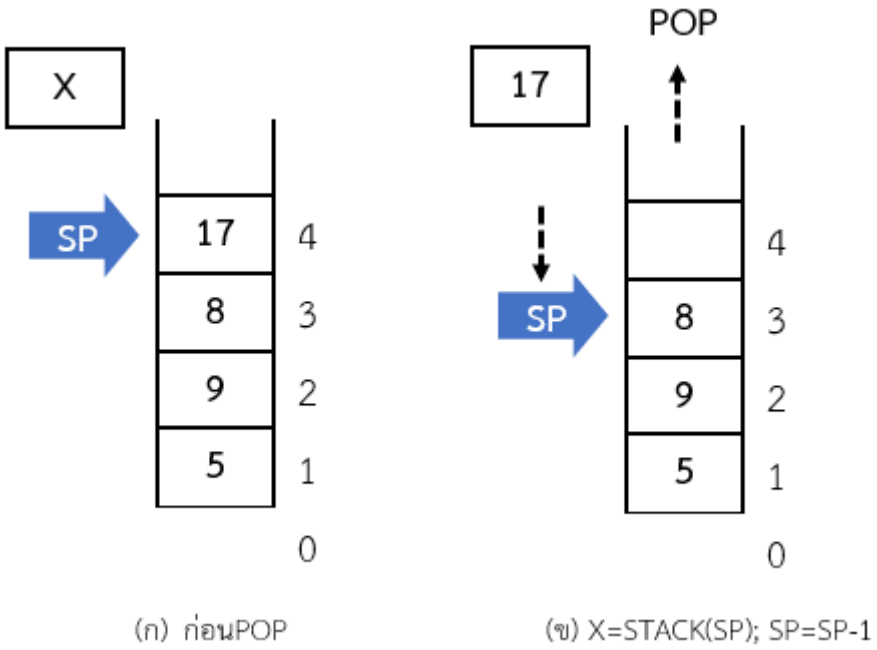


รูปที่ 2.4 แสดงขั้นตอนการ PUSH ค่าเข้าสู่สแตก



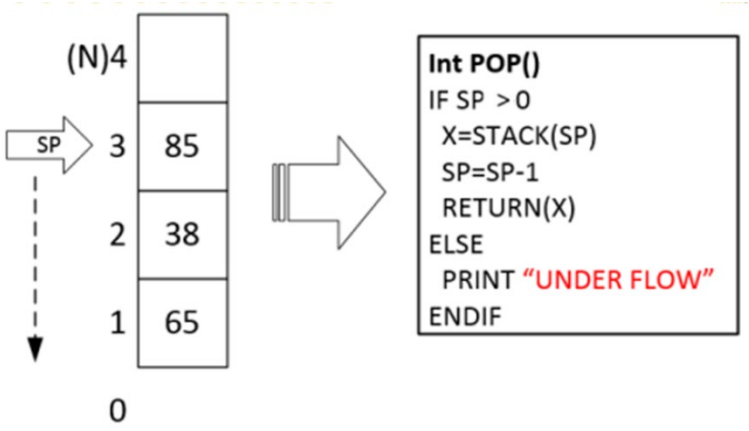
รูปที่ 2.5 แสดงรูปแบบคำสั่งการ PUSH

3. POP : เป็นกิริยาที่ทำต่อสแตกเพื่อนำข้อมูลที่อยู่บนสุดของสแตกออกมา โดยตอน เริ่มต้น SP จะชี้ไปยังตำแหน่งที่ 3 (ข้อมูล 17) หลังจาก POP แล้ว SP จะชี้ไปยัง ตำแหน่งที่ 2 (ข้อมูล 9) ส่วนค่าที่ถูก POP ออกมาจะไปอยู่ในในแปร X ตามขั้นตอน และรูปแบบคำสั่งดังรูป 2.6



