

โครงสร้างข้อมูล

26 ต.ค. 2536



รุ่งระพี กรานค้าย
ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์
มหาวิทยาลัยศรีนครินทรวิโรฒ ประสานมิตร

คานา

วิชา โครงสร้างข้อมูล เป็นวิชาที่มีความสำคัญสำหรับการเรียนวิชาทางด้านคอมพิวเตอร์ เนื่องจากเป็นวิชาที่สอนให้พิจารณาถึง หลักการการจัดเก็บข้อมูลที่ต้องการใช้ในการทำงานของโปรแกรมในหน่วยความจำของเครื่องคอมพิวเตอร์ และวิธีการเข้าถึงข้อมูลดังกล่าว โดยคำนึงถึงประสิทธิภาพในการทำงาน ทั้งในแง่ของความเร็วและการใช้เนื้อที่ของหน่วยความจำ ซึ่งเป็นแนวคิดสำคัญที่ผู้พัฒนา โปรแกรมควรต้องเรียนรู้ และรู้จักพิจารณาเลือกใช้โครงสร้างข้อมูลที่เหมาะสมสำหรับแต่ละงาน เพื่อให้ได้โปรแกรมที่มีประสิทธิภาพ การศึกษาวิชาโครงสร้างข้อมูลยังช่วยให้ผู้เรียน สามารถศึกษาทำความเข้าใจเทคนิคของโปรแกรมต่างๆ ได้ง่ายขึ้น นอกจากนี้ยังเป็นพื้นฐานสำหรับการศึกษาวิชาในระดับสูงขึ้นไป เช่นวิชาการประมวลผลเชิงข้อมูล ระบบการจัดการฐานข้อมูล ระบบปฏิบัติการ เป็นต้น

หนังสือโครงสร้างข้อมูลเล่มนี้ เขียนขึ้นเพื่อใช้ในการเรียนการสอนวิชาโครงสร้างข้อมูลในระดับปริญญาตรี ตลอดจนน่าสนใจทั่วไป โดยพยายามให้ผู้อ่านสามารถทำความเข้าใจได้ง่าย ใช้ภาษาปากกาลในการแสดงขั้นตอนวิธี และได้สร้างโปรแกรมตัวอย่างสำหรับแสดงการดำเนินงานในโครงสร้างแบบรายการโยงและโครงสร้างแบบต้นไม้ เพื่อให้ผู้อ่านสามารถทดลองปฏิบัติเพื่อให้เกิดความเข้าใจได้ดียิ่งขึ้น คำศัพท์ที่ใช้ในหนังสือเล่มนี้ส่วนมากจะใช้ตามศัพท์บัญญัติคอมพิวเตอร์ฉบับราชบัณฑิตยสถาน ยกเว้นบางคำที่เป็นที่รู้จักและยอมรับกันทั่วไป เนื้อหาทั้งหมดมี 8 บท ได้แก่

- บทที่ 1 บทนำ
- บทที่ 2 การเรียงลำดับข้อมูลและการค้นข้อมูล
- บทที่ 3 โครงสร้างข้อมูลแบบแถวลำดับ
- บทที่ 4 โครงสร้างข้อมูลแบบคิว
- บทที่ 5 โครงสร้างข้อมูลแบบสแตก
- บทที่ 6 โครงสร้างข้อมูลแบบรายการโยง
- บทที่ 7 โครงสร้างข้อมูลแบบต้นไม้
- บทที่ 8 โครงสร้างข้อมูลแบบกราฟ

ผู้เขียนหวังว่าหนังสือเล่มนี้คงเป็นประโยชน์แก่นักศึกษาและผู้สนใจตามสมควร และขอแสดงความขอบคุณ พันตำรวจโทวีระชัย กรานคำยี่ ซึ่งได้ให้ความสนับสนุนในการจัดทำหนังสือเป็นอย่างดี และขอขอบคุณลูกศิษย์ที่เรียนวิชาโครงสร้างข้อมูลที่ได้ช่วยในการทดสอบโปรแกรมตัวอย่างที่ปรากฏในหนังสือเล่มนี้

การเขียนหนังสือเล่มนี้ ผู้เขียนใช้เวลาและการค้นคว้าอ้างอิงจากหนังสือหลายเล่ม ความดีของหนังสือเล่มนี้จึงขอมอบแด่ ครูบาอาจารย์ผู้ให้ความรู้ และผู้เขียนตำราเผยแพร่ความรู้ อันเป็นประโยชน์ ให้ผู้เขียนและบุคคลทั่วไปได้ศึกษาค้นคว้า

รุ่งระพี กรานคำยี่

ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์

มหาวิทยาลัยศรีนครินทรวิโรฒ ประสานมิตร

กันยายน 2535

สารบัญ

	หน้า
คํานํา	ก
บทที่ 1 บทนํ้า	1
1.1 ความหมายและความสำคัญของ โครงสร้างข้อมูล	1
1.2 โครงสร้างหน่วยความจำ	2
1.2.1 ประเภทของหน่วยความจำ	2
1.2.2 การอ้างอิงถึงตำแหน่งของข้อมูลในหน่วยความจำ	3
1.3 การแทนข้อมูลประเภทต่าง ๆ ในหน่วยความจำ	3
1.3.1 ข้อมูลแบบจำนวน	4
1.3.2 ข้อมูลแบบอักขระ	9
1.3.3 ข้อมูลแบบบูล	9
1.3.4 ข้อมูลแบบตัวชี้	9
1.4 ประเภทของโครงสร้างข้อมูล	9
1.4.1 โครงสร้างข้อมูลแบบเชิงเส้น	10
1.4.2 โครงสร้างข้อมูลแบบไม่เป็นเชิงเส้น	10
1.5 การจัดสรรเนื้อที่สำหรับข้อมูลในหน่วยความจำ	10
1.5.1 การจัดสรรเนื้อที่แบบสถติก	10
1.5.2 การจัดสรรเนื้อที่แบบไดนามิก	10
แบบฝึกหัด	12

บทที่ 2	การเรียงลำดับข้อมูลและการค้นข้อมูล	13
2.1	การเรียงลำดับข้อมูล	13
2.2	ประเภทการเรียงลำดับข้อมูล	14
2.3	หลักการเรียงลำดับข้อมูล	14
2.3.1	หลักการสลับค่า	15
2.3.2	หลักการคัดเลือกค่า	15
2.3.3	หลักการแทรกค่า	15
2.4	การพิจารณาประสิทธิภาพของการเรียงลำดับข้อมูล	15
2.5	ฟังก์ชันบิก โอ	16
2.6	การเรียงลำดับแบบฟอง	17
2.7	การเรียงลำดับแบบคัดเลือก	19
2.8	การเรียงลำดับแบบแทรก	21
2.9	การเรียงลำดับแบบควิก	23
2.10	การค้นข้อมูล	27
2.11	ตัวอย่าง โปรแกรม	32
	แบบฝึกหัด	43
บทที่ 3	โครงสร้างข้อมูลแบบแถวลำดับ	44
3.1	ความหมายและการเรียกใช้	44
3.1.1	การดำเนินงานกับตัวแปรแถวลำดับ	45
3.1.2	มิติ	45
3.2	ลักษณะของตัวแปรแถวลำดับ	47
3.2.1	ตัวแปรแถวลำดับแบบหนึ่งมิติ	47
3.2.2	ตัวแปรแถวลำดับแบบสองมิติ	48
3.2.3	ตัวแปรแถวลำดับแบบสามมิติ	48

3.3	การเก็บตัวแปรแถวลำดับในหน่วยความจำ	
	และการคำนวณฟังก์ชันเพื่อเข้าถึง_____	49
3.3.1	การเก็บตัวแปรแถวลำดับแบบหนึ่งมิติ_____	49
3.3.2	ฟังก์ชันเพื่อเข้าถึงตัวแปรแถวลำดับแบบหนึ่งมิติ_____	50
3.3.3	การเก็บตัวแปรแถวลำดับแบบสองมิติ_____	51
3.3.4	ฟังก์ชันเพื่อเข้าถึงตัวแปรแถวลำดับสองมิติ_____	53
3.3.5	การเก็บตัวแปรแถวลำดับแบบสามมิติ_____	56
3.4	การประยุกต์ของแถวลำดับ_____	57
	แบบฝึกหัด_____	60
บทที่ 4	โครงสร้างข้อมูลแบบคิว_____	62
4.1	ลักษณะของคิว_____	62
4.2	การดำเนินงานในคิว_____	63
4.3	การกำหนดคิวและขั้นตอนวิธีดำเนินการ_____	64
4.4	คิววงกลม_____	68
4.5	การประยุกต์ของคิว_____	71
	แบบฝึกหัด_____	73
บทที่ 5	โครงสร้างข้อมูลแบบสแตก_____	75
5.1	ลักษณะของสแตก_____	75
5.2	การดำเนินงานในสแตก_____	76
5.2.1	การใส่ข้อมูลเข้าในสแตก_____	76
5.2.2	การนำข้อมูลออกจากสแตก_____	77

