มหาวิทยาลัยราชภัฏนครปฐม

บทที่ 6
สแตกและคิว

ผู้บรรยาย : ผศ.ดร.ณัฐชามญฐ์ ศรีจำเริญรัตนา
สาขาวิชาคอมพิวเตอร์ธุรกิจ คณะวิทยาการจัดการ
มหาวิทยาลัยราชภัฏนครปฐม

สแตกและคิว

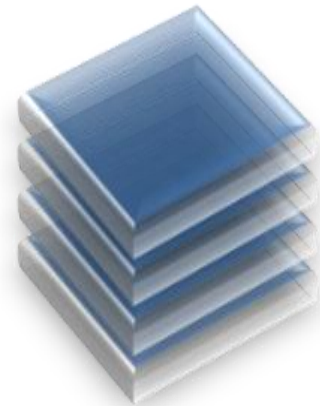
ในชีวิตประจำวันสามารถพบการทำงานลักษณะโครงสร้างข้อมูลแบบสแตก ได้โดยปกติ เช่น การวางหนังสือซ้อนกันในกล่องที่ต้องวางหนังสือทีละเล่มจนเต็มกล่อง และเมื่อกล่องถูกบรรจุหนังสือจนเต็มกล่องจนไม่สามารถวางหนังสือได้อีก และเมื่อต้องการหนังสือเล่มใดจำเป็นต้องหยิบหนังสือจากเล่มบนสุด นั่นคือหนังสือเล่มสุดท้ายที่บรรจุลงในกล่องจะถูกนำออกเป็นเล่มแรก และจะหยิบต่อไปจะถึงหนังสือเล่มที่ต้องการหรือหนังสือเล่มสุดท้ายที่บรรจุในกล่องนั่นเอง

6.1 สแตก

- สแตก (stack) เป็นโครงสร้างข้อมูลที่สามารถเพิ่มข้อมูลหรือลบข้อมูลจะทำที่ตำแหน่งปลายของข้อมูลสแตก (TOP) จึงมีการเรียกการทำงานลักษณะนี้ว่า "มาทีหลังแต่ออกก่อน" (Last In – First Out : LIFO)
- สมาชิกในโครงสร้างสแตกเป็นลักษณะรายการที่เรียงต่อกัน
- โอเปอร์เรเตอร์ที่ใช้ในการจัดการข้อมูลในสแตก ประกอบด้วย
  - การนำข้อมูลเพิ่มเข้าสู่สแตกเรียกว่า การพุชสแตก (push stack)
  - การนำข้อมูลออกจากสแตกเรียกว่า การป๊อปสแตก (pop stack)
  - ตัวชี้ข้อมูลของสแตกที่อยู่ด้านบนสุดจะเรียกว่า ทอปของสแตก (top of stack)
  - สมาชิกของสแตกเรียก สแตก อิลิเมนต์ (stack elements) ซึ่งข้อมูลภายในแต่ละตัวของสมาชิกจะกำหนดให้มีชนิดข้อมูลเป็นแบบเดียวกัน

6.1 สแตก(ต่อ)

ตัวอย่าง ภาพข้อมูลที่เรียงซ้อนกันในลักษณะสแตก



6.1 สแตก(ต่อ)

การดำเนินการเกี่ยวกับสแตก

โครงสร้างแบบสแตกมีระบบปฏิบัติเพื่อดำเนินการพื้นฐานซึ่งประกอบด้วย

- การสร้างสแตกใหม่ขึ้นเพื่อนำมาใช้งาน
- การเพิ่มข้อมูลเข้าไปในสแตก
- การดึงค่าหรือลบค่าออกจากสแตก

เพื่อแบ่งการทำงานแต่ละด้านดังแสดงวิธีการทำงานดังต่อไปนี้

6.1.1 การเพิ่มข้อมูลเข้าสแตก

การเพิ่มข้อมูลเข้าสแตก เรียกวิธีการดำเนินการนี้ว่าพุช (push) เป็นวิธีการนำข้อมูลใหม่เข้าสแตก การเพิ่มข้อมูลจะมีซูโดโค้ด (pseudocode) ตามขั้นตอนดังต่อไปนี้

ตรวจสอบสแตกเกินขนาดที่กำหนดหรือไม่ โดยเลือกการทำงานดังต่อไปนี้

1) ถ้าไม่เต็ม

- ปรับตัวชี้ TOP OF STACK ไปตำแหน่งใหม่
- เพิ่มข้อมูลเข้าสแตกในตำแหน่ง TOP OF STACK

2) ถ้าสแตกเต็ม

- แสดงข้อความ Stack Overflow

6.1 สแตก(ต่อ)