(ก) ก่อนPOP (ข) X=STACK(SP); SP=SP-1

รูปที่ 2.6 แสดงขั้นตอนการ POP ค่าออกจากสแตก

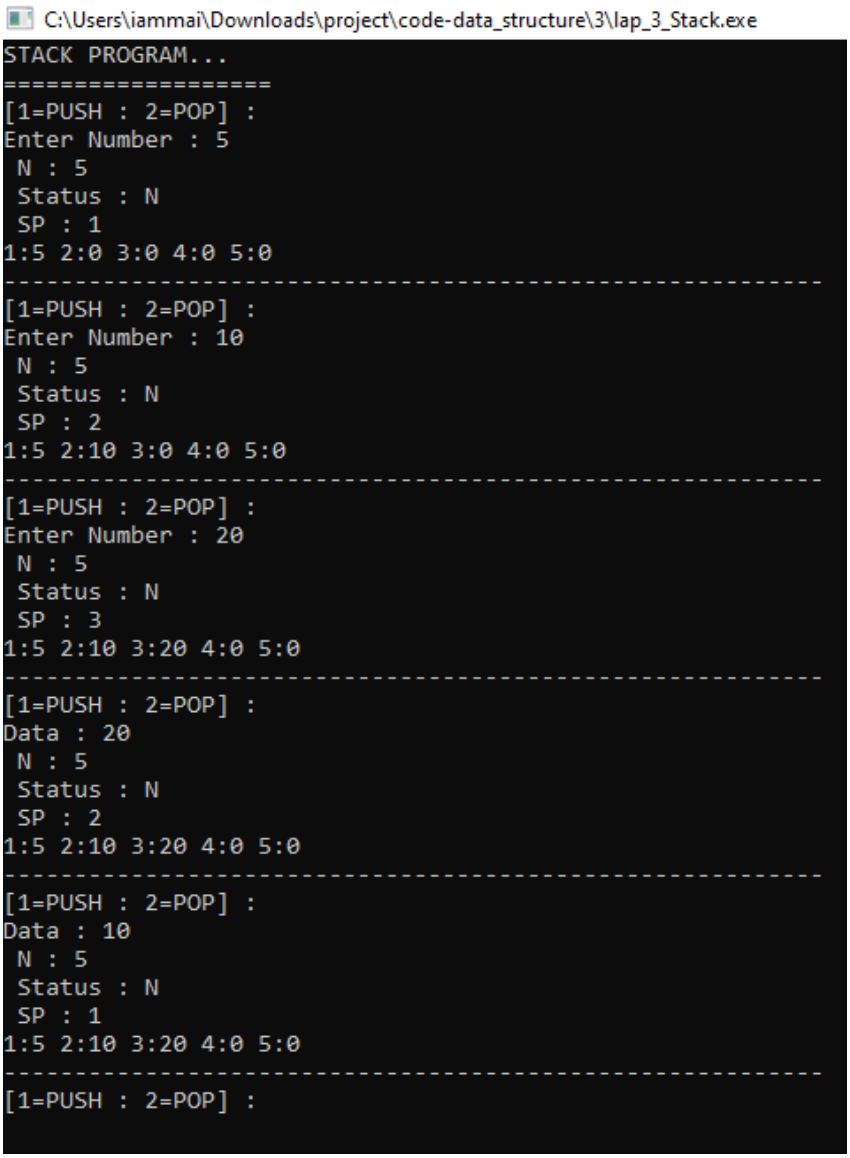


รูปที่ 2.7 แสดงรูปแบบคำสั่งการ POP

2.1.2.1 โปรแกรมสแตก

```
/*
Program create PUSH/POP function of Stack and use its.
The program will exit when status are "OVER FLOW" or "UNDER FLOW".
================================================================================
*/
#include <stdio.h> //use printf()
#include <conio.h> //use getch()
#define MaxStack 6 //Set Max Stack
int stack[MaxStack]; //Declare Max Stack 0..5
int x; //Temperature variable
int SP = 0; //Initial SP=0
char status = 'N'; //Initial Status = NORMAL
char ch; //KBD Read variable

void push(int x) //PUSH Function
{
    if(SP == MaxStack-1){ //Check Stack FULL?
        printf("!!!OVER FLOW!!!...\n");
        status='O'; //set status = OVER FLOW
    }
    else{
        SP=SP+1; //Increase SP
        stack[SP]=x; //Put data into Stack
    }
}

int pop() //POP Function
{
    int x;
    if (SP != 0) //Check Stack NOT EMPTY?
    {
        x=stack[SP]; //Get data from Stack
         SP--; //Decrease SP
        return(x); //Return data
    }
    else
    {
        printf("\n!!!UNDER FLOW!!!...\n");
        status = 'U'; //set STATUS = "UNDER FLOW"
    }
}

void ShowAllStack() //Display Function
```

```
{
    int i; //Counter variable
    printf(" N : %d\n ",MaxStack-1); //Display N
    printf("Status : %c\n ",status); //Display STATUS
    printf("SP : %d\n",SP); //Display SP
    for ( i = 1; i < MaxStack; i++ )
    {
        printf("%d:%d ",i, stack[i]); //Display all of data in Stack
    }
    printf("\n-------------------------------------------------------
-\n");
}

int main()
{
    printf("STACK PROGRAM...\n");
    printf("==================\n");
    while (status == 'N')
    {
        printf("[1=PUSH : 2=POP] : "); //Show MENU
        ch = getch(); //Wait and read KBD with out ENTER Press
        switch(ch) //Check ch
    {
    case '1' : printf("\nEnter Number : ");
        scanf("%d", &x); //Read data from KBD
        push(x); //Call PUSH Function
        ShowAllStack(); //Display all data in Stack
        break;
    case '2' : x=pop(); //POP data
        printf("\nData : %d\n",x); //Display it
        ShowAllStack(); //Display all data in Stack
        break;
    } //End SWITCH CASE
} //End WHILE Loop
printf("\n"); //line feed
return(0);
}//End MAIN Fn.
```



รูปที่ 2.8 แสดงผลการทำงานของโปรแกรมสแตก

2.2 คิว (Queue)