5.3	การกำหนดสแตทและขั้นตอนวิธีดำเนินงาน	77
5.4	การประยุกต์ใช้สแตท	79
5.4.1	การเรียกซ้ำ	80
5.4.2	สัญญาณเต็มกลางและสัญญาณเต็มหลัง	84
	แบบฝึกหัด	94
บทที่ 6	โครงสร้างข้อมูลแบบรายการโยง	96
6.1	บทนำ	96
6.1.1	การใช้เนื้อที่แบบสแตติก	96
6.1.2	การใช้เนื้อที่แบบไดนามิก	97
6.2	รายการโยง	98
6.3	การดำเนินการในรายการโยง	99
6.4	ประเภทของรายการโยง	100
6.5	รายการโยงเดี่ยว	100
6.5.1	การสร้างรายการโยง	100
6.5.2	การแนวม่านรายการโยง	101
6.5.3	การใส่ข้อมูลเข้าในรายการโยง	102
6.5.4	การนำข้อมูลออกจากรายการโยง	105
6.6	การแทนรายการโยงด้วยตัวแปรแถวลำดับ	107
6.7	รายการโยงคู่	109
6.8	การดำเนินการในรายการโยงคู่	110
6.8.1	การใส่ข้อมูลเข้าในรายการโยงคู่	110
6.8.2	การนำข้อมูลออกจากรายการโยงคู่	112
6.9	ตัวอย่างโปรแกรม	113
	แบบฝึกหัด	125

บทที่ 7	โครงสร้างข้อมูลแบบต้นไม้	129
7.1	นิยามของต้นไม้	129
7.2	การแทนรูปต้นไม้	131
7.3	ต้นไม้แบบทวิภาค	133
7.4	การกำหนดโครงสร้างแบบต้นไม้	134
7.5	ตัวอย่างการสร้างต้นไม้	136
7.6	การแนวม่านต้นไม้แบบทวิภาค	138
7.7	ต้นไม้แบบค้นหาทวิภาค	144
7.7.1	นิยามของต้นไม้แบบทวิภาค	144
7.7.2	การใส่ข้อมูลเข้าไปในต้นไม้แบบทวิภาค	145
7.7.3	การนำข้อมูลออกจากต้นไม้แบบทวิภาค	146
7.7.4	ขั้นตอนวิธีการค้นหาข้อมูลในต้นไม้แบบทวิภาค	147
7.7.5	ขั้นตอนวิธีการใส่ข้อมูลเข้าไปในต้นไม้แบบทวิภาค	148
7.7.6	ขั้นตอนวิธีการนำข้อมูลออกจากต้นไม้แบบทวิภาค	149
7.8	ต้นไม้ได้ดูล	151
7.8.1	นิยามของต้นไม้ได้ดูล	151
7.8.2	การปรับสมดุลของต้นไม้	152
7.9	ตัวอย่างโปรแกรม	156
	แบบฝึกหัด	166

บทที่ 8	โครงสร้างข้อมูลแบบกราฟ	168
8.1	กราฟและเครือข่าย	168
8.2	การแทนโครงสร้างข้อมูลแบบกราฟ	170
8.2.1	Adjacency Matrix	170
8.2.2	Adjacency List	171
8.2.3	การกำหนดโครงสร้างข้อมูลแบบกราฟ	174

8.3	การฉายภาพ	176
8.3.1	การค้นตามแนวกว้าง	176
8.3.2	ขั้นตอนวิธีการค้นตามแนวกว้าง	177
8.3.3	การค้นในแนวลึก	179
8.3.4	ขั้นตอนวิธีการค้นตามแนวลึก	181
8.5	การประยุกต์ของกราฟ	184
	แบบฝึกหัด	185
	บรรณานุกรม	188
	ภาคผนวก	189
	ดัชนีรูป	209
	ดัชนีตาราง	215

บทที่ 1

บทนำ

- หัวข้อ :
- 1.1 ความหมายและความสำคัญของโครงสร้างข้อมูล
 - 1.2 โครงสร้างหน่วยความจำ
 - 1.3 การแทนข้อมูลประเภทต่าง ๆ ในหน่วยความจำ
 - 1.4 ประเภทของโครงสร้างข้อมูล
 - 1.5 การจัดสรรเนื้อที่ของข้อมูลในหน่วยความจำ

1.1 ความหมายและความสำคัญของโครงสร้างข้อมูล

โครงสร้างข้อมูล (Data Structures) หมายถึง โครงสร้างการจัดเก็บข้อมูลในหน่วยความจำ ซึ่งมีข้อกำหนดเกี่ยวกับระเบียบวิธีการจัดเก็บข้อมูล (store) ตลอดจนการเข้าถึง (access) ข้อมูลเพื่อการใช้งาน

โครงสร้างข้อมูลมีความสำคัญ และเกี่ยวข้องกับการทำงานของโปรแกรมทุก ๆ โปรแกรม พิจารณาจากข้อความ

Algorithms + Data Structures = Programs

จะเห็นว่าในการที่จะเขียนโปรแกรมเพื่อจัดการกับข้อมูล เพื่อให้ได้ผลลัพธ์ตามต้องการนั้น สิ่งที่ต้องนำมาใช้ประกอบด้วย ขั้นตอนวิธี และ โครงสร้างข้อมูล

ขั้นตอนวิธี (Algorithms) หมายถึงขั้นตอนวิธีการแก้ปัญหาที่กำหนดไว้อย่างชัดเจน และสามารถกระทำให้สิ้นสุดได้ภายในช่วงเวลาอันจำกัด

การวัดประสิทธิภาพของขั้นตอนวิธี ในการแก้ปัญหาหนึ่ง ๆ อาจมีวิธีการสำหรับแก้ปัญหา นั้นมากกว่าหนึ่งวิธี ซึ่งเราควรเลือกใช่วิธีการที่ดีที่สุด โดยพิจารณาถึง

- ความเร็วในการทำงานของขั้นตอนวิธี

- และ - การใช้เนื้อที่ของหน่วยความจำ

เนื่องจากโครงสร้างข้อมูลแต่ละแบบเหมาะสมกับงานต่างลักษณะกัน การเลือกใช้โครงสร้างข้อมูลแบบใดจะต้องพิจารณาถึงประสิทธิภาพในการทำงาน ซึ่งพิจารณาจากความเร็วในการเข้าถึงข้อมูล และการใช้เนื้อที่ของหน่วยความจำ

1.2 โครงสร้างหน่วยความจำ

ในที่นี้จะกล่าวถึงเฉพาะหน่วยความจำหลัก หรือหน่วยความจำภายในเครื่องคอมพิวเตอร์ ซึ่งมีหน้าที่ในการเก็บข้อมูลต่าง ๆ ที่กำลังใช้งานในขณะนั้น ขณะที่โปรแกรมกำลังประมวลผล สิ่งที่อยู่ในหน่วยความจำประกอบด้วย โปรแกรมระบบปฏิบัติการ (Operating System) โปรแกรมที่กำลังประมวลผล และข้อมูลที่ใช้งานโดยโปรแกรมนั้น

1.2.1 ประเภทของหน่วยความจำ

ประเภทของหน่วยความจำหลัก แบ่งได้เป็น 2 ประเภท ได้แก่

- (1) แรม
- (2) รอม

แรม (RAM : Random Access Memory) เป็นหน่วยความจำที่ผู้ใช้ใช้งานนั่นเอง โปรแกรมและข้อมูลของผู้ใช้จะอยู่ในส่วนนี้ รวมทั้งโปรแกรมระบบปฏิบัติการด้วย ข้อมูลในแรมจะเก็บได้ไม่ถาวร และจะเก็บค่าได้เฉพาะเมื่อมีกระแสไฟฟ้าอยู่เท่านั้น เนื้อที่ของหน่วยความจำจะมีขนาดจำกัดตามแต่ละระบบคอมพิวเตอร์ที่ใช้ เช่น 640 กิโลไบต์ 4 เมกะไบต์ เป็นต้น

รอม (ROM : Read Only Memory) เป็นชิป (Chip) ที่เก็บค่าที่อยู่ในรูปรหัสฐานสอง ซึ่งมีลักษณะเหมือนสวิตช์ที่ตั้งไว้ถาวร มักใช้สำหรับบันทึกโปรแกรมที่จะติดตั้งไว้ถาวรในเครื่องคอมพิวเตอร์ ซึ่งผลิตมาจากโรงงานผลิตโดยตรง ผู้ใช้ไม่สามารถเขียนหรือบันทึกโปรแกรมหรือข้อมูลลงในรอมได้ ตัวอย่างโปรแกรมที่บันทึกในรอม ได้แก่ โปรแกรมปลูกเครื่อง (Bootstrap Program) ที่ใช้ในตอนที่เริ่มเปิดเครื่อง โปรแกรมนี้จะไปสั่งให้เครื่องเริ่มทำการอ่านข้อมูลจากหน่วยความจำรอง นั่นคืออ่านเอาโปรแกรมระบบปฏิบัติการเข้ามาในหน่วยความจำเพื่อให้เครื่องสามารถปฏิบัติงานได้ หรือในไมโครคอมพิวเตอร์บางเครื่องก็อาจมีโปรแกรมตัวแปลภาษาเบสิก (Basic Interpreter) บันทึกอยู่ในรอม เป็นต้น