ตัวอย่างที่ 6.1 จงแสดงการนำข้อมูลเข้าสแตก

กำหนดให้สแตกมีขนาด 10 และต้องการเพิ่มข้อมูลต่อไปนี้ ได้แก่ 0,1,2,3,4,5,6,7,8,9 ตามลำดับ

โดยเริ่มต้นภายในสแตกมีข้อมูลอยู่ก่อนแล้ว 1 ตัว คือ 0

เริ่มต้น		Push 5		จบการทำงาน	
Max=10		Max=10		Max=TOP=10	9
					8
					7
					6
		TOP=6	5		5
			4		4
			3		3
			2		2
			1		1
TOP=1	0		0		0
1)		2)		3)	

6.1 สแตก(ต่อ)

ตัวอย่างที่ 6.2 จงเขียนโปรแกรมสำหรับการนำข้อมูลเพิ่มเข้าในสแตก

```
1   int push()
2   {
3           int n;
4            cout<<"\n Enter item in stack" ;
5           cin>>n;
6           if(top==size-1)
7           {
8               cout <<"\nStack is Full" ;
9           }else{
10                  top=top+1;
11                  stack[top]=n;
12          }
13  }
```

6.1.2 การลบข้อมูลออกจากสแตก

การลบข้อมูลออกจากสแตก มีชื่อเรียกวิธีการดำเนินการนี้ว่า pop ซึ่งเป็นวิธีการนำข้อมูลออกหรือลบข้อมูลออกจากสแตก โดยการทำงานกับข้อมูลนี้จะทำกับข้อมูลที่อยู่ด้านบนสุดของสแตก หรือตำแหน่ง TOP OF STACK ซึ่งมีขั้นตอนดังต่อไปนี้

ตรวจสอบสแตกมีสมาชิกหรือเป็นสแตกว่างหรือไม่ โดยเลือกการทำงานดังต่อไปนี้

1) ถ้าสแตกไม่ว่าง

- ทำการ pop ข้อมูลออกจากแสตก
- ปรับลดตำแหน่งตัวชี้ TOP OF STACK ไปตำแหน่งถัดไป

2) ถ้าสแตกว่าง

- แสดงข้อความ UNDERFLOW

การลบข้อมูลออกจากสแตกจะกระทำกับข้อมูลที่อยู่ด้านบนสุด

6.1 สแตก(ต่อ)

ตัวอย่างที่ 6.3 จงแสดงการนำข้อมูลออกจากสแตก

กำหนดให้สแตกมีขนาด 10 และในสแตกมีข้อมูลต่อไปนี้ ได้แก่ 0,1,2,3,4,5,6,7 ตามลำดับ

โดยต้อง pop ค่า 5 ออกจากสแตก มีวิธีการทำดังนี้

เริ่มต้น		Pop ค่า 7		Pop ค่า 6		Pop ค่า 5, จบการทำงาน	
Max=10		Max=10		Max= 10		Max= 10	
TOP=8	7						
	6	TOP=7	6				
	5		5	TOP=6	5		
	4		4		4	TOP=5	4
	3		3		3		3
	2		2		2		2
	1		1		1		1
	0		0		0		0

6.1 สแตก(ต่อ)

ตัวอย่างที่ 6.4 จงเขียนโปรแกรมสำหรับการนำข้อมูลออกจากในสแตก

```
1   void pop()
2   {
3           int item;
4           if(top==-1)
5           {
6                   cout <<"\n Stack Overflow" ;
7           }else{
8                       item=stack[top];
9                         cout <<"\n item popped is= "<<item ;
10                      top--;
11          }
12  }
```

6.1.3 การแสดงข้อมูลของสแตก

- สามารถดำเนินการโดยใช้ฟังก์ชั่น display( ) เพื่อนำข้อมูลออกมาแสดง
- สามารถใช้ฟังก์ชั่น while, for สำหรับวนการแสดงค่า และใช้การอ้างอิงตำแหน่งในสแตกร่วมกัน
- ลักษณะการดำเนินการของสแตก คือ ข้อมูลที่เข้ามาก่อนจะถูกนำออกทีหลัง

ฟังก์ชั่น display( ) นี้เป็นการนำข้อมูลออกมาแสดงผลเท่านั้น

โดยไม่ได้ทำการลบหรือเพิ่มข้อมูลใด ๆ เข้าสู่สแตก

❑ ตัวอย่างที่ 6.5 จงเขียนโปรแกรมสำหรับการแสดงข้อมูลในสแตก

1	void display()
2	{
3	cout <<"\n ==display data==" ;
4	for(int i=top; i>-1; i--)
5	cout<<"\nItem No."<<i+1<<"."<<stack[i];
6	}