2.2.1 ลักษณะของคิว

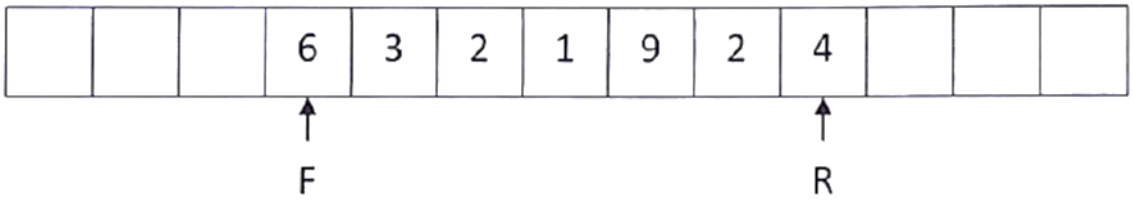
โครงสร้างแบบคิวเป็นโครงสร้างที่ปรากฏทั่วไปโดยเฉพาะอย่างยิ่งใน ระบบปฏิบัติการในคอมพิวเตอร์ ในระบบโทรคมนาคม รวมทั้งในระบบการทดลองเทียม ลักษณะของคิวเหมือน แถวรอคอย หรือแถวคิว นั่นเอง ดังรูป 2.9

รูปที่ 2.9 ตัวอย่างของคิวธรรมดาในชีวิตจริง

คุณสมบัติที่เด่นชัดของคิวคือ อะไรที่ เข้าไปในคิวก่อนย่อมออกจากคิวมาก่อน คุณสมบัติข้อนี้ทำให้คิวได้อีกชื่อหนึ่งว่าเป็น โครงสร้างแบบเข้าก่อน-ออกก่อน(First-In First-Out : **FIFO**)

2.2.2 คิวธรรมดา (Queue)

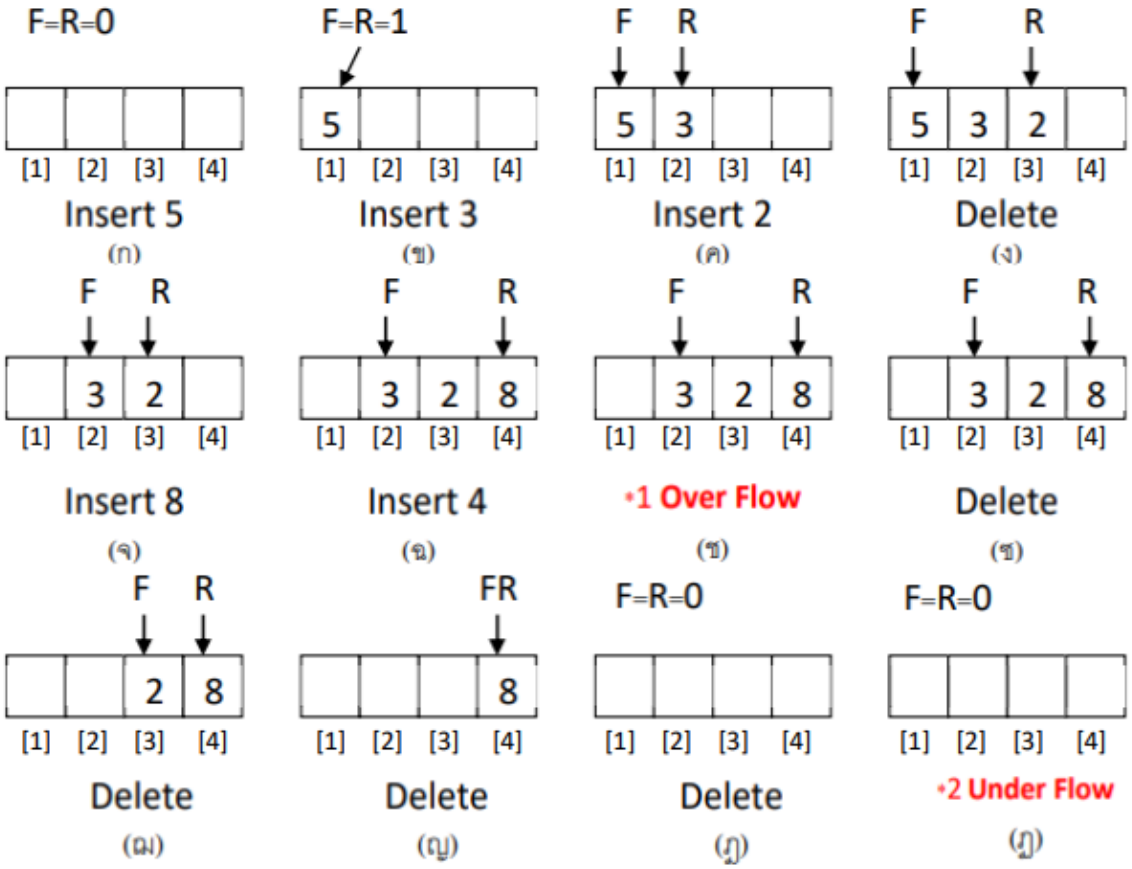
สร้างโดยใช้อาเรย์เป็นตัวคิวพร้อมกันได้นั้น ต้องใช้พอยน์เตอร์อีก 2 ตัวชื่อ **F** (front pointer) และ **R** (rear pointer) โดยข้อมูลจะเข้าไปทาง R และจะออกจากคิวทาง F ในตอนแรก ตัวพอยน์เตอร์ F จะชี้ไปยังหัวแถว (ตัวแรก) และพอยน์เตอร์ R จะชี้ไปที่หางแถว (ตัวสุดท้าย) การเลื่อนพอยน์เตอร์ R จะเลื่อนได้จนสุดอาเรย์โดยไม่มีการวก กลับ ดังรูป 2.10