การเข้าหาข้อมูลในรอมและแรมจะสามารถทำได้อย่างรวดเร็ว นอกจากนี้ ประเภทของรอมมีหลายประเภท เช่น PROM (Programable ROM), EPROM (Erasable Programable ROM) เป็นต้น

1.2.2 การอ้างอิงถึงตำแหน่งของข้อมูลในหน่วยความจำ

หน่วยย่อยสุดของหน่วยความจำ ได้แก่ บิต (Bit) ซึ่งเก็บค่าที่เป็นเลขฐานสองได้ 1 ค่า คือ 0 หรือ 1 เท่านั้น การจัดแบ่งเขตของหน่วยความจำเพื่อแทนค่าข้อมูลจึงมีการกำหนดเป็น ไบต์ (Byte) ซึ่งแทนได้ 1 ตัวอักษร (Character) หรือกำหนดเป็นเวิร์ด (Word) เป็นต้น การอ้างอิงถึงแต่ละตำแหน่งของหน่วยความจำใช้การกำหนดด้วยหมายเลขประจำตำแหน่ง เรียกว่า เลขที่อยู่ (Address) เช่น เลขที่อยู่ของไบต์กำหนดเป็น 0,1,2,3,... ไปตามลำดับ ซึ่งจะช่วยในการอ้างอิงถึงตำแหน่งที่เก็บข้อมูลในหน่วยความจำ ดังแสดงในรูป 1.1

เลขที่อยู่ แบบไบต์	0	1	2	3	...	(M-1)
ข้อมูลฐานสอง	01011011	00000001				
ค่าในฐานสิบ	99	1				

รูป 1.1 การกำหนดตำแหน่งของข้อมูลในหน่วยความจำ

1.3 การแทนข้อมูลประเภทต่าง ๆ ในหน่วยความจำ

การแทนข้อมูลประเภทต่าง ๆ ในหน่วยความจำจะใช้วิธีการที่ต่างกันไป ตลอดจนเนื้อที่ที่ใช้ก็แตกต่างกันด้วย ในที่นี้จะแสดงการแทนข้อมูล 4 ประเภท ได้แก่ ข้อมูลแบบจำนวน (Integer) ข้อมูลแบบอักขระ (Character) ข้อมูลแบบบูล (Boolean) และข้อมูลแบบตัวชี้ (Pointer)

1.3.1 ข้อมูลแบบจำนวน

ข้อมูลแบบจำนวน (Number) ได้แก่ ข้อมูลจำนวนเลขที่นำมาใช้ในการคำนวณ ข้อมูลประเภทนี้มักถูกแทนในระบบคอมพิวเตอร์ด้วยจำนวนในระบบเลขฐานสอง (Binary System) ซึ่งใช้วิธีการแทนข้อมูลที่แตกต่างกันระหว่างเลขจำนวนเต็ม และเลขจำนวนจริง

1.3.1.1 เลขจำนวนเต็ม

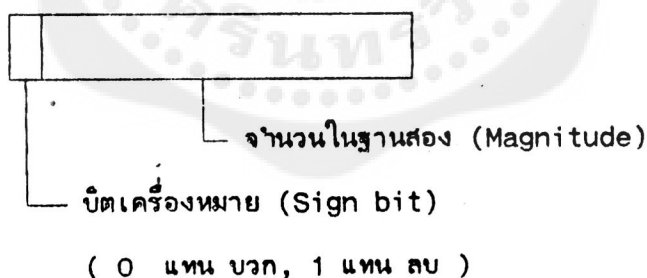
เลขจำนวนเต็ม (Integer) เป็นเซตของจำนวน ในรูป

$\{ \dots, -(n-1), -n, \dots, -2, -1, 0, 1, 2, \dots, n, (n+1), \dots \}$

ช่วงค่าของจำนวนเต็มที่ใช้ในการคำนวณ ถูกจำกัดโดยขนาดเนื้อที่ที่ได้กำหนดไว้สำหรับแทนข้อมูล การแทนเลขจำนวนเต็มในระบบคอมพิวเตอร์ สามารถทำได้หลายวิธี ได้แก่

1. เครื่องหมายและขนาด
2. ส่วนเต็มเต็มฐาน
3. ฐานสิบ

เครื่องหมายและขนาด (Sign and Magnitude) เป็นการแทนข้อมูลด้วยจำนวนฐานสอง โดยกำหนดหนึ่งบิตแรกสำหรับแทนเครื่องหมาย บวกหรือลบ และตามด้วยจำนวนในฐานสอง รูปแบบการเก็บข้อมูลเป็นดังรูป 1.2



รูป 1.2 การแทนจำนวนเต็มแบบเครื่องหมายและขนาด

ตัวอย่างเช่น ถ้าใช้เนื้อที่ 1 ไบต์ จะแทนข้อมูลได้ในช่วง $-(2^7-1)$ ถึง (2^7-1) หรือ -127 ถึง 127 หรือ $1111111(2)$ ถึง $0111111(2)$

+7 แทนด้วย $00000111(2)$

-6 แทนด้วย $10000110(2)$

การแทนจำนวนเต็มด้วยวิธีนี้ ถึงแม้จะดูง่าย แต่ทำให้ต้องตรวจสอบบิตเครื่องหมายก่อนการคำนวณทุกครั้ง นอกจากนี้วงจรการคำนวณของเครื่องคอมพิวเตอร์ต้องมีทั้ง วงจรบวก (Adder) และ วงจรลบ (Subtractor)

ส่วนเติมเต็มฐาน (Radix Complement) วิธีนี้เป็นการแทนข้อมูลจำนวนเต็มด้วยค่าฐานสอง โดยจะแปลงค่าที่เป็นจำนวนเต็มลบให้อยู่ในรูปส่วนเติมเต็มของค่านั้น ซึ่งมีข้อดีคือสามารถทำการคำนวณได้ทันทีโดยไม่ต้องมีการตรวจสอบเครื่องหมาย และวงจรคำนวณใช้เพียงวงจรบวกอย่างเดียว ไม่ต้องมีวงจรลบ ส่วนเติมเต็มฐานมี 2 แบบ ได้แก่

- ส่วนเติมเต็มหนึ่ง (1's Complement)

และ - ส่วนเติมเต็มสอง (2's Complement)

ตัวอย่างการแทนจำนวนแบบส่วนเติมเต็มสอง เมื่อใช้เนื้อที่ 4 บิต บิตแรกจะใช้แทนเครื่องหมาย ค่าที่เป็นจำนวนเต็มบวกจะแทนด้วยค่าฐานสองตามปกติ ค่าที่เป็นลบจะแทนโดยการแปลงแต่ละบิตของจำนวนบวกให้เป็นตรงข้าม (0 เป็น 1 และ 1 เป็น 0) แล้วบวกด้วย 1 เช่น

+5 แทนด้วย 0101
-5 แทนด้วย 1010 + 1 = 1011

จำนวนเต็ม	ค่าที่เก็บ	จำนวนเต็ม	ค่าที่เก็บ
0	0000	-1	1111
1	0001	-2	1110
2	0010	-3	1101
3	0011	-4	1100
4	0100	-5	1011
5	0101	-6	1010
6	0110	-7	1001
7	0111	-8	1000

ตาราง 1-1 การแทนจำนวนเต็มแบบส่วนเติมเต็มสอง เมื่อใช้เนื้อที่ 4 บิต

เมื่อใช้ 4 บิต ขอบเขตของข้อมูลที่แทนได้คือ -8 ถึง $+7$ หรือ -2^3 ถึง 2^3-1

เมื่อใช้เนื้อที่ n บิต ขอบเขตของข้อมูลที่แทนได้ ในการแทนจำนวนเต็มแบบส่วนเติมเต็มสอง คือ -2^{n-1} ถึง $2^{n-1}-1$ ตัวอย่างเช่น ถ้าใช้เนื้อที่ 16 บิต จะแทนค่าได้ระหว่าง -2^{15} ถึง $2^{15}-1$ หรือ $-32,768$ ถึง $32,767$ และถ้าใช้เนื้อที่ 32 บิต จะแทนค่าได้ระหว่าง -2^{31} ถึง $2^{31}-1$ หรือ $-2,147,438,648$ ถึง $2,147,438,647$