6.1.4 การบอกตำแหน่งข้อมูลหรือตำแหน่งบนสุดของสแตก

- ฟังก์ชั่น stack_top เป็นตัวฟังก์ชั่นที่ใช้ตรวจสอบตำแหน่งของข้อมูลที่บรรจุอยู่ภายในสแตกและแสดงผลตำแหน่งที่สแตกชี้โดยการส่งค่าตำแหน่งบนสุดที่ตรวจสอบมาให้
- กรณีทำการปรับตัวชี้ TOP OF STACK ไปตำแหน่งใหม่ สามารถแบ่งการปรับตัวชี้ตำแหน่งได้ ดังนี้

1) ถ้าสแตกว่างและสแตกไม่มีข้อมูล ให้แสดงข้อความ “Stack Overflow”

2) ถ้าสแตกไม่ว่างและสแตกยังไม่เต็ม ให้แสดงข้อความ “TOP STACK IS”

3) ถ้าสแตกไม่ว่างและสแตกเต็ม ให้แสดงข้อความ “Stack is Full”

ตัวอย่างที่ 6.6 จงเขียนโปรแกรมสำหรับการบอกตำแหน่งข้อมูลของสแตก

```
1	void stack_top()
2	{
3			if(top==size-1)
4			{
5				cout <<"\nStack is Full" ;
6				}else{
7				if(top==-1)
8				{
9					cout <<"\n Stack Overflow" ;
10					}else{
11					cout<<"\n TOP STACK IS="<<top+1;
12				}
13			}
14	}
```

6.2 การประยุกต์ใช้งานสแตก

- การนำสแตกมาประยุกต์ใช้ในการคำนวณนิพจน์ทางคณิตศาสตร์ (arithmetic expression) เกิดขึ้นในในปี ค.ศ. 1950
- โดยนักตรรกะวิทยาชาวโปแลนด์ Jan Lukasiewicz พบว่าการเขียนสัญลักษณ์เติมหลังนั้นไม่มีความจำเป็นต้องใช้เครื่องหมายวงเล็บ และเรียกว่า สัญกรณ์โพลิช
- แต่เนื่องจากระบบคอมพิวเตอร์ในระบบการคอมไพเลอร์ค่อนข้างมีความซับซ้อนในเรื่องลำดับหรือขั้นตอนการทำงานจึงมีการนำสแตกมาช่วยใช้ในการคำนวณในระบบคอมพิวเตอร์

6.2 การประยุกต์ใช้งานสแตก

- รูปแบบนิพจน์ทางคณิตศาสตร์จะแบ่งออกเป็น 3 ประเภทคือ
  1) นิพจน์ prefix คือ นิพจน์ที่มีเครื่องหมายดำเนินการ (operator) อยู่ด้านหน้า ตัวถูกดำเนินการ (operand) เช่น + A B
  2) นิพจน์ infix คือ นิพจน์ที่มีเครื่องหมายดำเนินการ (operator) อยู่ระหว่าง ตัวถูกดำเนินการ (operand) เช่น A + B
  3) นิพจน์ postfix คือ นิพจน์ที่มีเครื่องหมายดำเนินการ (operator) อยู่ด้านหลัง ตัวถูกดำเนินการ (operand) เช่น A B +

Prefix	+	A	B
Infix	A	+	B
Postfix	A	B	+

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ลำดับความสำคัญของตัวนิพจน์

ในการแปลงนิพจน์ infix เป็น postfix ของ stack นั้นจะคำนึงถึงโอเปอเรเตอร์ (operator) หรือเรียกว่าเครื่องหมาย +, *, /, ^ ในการแปลงข้อมูลจะคำนึงถึงลำดับความสำคัญของเครื่องหมายซึ่งจะมีเครื่องหมายต่าง ๆ ตามที่กล่าวมาข้างต้น รวมถึงเครื่องหมายวงเล็บ ( ) ดังต่อไปนี้

ลำดับความสำคัญ	ตัวดำเนินการ	คำอธิบาย
1	^	ยกกำลัง ความสำคัญลำดับแรก
2	* , /	คูณ, หาร รองจากยกกำลัง
3	+ , -	บวก, ลบ รองจาก ยกกำลัง คูณ หาร
4	( )	เครื่องหมายวงเล็บใช้กำหนดลำดับการคำนวณ