รูปที่ 2.10 โครงสร้างของคิวธรรมดา

2.2.2.1 การทำงานของคิวธรรมดา

ในตอนแรกซึ่งคิวยังว่างเปล่าอยู่นั้น F และ R มีค่า 0 ทั้งคู่ สิ่งที่ต้องการกระทำกับคิว คือการนำข้อมูลเข้าสู่คิว (insertion) และการนำข้อมูลออกจากคิว (deletion) ลำดับ เหตุการณ์การเข้าสู่คิวและออกจากคิว ดังรูป 2.11



***1** ไม่สามารถ **Insert** ได้เพราะคิวเต็ม ***2** ไม่สามารถ **Delete** ได้เพราะคิวว่าง

รูปที่ 2.11 แสดงการทำงานของคิวธรรมดา

การแจ้งเตือนกรณีไม่สามารถนำข้อมูลเข้า-ออกจากคิว

การนำข้อมูลเข้าสู่คิวในขณะที่คิวเต็มหรือไม่มีที่ว่างด้าน R ถ้าพยายามทำจะต้องแจ้งข้อความว่า คิวเต็ม (Overflow) ส่วนในการนำข้อมูลออกจากคิว จะไม่สามารถนำอะไรออก จากคิวที่ว่างเปล่า (เนื่องจากไม่มีข้อมูลอยู่) ถ้าพยายามทำจะจะต้องแจ้งข้อความว่า คิวหมด (Underflow)

จากการพิจารณาขั้นตอนข้างต้น สามารถสรุปเป็นชุดคำสั่งของคิวธรรมดาได้เป็น ดังรูป 2.12

```
INSERT : PROCEDURE (Y:Integer)
IF R=N THEN
   PRINT "Over flow"
ELSE
   R=R+1
   IF F=0 THEN
      F=1
   ENDIF
      Q(R) = Y
ENDIF
END : INSERT
```

```
DELETE : PROCEDURE;
Y : Integer;
Y=NULL
IF F=0  THEN
   PRINT "Under flow"
ELSE
   Y=Q(F)
   IF  F=R THEN
      F=0 :R=0
   ELSE
      F=F+1
   ENDIF
ENDIF
RETURN (Y)
END : DELETE
```

รูปที่ 2.12 แสดงรูปแบบคำสั่งการ INSERT/DELETE ของคิวธรรมดา

2.2.2.2 โปรแกรมคิวธรรมดา

```
/*
Program create INSERT/DELETE function of Queue and use its.
The program will exit when status are "OVER FLOW" or "UNDER FLOW".
=================================================================*/
#include <stdio.h> //use printf()
#include <conio.h> //use getch()
#define N 5 //Set Max Queue
int Q[N]; //Prepare Queue 0..N-1
int x, F = 0,R = 0; //Declare x and initial Front/Rear variable
char status = 'N'; //Initial Status = NORMAL
char ch; //KBD Read variable

void insertQ(int y) //INSERT Function
{
    if(R == N-1) //Check Queue FULL?
    {
        printf("!!!OVER FLOW!!!...\n");
        status='O'; //set status = OVER FLOW
    }
    else
    {
        R=R+1; //Increase R
        if(F==0)
        {
```

```
            F++;
        }
    Q[R]=y; //Put data into Queue
    }
}

int deleteQ() //DELETE Function
{
    int y;
    if (F != 0) //Check QUEUE NOT EMPTY?
    {
        y=Q[F]; //Get data from Queue
        if (F==R)
        {
            F=0; R=0;
        }
        else
        {
            F++;
        }
        return(y); //Return data
    }
    else
    {
        printf("\n!!!UNDER FLOW!!!...\n");
        status = 'U'; //set STATUS = "UNDER FLOW"
    }
}

void ShowAllQueue() //Display Function
{
    int i; //Counter variable
    printf("N : %d\n",N-1);
    printf("Status : %c\n ",status); //Display STATUS
    printf("F : %d R : %d\n",F,R); //Display F&R
    for (i=1;i<N;i++)
    {
        printf("%d:%d / ",i, Q[i]); //Display all of data in QUEUE
    }
    printf("\n-----------------------------------------------------------
-\n");
}

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

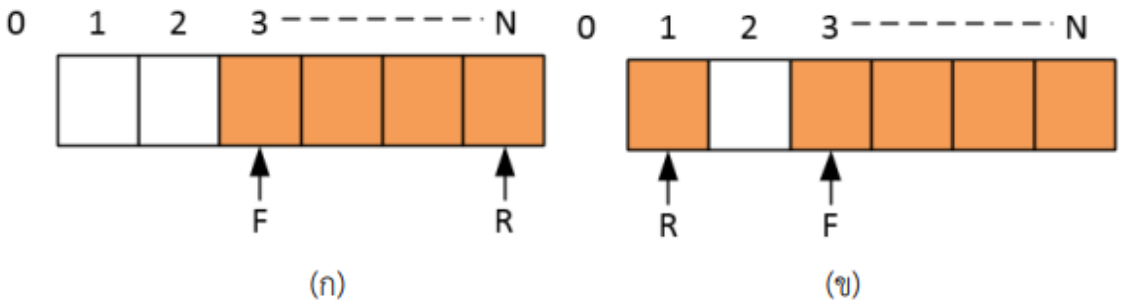
```
    printf("======================\n");
    while (status == 'N')
    {
        printf("[1=INSERT : 2=DELETE] : "); //Show MENU
        ch = getch(); //Wait and read KBD with out ENTER Press
        switch(ch) //Check ch
        {
            case '1' : printf("\nInsert Number : ");
                scanf("%d", &x); //Read data from KBD
                insertQ(x); //Call INSERTNQ Function
                ShowAllQueue(); //Display all data in Queue
                break;
            case '2' : x=deleteQ(); //Delete data
                printf("\nData from Queue = %d\n",x); //Display it
                ShowAllQueue(); //Display all data in Queue
                break;
        } //End SWITCH CASE
    } //End WHILE Loop
    printf("\n"); //line feed
    return(0);
}//End MAIN Fn.
```