การแทนจำนวนเต็มด้วยวิธีส่วนเติมเต็มหนึ่ง จะคล้ายคลึงกับวิธีส่วนเติมเต็มสอง โดยการแทนจำนวนลบ ใช้วิธีแปลงแต่ละบิตของจำนวนบวกให้เป็นตรงข้ามเท่านั้น (ไม่ต้องบวกด้วยหนึ่ง) เช่น -3 แทนด้วย การแปลงบิตของจำนวน $+3$ คือ 0110 เป็นตรงข้ามคือ 1001 แต่วิธีนี้มีข้อเสียคือ ค่าศูนย์จะแทนได้สองแบบแตกต่างกัน เป็น $+0$ (0000) และ -0 (1111)

จำนวนเต็ม	ค่าที่เก็บ	จำนวนเต็ม	ค่าที่เก็บ
0	0000	-1	1110
1	0001	-2	1101
2	0010	-3	1100
3	0011	-4	1011
4	0100	-5	1010
5	0101	-6	1001
6	0110	-7	1000
7	0111	-0	1111

ตาราง 1-2 การแทนจำนวนเต็มแบบส่วนเติมเต็มหนึ่ง เมื่อใช้เนื้อที่ 4 บิต

เมื่อใช้ 4 บิต ขอบเขตของข้อมูลที่แทนได้คือ -7 ถึง $+7$ หรือ $-(2^3-1)$ ถึง (2^3-1)

เมื่อใช้เนื้อที่ n บิต ขอบเขตของข้อมูลที่แทนได้ในการแทนจำนวนเต็มแบบส่วนเติมเต็มหนึ่ง คือ $-(2^{n-1}-1)$ ถึง $(2^{n-1}-1)$ ตัวอย่างเช่น ถ้าใช้เนื้อที่ 16 บิต จะแทนค่าได้ระหว่าง $-(2^{15}-1)$ ถึง $(2^{15}-1)$ หรือ $-32,767$ ถึง $32,767$ และถ้าใช้เนื้อที่ 32 บิต จะแทนค่า

ได้ระหว่าง $-(2^{31}-1)$ ถึง $(2^{31}-1)$ หรือ $-2,147,438,647$ ถึง $2,147,438,647$

ฐานสิบ (Decimal) นอกจากการแทนจำนวนด้วยเลขฐานสองแล้ว อาจแทนข้อมูลในฐานสิบ โดยใช้แต่ละ ไบต์แทนจำนวนแต่ละหลักในฐานสิบ เช่น ข้อมูล 375 จะใช้เนื้อที่ 3 ไบต์ และแทนข้อมูลดังรูป 1.3

3	7	5
00000110	00000111	00000101

รูป 1.3 การแทนจำนวนเต็มแบบฐานสิบ

หรืออาจประหยัดเนื้อที่โดยการใช้เพียง 4 บิต แทนเลขฐานสิบแต่ละตัว ดังนั้นแต่ละ ไบต์สามารถแทนเลขฐานสิบได้ 2 ตัว และใช้ครึ่งไบต์หลังสำหรับแทนเครื่องหมาย ดังรูป 1.4 การแทนข้อมูลฐานสิบแบบดังกล่าว เรียกว่า **ทศนิยมอัดแน่น (Packed Decimal)**

Digit	Digit	...	Digit	Sign

<-- ไบต์ -->

<-- ไบต์ -->

รูป 1.4 การแทนจำนวนเต็มแบบทศนิยมอัดแน่น

1.3.1.2 เลขจำนวนจริง

เลขจำนวนจริง (Real) แทนในระบบคอมพิวเตอร์ได้ 2 แบบ คือ แบบจุดตายตัว (Fixed point) และแบบจุดลอยตัว (Floating point หรือ Scientific notation)

การแทนเลขจำนวนจริงแบบจุดตายตัว เป็นแบบที่ใช้ในการแสดงเลขจำนวนจริงโดยทั่วไป นั่นคือมีเครื่องหมายบวกหรือลบ ตามด้วยตัวเลขหน้าจุดทศนิยม และตัวเลขหลังจุดทศนิยม ในรูปแบบดังนี้

$$A_R = \pm (a_{n-1}a_{n-2}\dots a_1a_0 . a_{-1}\dots a_{-(M-1)}a_{-M})_R$$

เมื่อ A เป็นจำนวนจริงในระบบฐาน R ซึ่งจะมีค่าในฐานสิบคือ

$$A = \pm \sum_{i=-M}^{N-1} a_i R^i$$

ซึ่งช่วงค่าที่แทนได้จะไม่มากนัก ในกรณีที่ต้องการแทนจำนวนที่มีค่าสูงมาก ๆ หรือค่าต่ำมาก ๆ การแทนข้อมูลแบบจุดตายตัวจะไม่เหมาะสม เพราะต้องใช้จำนวนบิตในการแทนข้อมูลจำนวนมากซึ่งเป็นการเปลืองเนื้อที่ ดังนั้นจึงมีการนำการแทนข้อมูลแบบจุดลอยตัวเข้ามาใช้ ซึ่งจะอยู่ในรูป

$$f_{-1} f_{-2} \dots f_{-m} \times R^E$$

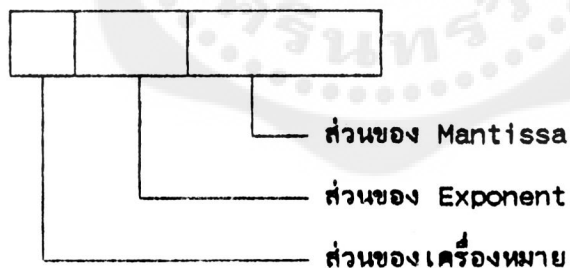
โดยที่ $f_{-1}f_{-2}\dots f_{-m}$ เรียกว่า Mantissa หรือ Fraction part

และ E (เป็นเลขจำนวนเต็ม) เรียกว่า Exponent

เช่น 16,800,000,000,000 จะแทนด้วย $.168 \times 10^{14}$

.0000000000832 จะแทนด้วย $.832 \times 10^{-10}$ เป็นต้น

การจัดเนื้อที่ในการแทนเลขจำนวนจริง แบ่งเป็น 3 ส่วน ดังรูป 1.5



รูป 1.5 การแทนเลขจำนวนจริงแบบจุดลอยตัว

การกำหนดจำนวนบิตในแต่ละส่วนที่แตกต่างกันจะทำให้ขอบเขตของค่าที่ใช้ได้แตกต่างกันไปด้วย การแทนข้อมูลในลักษณะดังกล่าวในบางครั้งอาจให้ค่าที่ผิดพลาดไปจากความเป็นจริงบ้าง เช่น การใช้ค่า $.16817 \times 10^{14}$ แทนค่า 16,817,210,391,704 เป็นต้น ค่าผิดพลาดดังกล่าว เรียกว่า ค่าผิดพลาดปัดเศษ (Rounding Error)

1.3.2 ข้อมูลแบบอักขระ

ข้อมูลแบบอักขระ (Character) ได้แก่ ข้อมูลที่เป็นตัวอักขระต่าง ๆ ที่ใช้ในระบบคอมพิวเตอร์จะถูกแปลงเป็นรหัสซึ่งกำหนดไว้แน่นอนสำหรับแทนแต่ละอักขระ ระบบรหัสที่ใช้กันในปัจจุบันได้แก่ รหัสแอสกี (ASCII Code) และรหัสเอ็บซีดิก (EBCDIC) โดยแต่ละอักขระใช้เนื้อที่หนึ่งไบต์ในหน่วยความจำ ในกรณีที่เก็บข้อมูลแบบข้อความหรือสายอักขระ (Character String) มักใช้เนื้อที่เป็นจำนวนไบต์เท่ากับจำนวนอักขระบวกด้วยหนึ่ง โดยใช้ไบต์แรกบอกความยาวของสายอักขระ และไบต์ต่อ ๆ มาเก็บสายอักขระ การกำหนดไบต์แรกเป็นตัวเก็บความยาวของสายอักขระทำให้มีข้อจำกัดเกี่ยวกับความยาว เนื่องจากค่าสูงสุดที่เก็บได้ในหนึ่งไบต์คือ 255 ดังนั้นจะเห็นว่าภาษาคอมพิวเตอร์บางภาษา เช่น เทอร์โบปาสกาลกำหนดความยาวสายอักขระได้สูงสุด 255 ตัว

1.3.3 ข้อมูลแบบบูล

ข้อมูลแบบบูล (Boolean) เป็นค่าแบบตรรกะซึ่งมีค่าที่เป็นได้คือ จริง (TRUE) หรือ เท็จ (FALSE) เท่านั้น การแทนค่าข้อมูลดังกล่าวอาจใช้ค่าเลขฐานสอง เช่น ในเทอร์โบปาสกาล ใช้ 0 แทน เท็จ และ 1 แทน จริง และใช้เนื้อที่ 1 ไบต์