กรณีที่ลำดับความสำคัญเท่ากัน และ ไม่มีวงเล็บ ให้เทียบลำดับความสำคัญจากซ้ายไปขวายกเว้นเลขยกกำลังความสำคัญจากขวาไปซ้าย

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

6.2.1 การแปลงนิพจน์ infix ให้เป็น prefix

การแปลงนิพจน์แบบ prefix เป็นการแปลงนิพจน์ infix คือ

- นิพจน์ที่มีตัวดำเนินการ (operator) อยู่ระหว่างตัวถูกกระทำ (operand)
- เป็นนิพจน์ที่มีตัวดำเนินการอยู่ก่อนตัวถูกกระทำ
- การท่องแบบ prefix จะมีนิพจน์แสดง ดังนี้ + * a + b c d

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ขั้นตอนการแปลงนิพจน์ infix ให้เป็น prefix

1) สร้างสแตกว่างขึ้นมาก่อน

2) พิจารณาข้อมูลจากขวามาซ้ายทีละลำดับข้อมูล

3) ถ้าข้อมูลที่นำเข้ายังไม่หมดให้พิจารณาข้อมูลนำเข้า ดังนี้

   3.1) ถ้าข้อมูลนำเข้าเป็น operand นำเข้าสู่ผลลัพธ์

   3.2) ถ้าข้อมูลนำเข้าเป็น วงเล็บปิด ให้นำเข้าสู่สแตก

   3.3) ถ้าข้อมูลนำเข้าเป็น วงเล็บเปิด ให้นำ operator ออกเป็นผลลัพธ์ จนพบวงเล็บเปิดและลบวงเล็บเปิด และวงเล็บปิดนั้นออกจากสแตก

   3.4) ถ้าข้อมูลนำเข้าเป็น operator ให้พิจารณา ดังนี้

      3.4.1) สแตกว่าง ให้นำเข้าสแตก

      3.4.2) สแตกไม่ว่างและ operator อยู่ ให้พิจารณาดังต่อไปนี้

         - ถ้าตัว opertor ที่จะนำเข้า ลำดับความสำคัญมากกว่า operator ในสแตกให้นำเข้าสแตก

         - ถ้าตัว opertor ที่จะนำเข้า ลำดับความสำคัญน้อยกว่าหรือเท่ากับ operator ในสแตก ให้นำข้อมูลใน แสตกออกไปเป็นผลลัพธ์

4) กลับผลลัพธ์ข้อมูลก่อนแสดงผล

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ตัวอย่างที่ 6.7 จงแปลงนิพจน์ infix ต่อไปนี้ (A+B)/(C*D) ให้เป็น prefix

พิจารณาข้อมูลจากขวามาซ้ายทีละลำดับข้อมูล)D*C(/)B+A(

นิพจน์ infix	stack	นิพจน์ prefix
)	)	
D	)	D
*	)*	D
C	)*	DC
(	)(	DC*
/	/	DC*
)	/)	DC*
B	/)	DC*B
+	/)+	DC*B
A	/)+	DC*BA
(	)(	DC*BA+/

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ตัวอย่างที่ 6.8 จงแปลงนิพจน์ infix ต่อไปนี้ A-B/C-D*E ให้เป็น prefix

พิจารณาข้อมูลจากขวามาซ้ายทีละลำดับข้อมูล E*D-C/B-A

นิพจน์ infix	stack	นิพจน์ prefix
E	-	E
*	*	E
D	*	ED
-	-	ED*
C	-	ED*C
/	-/	ED*C
B	-/	ED*CB
-	-	ED*CB/
A		ED*CB/A- -

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

6.2.2 การแปลงนิพจน์ infix ให้เป็น postfix

การแปลงนิพจน์แบบ postfix การแปลงนิพจน์infix คือ

- นิพจน์ที่มีตัวดำเนินการ (operator) อยู่ระหว่างตัวถูกกระทำ (operand)
- เป็นนิพจน์ที่มีตัวดำเนินการอยู่**<u>หลัง</u>**ตัวถูกกระทำ
- การท่องแบบ postfix จะมีนิพจน์แสดง ดังนี้ abc+*d+

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ขั้นตอนการแปลง

1) สร้างสแตกว่าง

2) พิจารณาข้อมูลนำเข้า ดังนี้

2.1) ถ้าเป็น operand (ค่าคงที่หรือตัวแปร) ให้นำออกเป็นผลลัพธ์

2.2) ถ้าเป็น operator ให้ทำการเปรียบเทียบลำดับความสำคัญ ดังนี้

2.2.1) ถ้าสแตกว่างไม่มี operator ให้นำข้อมูลลงสแตก

2.2.2) ถ้าสแตกไม่ว่างและมี operator อยู่ ให้พิจารณาดังต่อไปนี้

- ถ้าตัว opertor ที่จะนำเข้า ลำดับความสำคัญมากกว่า operator ในสแตก ให้นำเข้าสแตก
- ถ้าตัว opertor ที่จะนำเข้า ลำดับความสำคัญน้อยกว่า operator ในสแตก ให้นำข้อมูลในแสตกออกไปเป็นผลลัพธ์