```
======================
[1=INSERT : 2=DELETE] :
Insert Number : 10
N : 4
Status : N
 F : 1 R : 1
1:10 / 2:0 / 3:0 / 4:0 /
------------------------------------------------------------
[1=INSERT : 2=DELETE] :
Insert Number : 20
N : 4
Status : N
 F : 1 R : 2
1:10 / 2:20 / 3:0 / 4:0 /
------------------------------------------------------------
[1=INSERT : 2=DELETE] :
Insert Number : 30
N : 4
Status : N
 F : 1 R : 3
1:10 / 2:20 / 3:30 / 4:0 /
------------------------------------------------------------
[1=INSERT : 2=DELETE] :
Data from Queue = 10
N : 4
Status : N
 F : 2 R : 3
1:10 / 2:20 / 3:30 / 4:0 /
------------------------------------------------------------
[1=INSERT : 2=DELETE] :
```

รูปที่ 2.13 แสดงผลการทำงานของโปรแกรมคิวธรรมดา

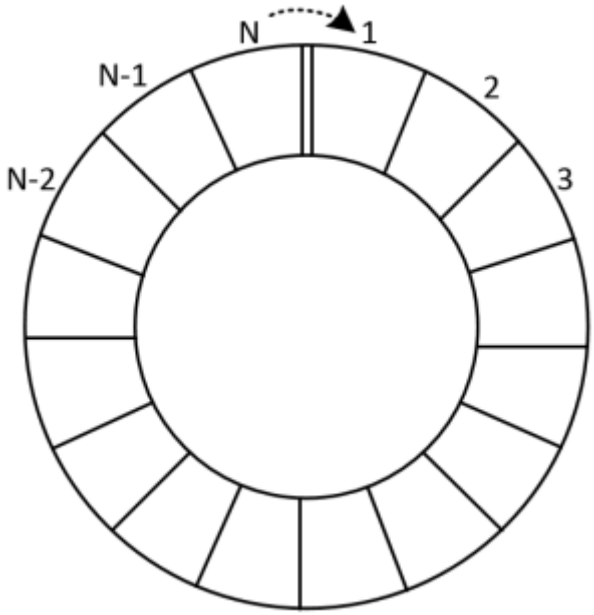
2.2.3 คิววงกลม (Circular Queue)

ในคิวธรรมดา ถ้า R ชี้ที่ช่องสุดท้ายของคิว การเพิ่มข้อมูลเข้าในคิวอีกจะทำไม่ได้ (คิวเต็ม) แม้ว่ายังมีเนื้อที่เหลือด้านหน้าก็ตามทำให้การใช้เนื้อที่ไม่เต็มที่ ทางแก้ไขก็คือยอม ให้เนื้อที่ส่วนหน้าถูกใช้บรรจุข้อมูลได้ หรือถือเสมือนว่าคิวนี้เป็นวงกลมดังรูป 2.14 แสดงคิววงกลมเมื่อเพิ่มข้อมูลเข้าอีกหนึ่งข้อมูล R จะเปลี่ยนไปชี้ที่ 1 เพื่อแสดงตำแหน่งสมาชิก สุดท้ายของคิว