1.3.4 ข้อมูลแบบตัวชี้

ข้อมูลแบบตัวชี้ (Pointer) เป็นข้อมูลที่เป็นค่าของเลขที่อยู่ของหน่วยความจำที่เป็นส่วนเก็บข้อมูลที่ต้องการอ้างอิงถึงนั่นเอง บางครั้งจึงเรียกว่า ลิงค์ (Link) การกำหนดเนื้อที่สำหรับข้อมูลแบบตัวชี้ ต้องกำหนดเพื่อให้สามารถอ้างอิงถึง ทุกตำแหน่งในหน่วยความจำได้ ตัวอย่างเช่น เทอร์โบปาสกาลกำหนดให้ใช้เนื้อที่สองเท่าของเวิร์ด (4 ไบต์) โดยเก็บค่าเซกเมนต์ไว้ 2 ไบต์สูง และค่าออฟเซตไว้ 2 ไบต์ต่ำ ค่า Nil เป็นค่าคงที่แสดงว่าไม่มีการชี้ไปที่ตำแหน่งใด

1.4 ประเภทของโครงสร้างข้อมูล

แบ่งเป็นประเภทใหญ่ ๆ ได้ 2 ประเภท คือ

1. โครงสร้างข้อมูลแบบเชิงเส้น
2. โครงสร้างข้อมูลแบบไม่เป็นเชิงเส้น

1.4.1 โครงสร้างข้อมูลแบบเชิงเส้น (Linear Data Structures) ได้แก่ โครงสร้างการจัดเก็บข้อมูลที่มีข้อมูลแต่ละหน่วยถูกบันทึกอย่างต่อเนื่องไปตามลำดับ มีลักษณะเหมือนเป็นแถวหรือเป็นเส้นตรง ได้แก่ โครงสร้างแบบ แถวลำดับ (Array) สแตก (Stack) คิว (Queue) และรายการโยง (Linked List)

1.4.2 โครงสร้างข้อมูลแบบไม่เป็นเชิงเส้น (Non-Linear Data Structures) ได้แก่ โครงสร้างการจัดเก็บข้อมูลที่มีข้อมูลไม่ได้มีลักษณะการบันทึกอย่างเป็นแถวหรือเป็นเชิงเส้น เช่น อาจเป็นลำดับชั้นหรือแบบอื่น ๆ ซึ่งมักขึ้นกับลักษณะความสัมพันธ์ของข้อมูล ได้แก่ โครงสร้างแบบต้นไม้ (Tree) และกราฟ (Graph)

1.5 การจัดสรรเนื้อที่สำหรับข้อมูลในหน่วยความจำ

การจัดสรรเนื้อที่ในหน่วยความจำ (Memory Allocation) สำหรับข้อมูลที่ใช้ในโปรแกรม สามารถทำได้ 2 แบบ คือ

1. การจัดสรรเนื้อที่แบบสถติก
2. การจัดสรรเนื้อที่แบบไดนามิก

1.5.1 การจัดสรรเนื้อที่แบบสถติก (Static Memory Allocation) หมายถึง ข้อมูลทุกตัวที่ใช้ในโปรแกรม จะได้รับการจัดสรรเนื้อที่ไว้ในหน่วยความจำก่อนการประมวลผลโดยขนาดเนื้อที่ของข้อมูลจะจำกัดและมีขนาดคงที่ตลอดเวลาของการประมวลผลโปรแกรม ขนาดเนื้อที่สำหรับข้อมูลที่จะใช้ในโปรแกรมจะเป็นข้อมูลที่ได้จากคอมไพเลอร์ ซึ่งก็ได้มาจากการประกาศตัวแปรในโปรแกรมนั่นเอง

1.5.2 การจัดสรรเนื้อที่แบบไดนามิก (Dynamic Memory Allocation) หมายถึง เนื้อที่สำหรับข้อมูลที่จะใช้ในโปรแกรมนั้นจะได้รับการจัดสรรให้ เมื่อพบคำสั่งขอเนื้อที่ในขณะที่กำลังประมวลผลโปรแกรม ขนาดของเนื้อที่ที่จัดสรรให้ จะเท่ากับที่ได้ประกาศไว้ในโปรแกรม เมื่อประมวลผลถึงคำสั่งขอเนื้อที่ ระบบจะหาเนื้อที่ที่ว่างหน่วยความจำที่จัดไว้สำหรับข้อมูลแบบได

นามิก และจัดให้เป็นเนื้อที่สำหรับข้อมูลนั้น ดังนั้นเนื้อที่สำหรับข้อมูลในโปรแกรมจึงสามารถเพิ่มหรือ
เปลี่ยนขนาดได้ในระหว่างการประมวลผล และถ้าเลิกใช้ข้อมูลตัวใดแล้วสามารถใช้คำสั่งคืนเนื้อที่
ให้กับระบบได้ ระบบก็จะเปลี่ยนสถานะเนื้อที่ส่วนนั้นเป็นที่ว่างที่สามารถนำไปใช้งานอื่นได้ (De-
allocate) คำสั่งสำหรับการจัดสรรเนื้อที่แบบไดนามิกจะมีเฉพาะในบางภาษาเท่านั้น เช่น ภาษา
ปาสคาล ฟอแทรน ซี เป็นต้น ซึ่งต้องมีการใช้ตัวแปรแบบตัวชี้เข้ามาเกี่ยวข้อง



แบบฝึกหัดบทที่ 1

1. อธิบายความหมายของโครงสร้างข้อมูล
2. อธิบายความหมายของ บิต ไบต์ เวิร์ด และเลขที่อยู่ของหน่วยความจำ
3. ช่วงค่าที่ใช้ได้ของข้อมูลแบบจำนวนจะมากหรือน้อยขึ้นอยู่กับอะไร
4. ประเภทของโครงสร้างข้อมูลแบ่งได้กี่ประเภท อะไรบ้าง
5. การจัดสรรเนื้อที่แบบสถิตกับแบบไดนามิก มีหลักการที่แตกต่างกันอย่างไร และการจัดสรรเนื้อที่แบบไดนามิกมีประโยชน์อย่างไร
6. การพิจารณาประสิทธิภาพของขั้นตอนวิธีพิจารณาจากอะไร
7. คณิตศาสตร์และทฤษฎีการคำนวณเกี่ยวกับการแทนรหัสภาษาไทยในไมโครคอมพิวเตอร์
8. ทฤษฎีการคำนวณเกี่ยวกับลักษณะของสื่อบันทึก โครงสร้างการจัดเก็บข้อมูล ความจุ ความเร็วในการถ่ายทอดข้อมูลของเทปและจานบันทึกข้อมูลแบบต่าง ๆ

บทที่ 2

การเรียงลำดับข้อมูลและการค้นข้อมูล

- หัวข้อ :
- 2.1 การเรียงลำดับข้อมูล
 - 2.2 ประเภทการเรียงลำดับข้อมูล
 - 2.3 หลักการเรียงลำดับข้อมูล
 - 2.4 การพิจารณาประสิทธิภาพของการเรียงลำดับข้อมูล
 - 2.5 ฟังก์ชันบิก โอ
 - 2.6 การเรียงลำดับแบบพอง
 - 2.7 การเรียงลำดับแบบคัดเลือก
 - 2.8 การเรียงลำดับแบบแทรก
 - 2.9 การเรียงลำดับแบบควิก
 - 2.10 การค้นข้อมูล
 - 2.11 ตัวอย่างโปรแกรม

การเรียงลำดับข้อมูล (Sorting) และการค้นข้อมูล (Searching) เป็นงานพื้นฐานซึ่งถูกกระทำบ่อยมากในระบบงานคอมพิวเตอร์ ไม่ว่าจะเป็นเกี่ยวข้องกับการทำงานของโปรแกรมระบบ เช่น ระบบปฏิบัติการ (Operating System) โปรแกรมประยุกต์ต่าง ๆ โดยเฉพาะอย่างยิ่งในระบบการจัดการฐานข้อมูล (Data Base Management System) ซึ่งเกี่ยวข้องกับการจัดเก็บและค้นคืนข้อมูลตลอดเวลา

2.1 การเรียงลำดับข้อมูล

การเรียงลำดับข้อมูล เป็นงานที่ใช้เวลาของซีพียูมากโดยเฉพาะเมื่อข้อมูลมาก แต่ก็ป็นงานที่จำเป็นต้องกระทำอยู่เสมอ ดังนั้นการใช้ขั้นตอนวิธีที่มีประสิทธิภาพจึงเป็นสิ่งจำเป็น ในบทนี้จะกล่าวถึงการเรียงลำดับข้อมูลแบบต่าง ๆ และประสิทธิภาพของแต่ละวิธี

การเรียงลำดับข้อมูลหมายถึง การเรียงลำดับข้อมูลชุดหนึ่งให้อยู่ในลำดับตามค่าใดค่าหนึ่งซึ่งใช้เป็นหลักในการเรียงลำดับ เรียกว่า คีย์ของการเรียงลำดับ (Sortkey) โดยอาจเรียงแบบลำดับขึ้น (Increasing order) หรือแบบลำดับลง (Decreasing order) ตามความเหมาะสมในการใช้ข้อมูลเพื่อการประมวลผล