2.3) ถ้าเป็นวงเล็บเปิดให้นำเข้าสแตก

2.4) ถ้าเป็นวงเล็บปิด ให้พิจารณาข้อมูลในสแตก ดังนี้

2.4.1) ถ้าพบ operator ให้นำออกเป็นผลลัพธ์

2.4.2) ถ้าพบวงเล็บเปิด ให้ลบวงเล็บเปิดในสแตกและวงเล็บปิดที่พิจารณาออกจากสแตก

2.4.3) ถ้าไม่พบวงเล็บเปิด หรือสแตกว่างก็ให้แจ้งความผิดพลาดที่พบ

3) พิจารณาข้อมูลนำเข้าจนกระทั่งหมด และถูกนำไปเป็นผลลัพธ์ทั้งหมดจนสแตกว่าง

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ตัวอย่างที่ 6.9 จงแปลงนิพจน์ infix ต่อไปนี้ (A+B)/(C*D) ให้เป็น postfix

นิพจน์ infix	stack	นิพจน์ postfix
(	(	
A	(	A
+	(+	A
B	(+	AB
)	()	AB+
/	/	AB+
(	/(	AB+
C	/(	AB+C
*	/(*	AB+C
D	/(*	AB+CD
)	()	AB+CD*/

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ตัวอย่างที่ 6.10 จงแปลงนิพจน์ infix ต่อไปนี้ A*(B+C)/D ให้เป็น postfix

นิพจน์ infix	stack	นิพจน์ postfix
A		A
*	*	A
(	(	A*
B	(	A*
+	(+	A*B
C	(+	A*BC
)		A*BC
/	/	A*BC
D	/	A*BCD/

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

จากตัวอย่างข้างต้นสามารถนำมาประยุกต์เป็นโปรแกรมการแปลงค่านิพจน์ infix เป็น postfix โดยใช้การทำงานของสแตก ได้ดังตัวอย่างที่ 6.14

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ตัวอย่างที่ 6.14 โปรแกรมในการแปลงค่านิพจน์ infix เป็น postfix โดยใช้การทำงานของสแตก

```
1   #include<iostream>
2   #include<cstring>
3   #include<conio.h>
4   #include<stack>
5   using namespace std;
6   int getWeightOP(char ch)
7   {
8       switch (ch)
9       {
10          case '/':
11          case '*': return 2;
12          case '+':
13          case '-': return 1;
14          default : return 0;
15      }
16  }
```

```
17  void infix2postfix(char infix[],
    char postfix[], int size)
18  {
19    stack<char> s;
20    int weight;
21    int InExp = 0;
22    int k = 0;
23    char ch;
24    while (InExp < size)
25    {
26      ch = infix[InExp];
27      if (ch == '(')
28      {
29          s.push(ch);
30          InExp++;
31          continue;
32      }
```

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ตัวอย่างที่ 6.14 โปรแกรมในการแปลงค่านิพจน์ infix เป็น postfix โดยใช้การทำงานของสแตก

```
33    if (ch == ')')
34    {
35       while (!s.empty() && s.top() != '(')
36       {
37       postfix[k++] = s.top();
38       s.pop();
39       }
40       if (!s.empty()) {      s.pop();      }
41       InExp++;
42       continue;
43    }
44    weight = getWeightOP(ch);
45    if (weight == 0)
46    {
47    postfix[k++] = ch;
48    } else {
49    if (s.empty())
50    {
51                s.push(ch);
52    } else {
```

```
53    while (!s.empty() && s.top() != '(' &&
      weight <=getWeightOP(s.top()))
54    {
55    postfix[k++] = s.top();
56    s.pop();
57    }
58    s.push(ch);
59    }
60    }
61    InExp++;
62    }
63    while (!s.empty())
64    {
65       postfix[k++] = s.top();
66       s.pop();
67    }
68    postfix[k] = 0;
69  }
```

6.2 การประยุกต์ใช้งานสแตก(ต่อ)