รูปที่ 2.14 (ก) แสดงลักษณะการชี้เมื่อ R ชี้ที่ N (ข) แสดงลักษณะการชี้เมื่อ R วนกลับไปชี้ที่ 1

จัดอาเรย์ให้ทำงานเป็นคิววงกลมดังกล่าว ทำได้โดยให้ส่วนท้ายของคิวที่ตำแหน่ง Q(N) ไปต่อกับส่วนแรกในตำแหน่ง Q(1) โครงสร้างคิวเช่นนี้เรียกว่าคิววงกลม (Circular Queue) ดังปรากฏดังรูป 2.15 และ ตัวอย่างของคิววงกลมในชีวิตจริงดังรูป 2.16



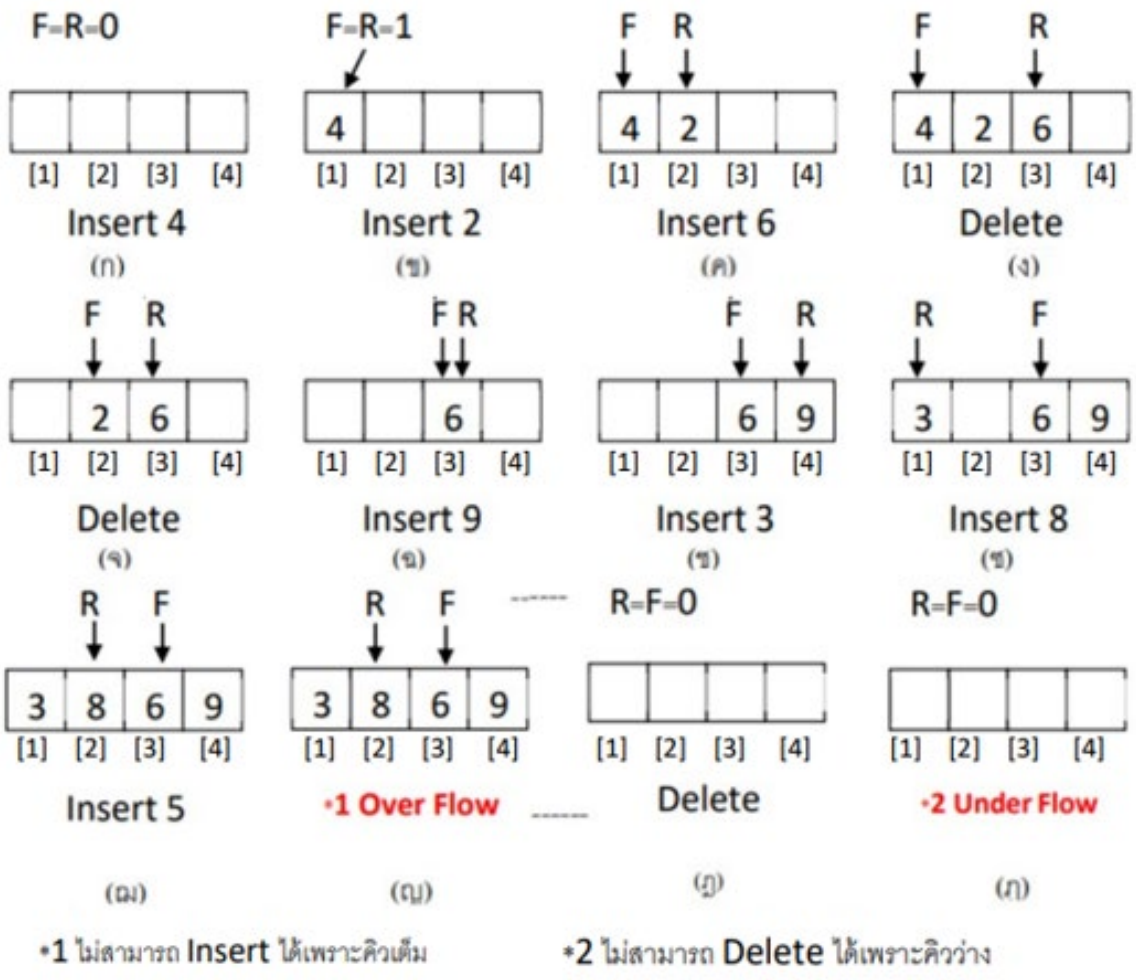
รูปที่ 2.15 แสดงลักษณะคิววงกลม

- เครื่องปิดฝาขวด
- ไฟจราจร
- เดือนในหนึ่งปี : ม.ค., ... ,ธ.ค.,-> ม.ค.
- วันในหนึ่งสัปดาห์: จันทร์-อาทิตย์->จันทร์
- เวลาชั่วโมงในหนึ่งวัน

รูปที่ 2.16 ตัวอย่างของคิววงกลมในชีวิตจริง

2.2.3.1 การทำงานของคิววงกลม

ในตอนแรกซึ่งคิวยังว่างเปล่าอยู่นั้น F และ R มีค่า 0 ทั้งคู่ สิ่งที่ต้องการคือการนำข้อมูลเข้าสู่คิว (Insertion) และการนำข้อมูลออกจากคิว (deletion) ลำดับดังรูป 2.17



รูปที่ 2.17 แสดงการทำงานของคิววงกลม

จากการพิจารณาขั้นตอนข้างต้น สามารถสรุปเป็นชุดคำสั่งของคิววงกลมได้เป็น ดังรูป 2.18