2.2 ประเภทการเรียงลำดับข้อมูล

การเรียงลำดับข้อมูลสามารถแบ่งได้เป็น 2 ประเภท ได้แก่

- การเรียงลำดับข้อมูลที่อยู่ภายในหน่วยความจำหลัก (Internal Sorting)
- และ - การเรียงลำดับข้อมูลที่อยู่ในหน่วยความจำภายนอก (External Sorting)

ข้อมูลทุกค่าที่อยู่ภายในหน่วยความจำหลักจะสามารถอ้างอิงถึงได้ตลอดเวลา ต่างกับข้อมูลที่บันทึกอยู่ในหน่วยความจำภายนอก ซึ่งในขณะหนึ่งจะมีเพียงบางส่วนเท่านั้นที่อยู่ภายในหน่วยความจำหลัก เมื่อต้องการอ้างอิงถึงข้อมูลส่วนอื่น ๆ จำเป็นต้องทำการอ่านเข้ามาและทำการบันทึกข้อมูลกลับลงในหน่วยความจำภายนอก การอ่าน/เขียนข้อมูลกับหน่วยความจำภายนอกเป็นงานที่ใช้เวลามากเมื่อเทียบกับเวลาที่ใช้ในการประมวลผล ซึ่งเป็นปัจจัยที่ต้องพิจารณาในการสร้างขั้นตอนวิธีในการเรียงลำดับข้อมูลในหน่วยความจำภายนอก ดังนั้นการเรียงลำดับข้อมูลทั้ง 2 แบบจึงต้องใช้หลักการที่แตกต่างกัน

ในที่นี้จะกล่าวถึงเฉพาะการเรียงลำดับข้อมูลในหน่วยความจำหลัก โดยจะประยุกต์ใช้กับการเรียงลำดับข้อมูลในแฟ้มสุ่ม (Random File) ด้วย เนื่องจากการเข้าถึงข้อมูลในแฟ้มสุ่มคล้ายกับการเข้าถึงข้อมูลในแถวลำดับ โดยแฟ้มสุ่มใช้หมายเลขระเบียบซึ่งมีลำดับเป็น 0,1,2,... ตามลำดับ

2.3 หลักการเรียงลำดับข้อมูล

การเรียงลำดับข้อมูลในแถวลำดับ มีหลักการสำคัญอยู่ 3 แบบ คือ

1. หลักการสลับค่า
2. หลักการคัดเลือกค่า
3. หลักการแทรกค่า

2.3.1 หลักการสลับค่า

ใช้การเปรียบเทียบข้อมูล 2 รายการที่อยู่ติดกันว่าอยู่ในลำดับถูกต้องหรือไม่ ถ้าไม่ ให้ทำการสลับค่าข้อมูล 2 รายการนั้น กระทำเป็นลำดับต่อเนื่องกันไปจนกว่าข้อมูลทั้งหมดจะเรียงลำดับอย่างถูกต้อง

2.3.2 หลักการคัดเลือกค่า

ใช้หลักการคัดเลือกข้อมูลที่มีค่าน้อยที่สุดจากชุดข้อมูลที่มีอยู่มาทีละ 1 รายการ แล้วมาเก็บเรียงต่อกันตามลำดับ ในที่สุดก็จะได้ข้อมูลที่จัดเรียงอย่างถูกต้อง

2.3.3 หลักการแทรกค่า

ใช้หลักการนำข้อมูลมาเก็บวางทีละรายการ โดยจัดแทรกข้อมูลแต่ละรายการลงไปให้อยู่ในลำดับที่ถูกต้องตามค่าของข้อมูลนั้นเทียบกับข้อมูลที่มีอยู่เดิม

2.4 การพิจารณาประสิทธิภาพของการเรียงลำดับข้อมูล

การใช้ขั้นตอนวิธีในการเรียงลำดับที่แตกต่างกัน ย่อมทำให้ประสิทธิภาพของการทำงานแตกต่างกันไปบ้าง โดยทั่วไปประสิทธิภาพของขั้นตอนวิธี อาจพิจารณาจากปัจจัยต่าง ๆ ประกอบกัน ได้แก่

- ความเร็วในการเรียงลำดับข้อมูลโดยเฉลี่ย
- ความเร็วในการเรียงลำดับข้อมูล กรณีที่ดีที่สุด (Best Case) และกรณีเลวที่สุด (Worst Case)

- การทำงานของขั้นตอนวิธี ถ้าข้อมูลจัดเรียงอยู่แล้ว
 - การทำงานของขั้นตอนวิธี เมื่อพบข้อมูลที่มีคีย์เท่ากัน
- ความเร็วในการเรียงลำดับข้อมูล พิจารณาจาก

(1) จำนวนครั้งของการเปรียบเทียบ หรือตรวจสอบเงื่อนไข (Number of comparison)

: ถ้าจำนวนครั้งที่ทำการเปรียบเทียบข้อมูลมาก การทำงานจะช้า

(2) จำนวนครั้งของการสลับค่า (Number of exchange)

: ถ้าต้องทำการสลับค่าข้อมูลบ่อย การทำงานจะช้า

กรณีที่ดีที่สุด และกรณีเลวที่สุด เป็นการพิจารณาผลการทำงานของขั้นตอนวิธีประกอบกับลักษณะของชุดข้อมูลที่เข้ามา กรณีที่ดีที่สุด เกิดขึ้นเนื่องจากลักษณะของชุดข้อมูลบางแบบ จะทำให้ขั้นตอนวิธีนั้นทำงานน้อยที่สุด ดังนั้นจะทำงานเร็วที่สุดด้วย และในทางตรงกันข้าม กรณีเลวที่สุด เกิดขึ้นเนื่องจากลักษณะชุดข้อมูลบางแบบ จะทำให้ขั้นตอนวิธีต้องทำงานมากที่สุด แต่โดยทั่วไปเรามักพิจารณาความเร็วของการเรียงลำดับข้อมูลโดยเฉลี่ย

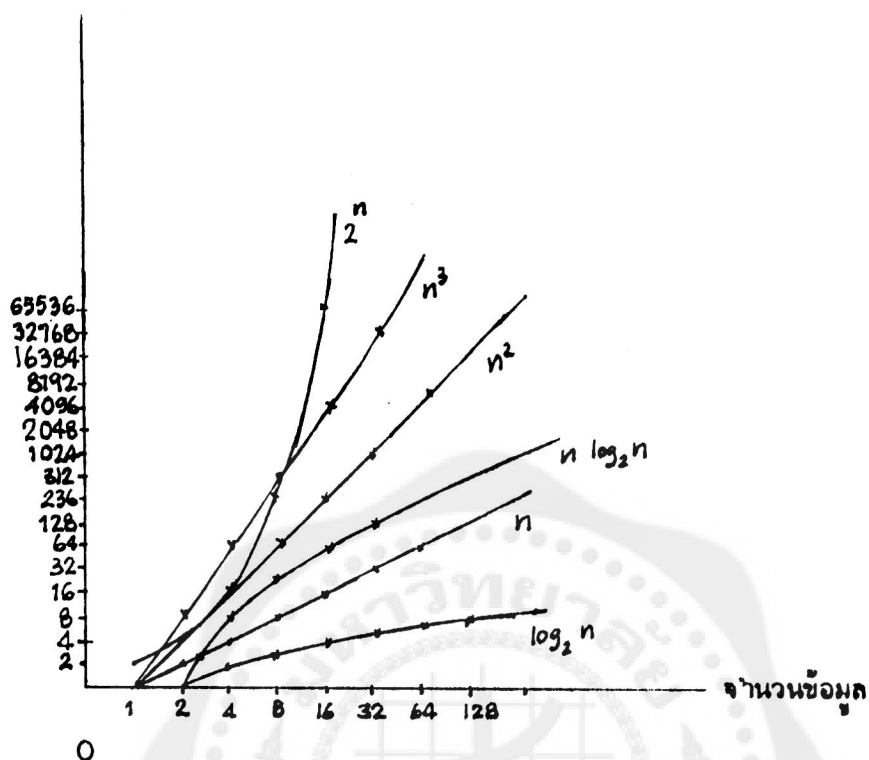
การทำงานของขั้นตอนวิธีถ้าข้อมูลจัดเรียงอยู่แล้ว ขั้นตอนวิธีที่ดีควรทำงานสอดคล้องกับพฤติกรรมโดยปกติที่ควรเป็น กล่าวคือ ถ้าข้อมูลที่เราเข้ามาเรียงลำดับอยู่แล้วก็ควรทำงานน้อย ถ้าข้อมูลไม่เรียงลำดับ การทำงานก็มาก ถ้าข้อมูลที่เราเข้ามาเรียงสลับลำดับ (reverse order) ก็ทำงานมากที่สุด

การทำงานของขั้นตอนวิธีเมื่อพบข้อมูลที่มีค่าซ้ำกัน ไม่ควรทำการสลับค่า หรือสับตำแหน่งของข้อมูลที่มีค่าซ้ำกัน เพราะเป็นการเสียเวลาเปล่า