ตัวอย่างที่ 6.14 โปรแกรมในการแปลงค่านิพจน์ infix เป็น postfix โดยใช้การทำงานของสแตก

```
70  int main()
71  {
72          char infix[] = "1*(2+3)/4";
73          int size = strlen(infix);
74          char postfix[size];
75          infix2postfix(infix,postfix,size);
76          cout<<"\nInfix Expression :: "<<infix;
77          cout<<"\nPostfix Expression :: "<<postfix;
78          return 0;
79  }
```

6.3 คิวแบบเส้นตรง

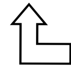
- โครงสร้างข้อมูลอีกแบบที่คล้ายกับสแตก (Stack) นั่นคือ คิว (Queue)
- ข้อมูลในคิวมีการทำงานแบบ First-In-First-Out (FIFO)
- องค์ประกอบแบบคิวสามารถเพิ่มข้อมูลเข้าคิวหรือลบออกจากคิวได้ตลอดเวลา
- เรานำข้อมูลเข้าในคิวทางท้ายของลำดับ (enqueue)และลบข้อมูลออกทางด้านหน้า
- ตัวอย่าง การต่อคิวซื้อข้าวในโรงอาหาร ถ้าในแถวไม่มีใครเข้าคิวซื้ออาหาร ผู้ซื้อคนแรกด้านหน้าของแถวจะเข้าคิวซื้อมาจากทางท้ายของแถวเพื่อรอรับบริการ และจะออกจากคิวไปจากทางด้านหน้า

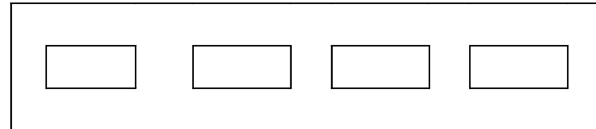
6.3 คิวแบบเส้นตรง(ต่อ)

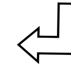
การทำงานของคิว

ข้อมูลออก

(Dequeue)







ข้อมูลเข้า

(Enqueue)

ด้านหน้า(front)

ด้านหลัง(rear)

6.3 คิวแบบเส้นตรง(ต่อ)

การประกาศตัวแปรเริ่มต้น ดังรายละเอียดต่อไปนี้

```
#define SIZE 5
int data[SIZE];
front=0;
rear=0;
```

6.3 คิวแบบเส้นตรง(ต่อ)

- การสร้างคิวเส้นตรงเป็นคิวแบบอาร์เรย์ต้องกำหนดขนาดของข้อมูลไว้ก่อน
- กำหนดตัวแปรให้กับคิวสองตัว คือ
  - front ทำหน้าที่อ้างอิงตำแหน่งข้อมูลส่วนหน้าของคิว
  - rear ทำหน้าที่อ้างอิงตำแหน่งข้อมูลส่วนท้ายของคิว

6.3 คิวแบบเส้นตรง(ต่อ)

6.3.1 การเพิ่มข้อมูลในคิวเส้นตรง

วิธีการเพิ่มข้อมูลในคิวเส้นตรง

- ทำการตรวจสอบก่อนว่ามีพื้นที่ให้เก็บข้อมูลหรือไม่
    - ถ้าพื้นที่เต็มควรแจ้งข้อความบอก
    - ถ้ามีพื้นที่ให้เพิ่มข้อมูลจึงเพิ่มข้อมูลเข้าสู่คิวข้อมูล

- การเพิ่มแต่ละครั้งช่องข้อมูลจะลดลงหนึ่งช่อง
- คิวจะสามารถเก็บข้อมูลได้เท่ากับตัวแปร SIZE
- ถ้าคิวยังไม่เต็มตัวแปร rear จะเพิ่มค่าไปเรื่อย ๆ

6.3 คิวแบบเส้นตรง(ต่อ)

ตัวอย่างที่ 6.15 การเพิ่มข้อมูลในคิวแบบเส้นตรง

โดยกำหนดให้ ตัวแปร rear ข้อมูลปลายคิว และตัวแปร front ข้อมูลต้นคิว

```
1   void enqueue()
2   {
3           int no;
4           if (rear==SIZE && front==0)
5           {
6                   cout<<"queue is full";
7                   }else{
8                   cout<<"Enter Number to enqueue :";
9                   cin>>no;
10                  data[rear]=no;
11          }
12          rear++;
13  }
```

6.3.2 การลบข้อมูลในคิวเส้นตรง

- ทำการตรวจสอบก่อนว่าคิวว่างหรือไม่ (front==rear) โดย
  - ถ้ามีพื้นที่ว่างให้แจ้งข้อความบอกว่า "Queue is empty"
  - ถ้าคิวมีข้อมูลอยู่จากเดิมการนำข้อมูลเข้าจะนำข้อมูลเข้าทางด้าน rear ดังนั้นการนำข้อมูลออกจะทำด้านตรงข้ามกับด้านนำข้อมูลเข้าโดยการนำข้อมูลออกจะสามารถทำในตำแหน่ง front