```
PROCEDURE INSERT(X: Integer)
IF  (R=F-1) OR (R=N AND F=1) THEN
   PRINT "Over Flow!!!"
ELSE
   IF (R=N) THEN
      R=1
   ELSE
      R=R+1
      IF F=0 THEN
         F=1
      ENDIF
   ENDIF
   Q(R) = X
ENDIF
END : INSERT
```

```
PROCEDURE DELETE()
Y : Integer
Y=NULL
If F=0 Then
   PRINT "Under Flow!!!"
ELSE
   Y=Q(F)
   IF F=R THEN
      F=0 : R=0
   ELSE
      IF F=N THEN
         F=1
      ELSE
         F=F+1
      ENDIF
   ENDIF
ENDIF
RERURN (Y)
END : DELETE
```

รูปที่ 2.18 แสดงรูปแบบคำสั่ง INSERT และ DELETE ของคิววงกลม

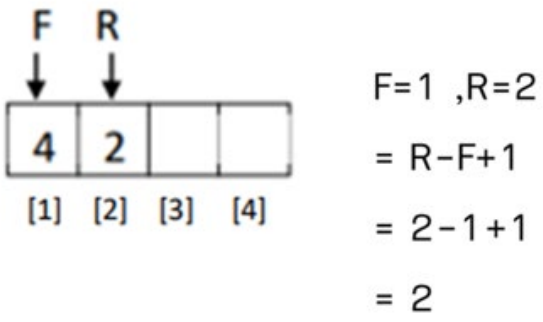
2.2.3.2 การคำนวณข้อมูลค้างในคิว

สิ่งที่ต้องคำนึงอีกประการหนึ่งของการใช้งานคิววงกลม คือ **"จำนวนข้อมูลที่ค้างในคิว"** เพื่อใช้เป็นข้อมูลในการบริหารจัดการคิวให้เกิดประสิทธิภาพ ตัวอย่างเช่น การใช้เครื่อง เข้าคิวของธนาคารต่าง ๆ (Queue Machine) หรือเคาน์เตอร์เซอร์วิสต่างๆ (Counter Service) ระบบจะแจ้งว่าเป็นคิวที่เท่าไร และมีคนรอในคิวกี่คน ให้ผู้มาใช้บริการได้ทราบ ข้อมูลอย่างครบถ้วนเพื่อเป็นข้อมูลในการประมาณเวลาที่ต้องรอเข้ารับบริการ

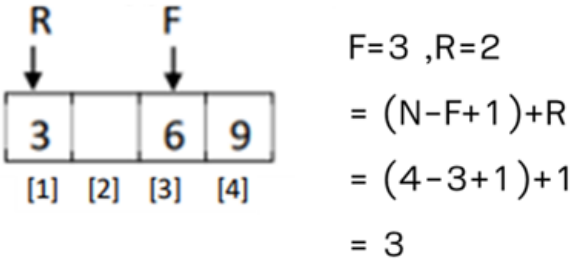
การคำนวณลำดับคิวนั้นให้นับลำดับตั้งแต่ข้อมูลตัวแรกที่นำเข้าสู่คิวให้เป็นลำดับที่ 1 และนับต่อเนื่องไปเรื่อย ๆ ส่วนการคำนวณข้อมูลที่รอในคิวจะต้องพิจารณาจำนวนข้อมูลใน อาเรย์ โดยมีสมการและเงื่อนไขดังนี้

กรณี F<=R : จำนวนข้อมูลรอในคิว = R-F+1

กรณี F>R : จำนวนข้อมูลรอในคิว = (N-F+1)+R



รูปที่ 2.19 การคำนวณลำดับคิว **กรณี F<=R**



รูปที่ 2.20 การคำนวณลำดับคิว **กรณี F>R**

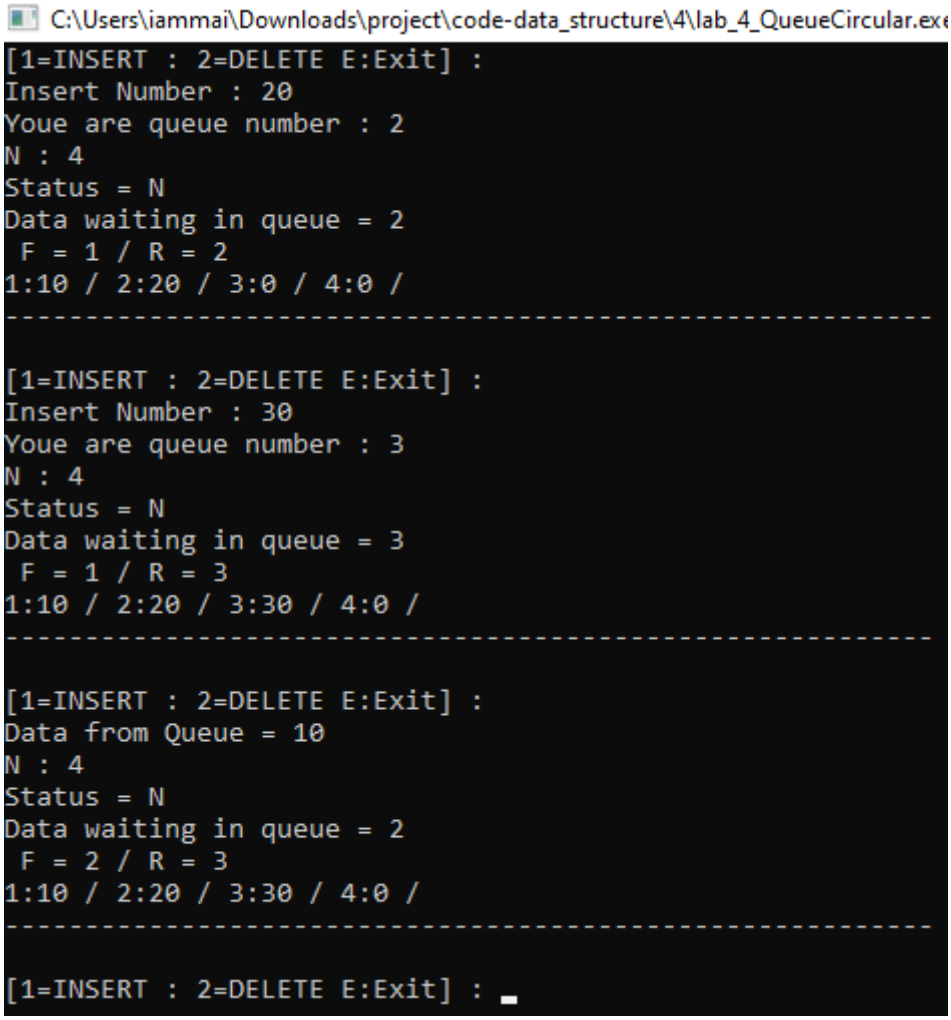
2.2.3.3 โปรแกรมคิววงกลม