2.5 ฟังก์ชันบิกโอ

ฟังก์ชันบิกโอ (Big Oh) เป็นฟังก์ชันที่แสดงถึงเวลาการประมวลผลของโปรแกรมที่เพิ่มขึ้น เมื่อจำนวนข้อมูล (n) มากขึ้น เวลาที่ใช้ในการเรียงลำดับข้อมูล มักพิจารณาจากจำนวนครั้งของการเปรียบเทียบข้อมูล และจำนวนครั้งของการสลับค่า ตัวอย่างเช่น การเรียงลำดับข้อมูลแบบหนึ่งต้องการเปรียบเทียบข้อมูลทั้งสิ้น $(n^2 - n)/2$ ครั้ง ดังนั้นเวลาที่ใช้ในการเรียงลำดับข้อมูลจะเป็นฟังก์ชันซึ่งขึ้นกับเทอมยกกำลังสองของ n ซึ่งเขียนด้วยสัญลักษณ์บิกโอ ในรูป $O(n^2)$ เราสามารถใช้ฟังก์ชันบิกโอช่วยในการเปรียบเทียบประสิทธิภาพของขั้นตอนวิธีได้ เช่น ขั้นตอนวิธีในการเรียงลำดับแบบหนึ่งใช้เวลาในรูป $O(n^2)$ จะดีกว่าแบบ $O(n^3)$ และดีกว่าแบบ $O(n^4)$ เป็นต้น หรืออาจพิจารณาจากกราฟแนวค่าของฟังก์ชันต่าง ๆ โดยประมาณ ดังรูป 2.1

เวลาประมวลผล



รูป 2.1 เปรียบเทียบเวลาที่เพิ่มขึ้นเมื่อจำนวนข้อมูลมากขึ้น

2.6 การเรียงลำดับแบบฟอง

การเรียงลำดับแบบฟอง (Bubble Sort) เป็นวิธีการที่รู้จักกันทั่วไป แต่จัดเป็นวิธีการที่แย่ที่สุดเมื่อเทียบกับการเรียงลำดับแบบอื่น ๆ การเรียงลำดับข้อมูลแบบฟองใช้หลักการสลับค่าข้อมูลทีละคู่ที่อยู่ติดกันให้อยู่ในลำดับที่ถูกต้อง

ส่วนการประกาศในโปรแกรมหลัก กำหนดไว้ดังนี้

```
Const Max = 1000;
```

```
Type
```

```
DataItem = Integer;
```

```
List = Array [1..max] of DataItem;
```

```

Procedure Bubblesort (Var a:list; count : Integer);
Var i,j : Integer;
    temp : DataItem;
Begin
    For i := 2 To count Do
        Begin
            For j := count Down To i Do
                If a[j-1] > a[j] Then
                    Begin {swap}
                        temp := a[j-1];
                        a[j-1] := a[j];
                        a[j] := temp
                    End
                End
            End
        End
    End; {Bubble sort}

```

รูป 2.2 ขั้นตอนการเรียงลำดับแบบฟอง

ประสิทธิภาพการทำงาน

พิจารณาจากจำนวนครั้งของการเปรียบเทียบ = $(n-1)+(n-2)+\dots+3+2+1=n(n-1)/2$
หรืออยู่ในรูป $O(n^2)$

ตัวอย่างการเรียงลำดับแบบพอง : จำนวนข้อมูล = 10

ข้อมูล :	.	78	93	91	41	73	12	0	41	83	10
รอบที่ 1	0	78	93	91	41	73	12	10	41	83	
รอบที่ 2	0	10	78	93	91	41	73	12	41	83	
รอบที่ 3	0	10	12	78	93	91	41	73	83	83	
รอบที่ 4	0	10	12	41	78	93	91	41	73	83	
รอบที่ 5	0	10	12	41	41	78	93	91	73	83	
รอบที่ 6	0	10	12	41	41	73	78	93	91	83	
รอบที่ 7	0	10	12	41	41	73	78	83	93	91	
รอบที่ 8	0	10	12	41	41	73	78	83	91	93	
รอบที่ 9	0	10	12	41	41	73	78	83	91	93	
ผลลัพธ์	0	10	12	41	41	73	78	83	91	93	

รูป 2.3 ตัวอย่างการเรียงลำดับแบบพอง

2.7 การเรียงลำดับแบบคัดเลือก

การเรียงลำดับแบบคัดเลือก (Selection Sort) ใช้วิธีการคัดเลือกข้อมูลที่มีค่าน้อยที่สุดจากชุดข้อมูลที่มีอยู่มาใส่ในตำแหน่งที่ 1,2,3...ไปตามลำดับ ขั้นตอนวิธีโดยสรุปเป็นดังนี้

```
For i:= 1 To n-1 Do
    Select the smallest among A[i]...A[n]
    and swap it with A[i];
```



```

Procedure Selectsort (Var a:list; count:integer);
Var   i,j,k   : Integer;
      temp    : Dataitem;
Begin
  For i := 1 To count-1 Do
    Begin
      k := i;
      temp := a[k];
      For j := i+1 To count Do
        {find Smallest item}
        If a[j] < temp Then
          Begin
            k := j;
            temp := a[j]
          End;
        {swap item[k] with item[i]}
        a[k] := a[i];
        a[i] := temp;
      End
    End; {Selection sort}
  End;

```

รูป 2.4 ขั้นตอนการเรียงลำดับแบบคัดเลือก

ประสิทธิภาพการทำงาน พิจารณาจากจำนวนครั้งของการเปรียบเทียบ = $(n^2-n)/2$
 นั่นคืออยู่ในรูป $O(n^2)$

ตัวอย่างการเรียงลำดับแบบคัดเลือก : จำนวนข้อมูล = 10

ข้อมูล	:	66	23	43	69	29	90	82	46	70	26
รอบที่ 1	:	(23)	66	43	69	29	90	82	46	70	(26)
รอบที่ 2	:	23	(26)	43	69	(29)	90	82	46	70	66
รอบที่ 3	:	23	26	(29)	69	(43)	90	82	46	70	66
รอบที่ 4	:	23	26	29	(43)	69	90	82	(46)	70	66
รอบที่ 5	:	23	26	29	43	(46)	90	82	69	70	(66)
รอบที่ 6	:	23	26	29	43	46	(66)	82	(69)	70	90
รอบที่ 7	:	23	26	29	43	46	66	(69)	82	(70)	90
รอบที่ 8	:	23	26	29	43	46	66	69	(70)	(82)	90
รอบที่ 9	:	23	26	29	43	46	66	69	70	(82)	90
ผลลัพธ์	:	23	26	29	43	46	66	69	70	82	90

รูป 2.5 ตัวอย่างการเรียงลำดับแบบคัดเลือก

2.8 การเรียงลำดับแบบแทรก

การเรียงลำดับแบบแทรก (Insertion Sort) ใช้วิธีแทรกข้อมูลตัวที่ $A[i]$ เข้าไปอยู่ในลำดับที่ถูกต้องของข้อมูลนั้นใน $A[1] \dots A[i-1]$ ซึ่งมีค่าที่เรียงลำดับอยู่แล้ว ดังนั้นผลในรอบที่ i จะได้ค่า $A[1] \dots A[i]$ ที่เรียงลำดับถูกต้อง

```

Procedure Insertsort (Var a:List; Count:Integer);
Var   i,j : Integer;
      temp : Dataitem;
Begin
  For i := 2 To count Do
  Begin
    temp := a[i];
    j    := i-1;
    While (temp < a[j]) AND (j>0) Do
      Begin
        a[j+1] := a[j];
        j      := j-1;
      End;
    a[j+1] := temp;
  End;
End; {Insertion Sort}

```

รูป 2.6 ขั้นตอนการเรียงลำดับแบบแทรก

ประสิทธิภาพการทำงานของ อัลกอริทึมจากจำนวนครั้งการเปรียบเทียบข้อมูลจะอยู่ในรูป $O(n^2)$ แต่จะ
ใช้ได้ดีเมื่อข้อมูลค่อนข้างเรียงลำดับอยู่แล้ว

จากขั้นตอนวิธีของทั้ง 3 วิธีที่กล่าวมานี้ ถึงแม้จะมีประสิทธิภาพในรูป $O(n^2)$ ทั้งสิ้น
ซึ่งจะ ไม่ได้ดีนักเมื่อปริมาณข้อมูลมาก

ตัวอย่างการเรียงลำดับแบบแทรก : จำนวนข้อมูล = 10

ข้อมูล	:	66	23	43	69	29	90	82	46	70	26
รอบที่ 1	:	(23)	66	43	69	29	90	82	46	70	26
รอบที่ 2	:	23	(66)	43	69	29	90	82	46	70	26
รอบที่ 3	:	23	(43)	66	69	29	90	82	46	70	26
รอบที่ 4	:	23	43	(66)	(69)	29	90	82	46	70	26
รอบที่ 5	:	23	(29)	43	66	69	90	82	46	70	26
รอบที่ 6	:	23	29	43	66	69	(90)	82	46	70	26
รอบที่ 7	:	23	29	43	46	66	69	(82)	90	70	26
รอบที่ 8	:	23	29	43	46	66	69	(70)	82	90	26
รอบที่ 9	:	23	(26)	29	43	46	66	69	70	82	90
ผลลัพธ์	:	23	26	29	43	46	66	69	70	82	90