6.3 คิวแบบเส้นตรง(ต่อ)

ตัวอย่างที่ 6.16 การลบข้อมูลในคิวแบบเส้นตรง

```
1   void dequeue()
2   {
3       int no,i;
4       if (front==rear)
5       {
6           cout<<"Queue is empty";
7           }else{
8           no=data[front];
9           front++;
10          cout<<"\n"<<no<<" -removed from the queue\n";
11      }
12  }
```

6.3.3 การแสดงข้อมูลในคิวเส้นตรง

การนำข้อมูลในคิวเส้นตรงเพื่อมาแสดงข้อมูล สามารถทำโดยตรวจสอบข้อมูลภายในคิว โดย

ถ้าตำแหน่งที่ชี้ข้อมูล front ที่อ้างอิงตำแหน่งข้อมูลส่วนหน้าของคิว และ rear ที่อ้างอิงตำแหน่งข้อมูลส่วนท้ายของคิวเท่ากัน จะมีตัวเลือกการทำงานสองแบบ คือ

1) ถ้าเป็นจริงให้แสดงว่าไม่มีข้อมูลให้แจ้งข้อความ "Queue is empty"

2) ถ้าเป็นเท็จให้แสดงข้อมูลที่มีในคิวทั้งหมดออกมา

6.3 คิวแบบเส้นตรง(ต่อ)

ตัวอย่างที่ 6.17 การนำข้อมูลในคิวเส้นตรงเพื่อมาแสดงข้อมูล

```
1	void display()
2	{
3			int i,temp=front;
4			if (front==rear)
5			{
6					cout<<"Queue is empty";
7					} else {
8					cout<<"\n element in the queue:";
9					for(i=temp;i<rear;i++)
10					{
11							cout<<data[i]<<" ";
12					}
13			}
14	}
```

6.3 คิวแบบเส้นตรง(ต่อ)

ตัวอย่างที่ 6.18 การประยุกต์ใช้งานคิวแบบเส้นตรง

โดยกำหนดให้ค่าเริ่มต้น Front = 0 Rear = 0 size=5

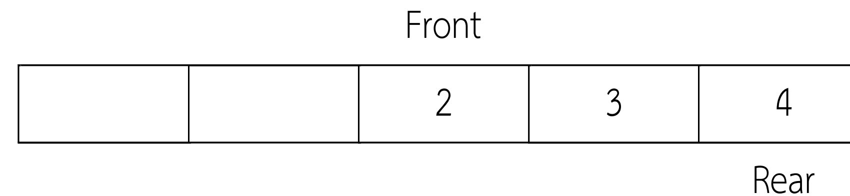
if (rear==SIZE && front==0) ถ้าจริง แสดงว่าไม่มีพื้นที่ให้เก็บข้อมูล และเริ่มต้นนำ 1 เข้าเก็บในคิว

6.3 คิวแบบเส้นตรง(ต่อ)

	Front	Rear				
นำ 1 เข้าคิว		1				
	Front		Rear			
นำ 3 เข้าคิว		1	3			
	Front			Rear		
นำ 5 เข้าคิว		1	3	5		
	Front				Rear	
นำ 7 เข้าคิว		1	3	5	7	
		Front			Rear	
ลบคิว			3	5	7	
			Front		Rear	
ลบคิว				5	7	
				Front	Rear	
ลบคิว					7	
				Front		Rear
นำ 9 เข้าคิว					7	9
					Front	Rear
ลบคิว						9
						Front, Rear
ลบคิว						

6.4 คิวแบบวงกลม

คิวแบบวงกลม จากตัวอย่างข้างต้นคิวแบบเส้นตรง การเพิ่มข้อมูล (Rear) และการลบข้อมูล (Front) ถ้าพิจารณาโครงสร้างคิวแบบเส้นตรง 5 ช่อง ตามภาพที่คิวแบบเส้นตรง ที่แสดงดังนี้



เมื่อพิจารณา front index ชี้ตำแหน่งที่ 2 และ rear ชี้ตำแหน่งที่ 4 ต่อมานำข้อมูลออกจากคิวเส้นตรงอีก front จะเลื่อนลำดับมาที่ 3,4 ตามลำดับ จะพบว่าการเพิ่มข้อมูลในคิวแบบเส้นตรงจะเกิดปัญหาในการเพิ่มข้อมูล ด้วยปัญหานี้จึงมีผู้พัฒนาคิวแบบวงกลมขึ้นมาเพื่อแก้ปัญหาการเพิ่มข้อมูลโดย เมื่อทำการเพิ่มข้อมูลอีกครั้ง Rear จะชี้ตำแหน่ง index = 0 จึงทำให้แก้ปัญหาการเพิ่มข้อมูลนี้ได้