```
/*
Program create INSERT/DELETE function of Circular Queue and use its
by...
1. Display queue value
2. Display queue status (N:NORMAL/ O:OVER FLOW/ U:UNDER FLOW)
3. Display Data waiting in queue
4. Display F and R
5. Display all data in queue
The program will exit when press "E" character.
================================================================================
======*/
#include <stdio.h> //use printf()
#include <conio.h> //use getch()
#define N 5 //Set Max Queue
int Q[N]; //Prepare Queue 0..N-1
int x, Qnumber = 0, F = 0,R = 0; //Declare x and initial
Qnumber/Front/Rear variable
char status = 'N'; //Initial Status = NORMAL
char ch; //KBD Read variable
```

```
void insertCQ(int y) //INSERT Function
{
    if((R==F-1) || (R==N-1 && F==1) ) //Check Queue FULL?
    {
        printf("!!!OVER FLOW!!!...\n");
        status='O'; //set status = OVER FLOW
    }
    else
    {
        if(R==N-1) // Loop back of R to 1 if it maximum
        {
            R=1;
        }
        else
        {
            R++; // increase R if Normal
            if(F==0) //if F is zero set it to 1
            F=1;
        }
        Qnumber++; //Increase queue number
        printf("Youe are queue number : %d\n", Qnumber);
    //Display queue number
        Q[R]=y; //Put data into Queue
        status = 'N'; //Set status to "NORMAL"
    }
}
int deleteCQ() //DELETE Function
{
    int y;
    if (F == 0)
    {
        printf("\n!!!UNDER FLOW!!!...\n");
    status = 'U'; //set STATUS = "UNDER FLOW"
    }
    else
    {
        y=Q[F]; //Get data from Queue
        if (F==R) //Set both to 0 if F and R are same value
        {
            F=0; R=0;
        }
        else
        {
            if(F==N-1) //Set F to 1 if F is maximum, otherwise increase
F
                F=1;
```

```
            else
                F++;
        }
        status = 'N'; //Set status to "NORMAL"
        return(y); //Return data
    }
}
int DataInQueue() // Calculate Data waiting in queue
{
    int y=0;
    if (F!= 0 && R!=0) //if not equal then can calculate
    {
        if (F<=R)
            y=R-F+1; //Normal F and R
        else
            y=(N-1)-F+1+R; //incase loop of R
    }
        return(y);
}
void ShowAllQueue() //Display Function
{
    int i; //Counter variable
    printf("N : %d\n",N-1);
    printf("Status = %c \n",status); //Display STATUS
    printf("Data waiting in queue = %d\n",DataInQueue());
//Display Data waitting in queue
    printf(" F = %d / R = %d\n",F,R); //Display F R
    for ( i = 1; i < N; i++ )
    {
        printf("%d:%d / ",i, Q[i]); //Display all of data in QUEUE
    }
    printf("\n-----------------------------------------------------------
-\n");
}
int main()
{
    printf("CICULAR QUEUE PROGRAM...\n");
    printf("=======================\n");
    ch=' ';
    while (ch != 'E')
    {
        printf("\n[1=INSERT : 2=DELETE E:Exit] : "); //Show MENU
        ch = getch(); //Wait and read KBD with out ENTER Press
        switch(ch) //Check ch
        {
            case '1' : printf("\nInsert Number : ");
```

```
                scanf("%d", &x); //Read data from KBD
                insertCQ(x); //Call INSERTNQ Function
                ShowAllQueue(); //Display all data in Queue
                break;
            case '2' : x=deleteCQ(); //Delete data
                printf("\nData from Queue = %d\n",x); //Display it
                ShowAllQueue(); //Display all data in Queue
                break;
        } //End SWITCH CASE
    } //End WHILE Loop
    printf("\n"); //line feed
    return(0);
}//End MAIN Fn.
```



รูปที่ 2.21 แสดงผลการทำงานของโปรแกรมคิววงกลม