6.4 คิวแบบวงกลม(ต่อ)

ตัวอย่างที่ 6.19 จงประกาศโครงสร้างคิวแบบวงกลม

```
1	class circularqueue
2	{
3			int data[5];
4			int front,rear;
5			int size;
6			public:
7			circularqueue()
8			{
9				front=0;
10				rear=0;
11				size=5;
12			}
13			void display();
14			void enqueue();
15			void dequeue();
16	};
```

6.4 คิวแบบวงกลม(ต่อ)

6.4.1 การเพิ่มข้อมูลในคิวแบบวงกลม

- การดำเนินการเพิ่มข้อมูลของคิวแบบวงกลมสามารถทำได้เหมือนกับคิวแบบเสนตรงคือ
    - มีการนำขอมูลเขาคิวและออกจากคิวเหมือนกัน
    - แต่การนำขอมูลเขาและออกจากคิวนี้จะมีกระบวนการที่แตกตางกันออกไปจากคิวแบบเส้นตรง คือ
        - การนำขอมูลเขาคิวจะตองการตรวจสอบก่อนทำการเพิ่มข้อมูล

6.4 คิวแบบวงกลม(ต่อ)

ตัวอย่างที่ 6.20 การเพิ่มข้อมูลในคิวแบบวงกลม

```
1   void circularqueue :: enqueue()
2   {
3   cout<<endl;
4   if(front==0 && rear==0)
5   {
6   cout<<"Enter Number to enqueue data["
    <<rear+1<<"] :";
7   cin>>data[1];
8   rear=1;
9   front=1;
10  }else{
11  int next=(rear % size)+1;
12          if(next==front)
13          {
14                  cout<<"Queue is Full";
15                  getch();
16                  } else {
17                  cout<<"Enter Number to enqueue
                    data["<<next<<"] :";
18                  cin>>data[next];
19                  rear=next;
20          }
21      }
22  }
```

6.4.2 การลบข้อมูลในคิวแบบวงกลม

การลบข้อมูลในคิวแบบวงกลมสามารถทำได้คล้ายกับคิวแบบเส้นตรงเช่นกัน ดังแสดงในตัวอย่าง 6.21 ตัวอย่างโปรแกรมด้านล่างนี้

6.4 คิวแบบวงกลม(ต่อ)

ตัวอย่างที่ 6.21 การลบข้อมูลในคิวแบบวงกลม

```
1   void  circularqueue :: dequeue()
2   {
3               cout<<endl;
4               if(rear==0 && front==0)
5               {
6                           cout<<"Queue is empty";
7                           getch();
8                           return;
9               }
10              if(rear==front)
11              {
12                          rear=0;
13                          front=0;
14                          } else {
15                          front=(front % size)+1;
16              }
17  }
```

6.4 คิวแบบวงกลม(ต่อ)

6.4.3 การแสดงข้อมูลในคิวแบบวงกลม

- การนำข้อมูลในคิวแบบวงกลมสามารถทำได้คล้ายกับคิวแบบเส้นตรงแต่มีข้อแตกต่างบ้าง ดังนี้
- การตรวจสอบข้อมูลภายในคิวทำได้โดย
  - ถ้าตำแหน่งที่ชี้ข้อมูล front ที่อ้างอิงตำแหน่งข้อมูลส่วนหน้าของคิว และ rear ที่อ้างอิงตำแหน่งข้อมูลส่วนท้ายของคิวไม่เท่ากับ 0 เป็นจริง ให้แสดงข้อมูลที่มีในคิวทั้งหมดออกมา
  - แต่ถ้าเป็นเท็จให้แสดงข้อความ Queue is empty

ตัวอย่างที่ 6.22 การแสดงข้อมูลในคิวแบบวงกลม

```
1   void circularqueue :: display()
2   {
3      cout<<endl;
4      if(front!=0 && rear!=0)
5      {
6         int no=front;
7         cout<<"data["<<no<<"] :"<<data[no]<<endl;
8         while(no!=rear)
9         {
10           no=(no % size)+1;
11           cout<<"data["<<no<<"] :"<<data[no]<<endl;
12        }
13        } else {
14           cout<<"Queue is empty"<<endl;
15     }
16  }
```

มหาวิทยาลัยราชภัฏนครปฐม