รูป 2.7 ตัวอย่างการเรียงลำดับแบบแทรก

2.9 การเรียงลำดับแบบควิก

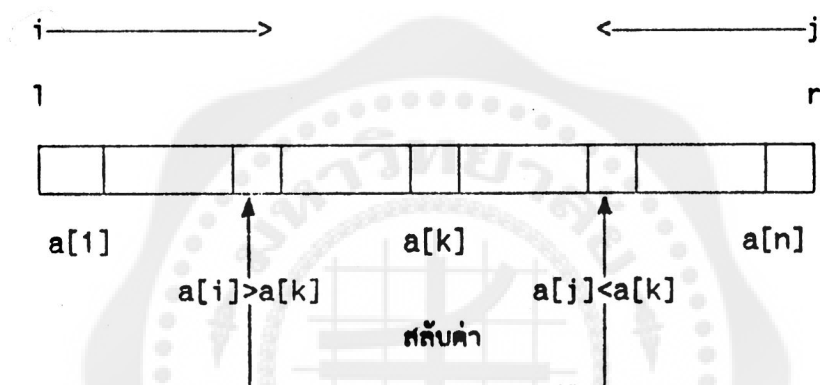
การเรียงลำดับแบบควิก (Quick Sort) เป็นวิธีการเรียงลำดับข้อมูลที่ใช้หลักเกณฑ์การสลับค่า และจัดว่าเป็นวิธีที่มีประสิทธิภาพดีที่สุด การทำงานของวิธีนี้จะใช้การจัดกลุ่มข้อมูล โดยกำหนดข้อมูลตัวหนึ่งเป็นตัวแบ่งกลุ่ม สมมติให้เป็น $a[k]$ เพื่อให้ค่าข้อมูลอยู่ในรูป

$$a[1]...a[k-1] \leq a[k] \leq a[k+1]...a[n]$$

เมื่อจัดกลุ่มข้อมูลได้ตั้งรูปแล้ว ข้อมูลในแต่ละกลุ่มก็จะนำไปจัดกลุ่มด้วยวิธีการแบบเดิมอีก และกระทำไปจนกระทั่งไม่สามารถแบ่งกลุ่มย่อยได้อีก ก็จะสิ้นสุดการทำงาน ซึ่งเป็นการทำงานแบบเรียกซ้ำ

การกำหนดข้อมูลที่เป็นตัวแบ่งกลุ่ม หรือ $a[k]$ อาจใช้ตัวกลางคือ $k = (l+r) / 2$ เมื่อ l : ตำแหน่งสมาชิกตัวแรก และ r : ตำแหน่งสมาชิกตัวสุดท้าย

วิธีการจัดกลุ่มใช้การค้นสมาชิกทางซ้ายของ $a[k]$ ซึ่งมีค่ามากกว่า $a[k]$ และสมาชิกทางขวาของ $a[k]$ ที่มีค่าน้อยกว่า $a[k]$ แล้วนำมาสลับที่กัน ดังรูป 2.8



รูป 2.8 วิธีการจัดกลุ่มข้อมูลในการเรียงลำดับแบบควิก

สลับค่า $a[i] \longleftrightarrow a[j]$ เมื่อพบ $a[i] > a[k]$ และ $a[j] < a[k]$ และหยุดการทำงานเมื่อ $i > j$ ผลการจัดกลุ่มจะได้ข้อมูล 2 กลุ่มดังนี้



ถ้า $j > 1$ ให้นำข้อมูล $a[1] \dots a[j]$ ไปจัดเรียงด้วยวิธีเดิม

ถ้า $i < r$ ให้นำข้อมูล $a[i] \dots a[r]$ ไปจัดเรียงด้วยวิธีเดิม

```

Procedure QuickSort (Var a:List; Lo,Hi : Integer);
{.....sort.....}
  procedure sort (l,r : Integer);
  Var   i,j,k : Integer;
        x,y   : Dataitem;
  Begin
    i :=l; j:=r; k:= (l+r) DIV 2;
    x := a[k];
    Repeat
      While a[i]<x Do i := i+1;
      While a[j]>x Do j := j-1;
      If i<=j Then
        Begin
          y := a[i]; a[i] :=a[j]; a[j] :=y;
          i:= i+1; j:=j-1
        End
      Until i>j;
      If j<=i Then sort (l,j);
      If i<=r Then sort (i,r)
    End; {sort}
  {.....}
  Begin [quicksort]
    sort (Lo,Hi)
  End;

```

รูป 2.9 ขั้นตอนเรียงลำดับแบบควิก

ประสิทธิภาพของการเรียงลำดับแบบควิก

พิจารณาจำนวนครั้งของการเปรียบเทียบ = $n+2(n/2)+4(n/4)+8(n/8)+\dots+n(n/n)$ ซึ่งมี $\log_2 n$ เทอม ดังนั้นจำนวนครั้งการเปรียบเทียบเท่ากับ $n(\log_2 n)$ หรือมีประสิทธิภาพเป็น $O(n \log_2 n)$

ตารางต่อไปนี้แสดงเวลาที่ใช้ในการเรียงลำดับข้อมูล เพื่อเปรียบเทียบประสิทธิภาพการเรียงลำดับแต่ละแบบ เมื่อจำนวนข้อมูลเพิ่มขึ้น และเมื่อข้อมูลเข้ามีลักษณะแตกต่างกัน

จำนวนข้อมูล = 256	ลักษณะข้อมูลนำเข้า		
การเรียงลำดับ	เรียงลำดับแล้ว	สุม	เรียงสลับลำดับ
แบบพอง	1.26	2.04	2.80
แบบแทรก	0.2	0.82	1.64
แบบคัดเลือก	0.94	0.96	1.18
แบบควิก	0.08	0.12	0.08

จำนวนข้อมูล = 2048	ลักษณะข้อมูลนำเข้า		
การเรียงลำดับ	เรียงลำดับแล้ว	สุม	เรียงสลับลำดับ
แบบพอง	80.18	128.84	178.66
แบบแทรก	0.22	50.74	103.80
แบบคัดเลือก	58.18	58.34	73.46
แบบควิก	0.72	1.22	0.76

ตาราง 2-1 เปรียบเทียบเวลาการเรียงลำดับข้อมูลแบบต่างๆ

2.10 การค้นข้อมูล

การค้นข้อมูลเป็นงานหนึ่งที่มีความสำคัญอย่างยิ่ง สำหรับงานในระบบคอมพิวเตอร์ ซึ่งประกอบด้วยการจัดเก็บข้อมูล (store) และค้นคืนข้อมูลกลับมาใช้งาน (retrieve) การค้นคืนข้อมูลที่ต้องการนั้น ต้องอาศัยเทคนิคในการค้นข้อมูลซึ่งจะมีวิธีการที่แตกต่างกันไปตามโครงสร้างของข้อมูลแต่ละแบบ

ถ้าข้อมูลที่จัดเก็บไม่ได้มีการเรียงลำดับไว้ตามค่าคีย์ที่กำหนด การค้นข้อมูลจะทำได้แบบตามลำดับ (Sequential Search) เท่านั้น แต่ถ้าข้อมูลได้เรียงลำดับไว้ตามคีย์ที่ต้องการแล้ว เราสามารถใช้วิธีการค้นข้อมูลแบบทวิภาค (Binary Search) ซึ่งจะช่วยให้มีความรวดเร็วในการค้นข้อมูลเป็นอย่างมาก

2.10.1 การค้นข้อมูลแบบตามลำดับ

การค้นข้อมูลแบบตามลำดับเป็นการตรวจสอบค่าข้อมูลที่บันทึกไว้ในหน่วยความจำไปทีละรายการ ตั้งแต่รายการแรกไปตามลำดับ จนกว่าจะพบข้อมูลที่มีค่าเท่ากับคีย์ที่กำหนด หรือหมดข้อมูล

ในที่นี้จะแสดงการค้นข้อมูลที่เก็บในแถวลำดับ กำหนดโครงสร้างข้อมูลไว้ดังนี้

```
Const Max = 100;
Type Dataitem = Integer;
Data_array = Array [1..max] of Dataitem;
Var item : Data_array;
    n : Integer;
    (n : จำนวนข้อมูล)
```

ขั้นตอนวิธีเป็นดังนี้

```
i := 1;
While (key <> item[i]) AND (i <= n) Do i := i+1
```


การออกจากวนซ้ำ อาจออกเนื่องจากการพบข้อมูล ($\text{key} = \text{item}[i]$) หรือเนื่องจากหมดข้อมูล ($i > n$)

ดังนั้น จึงต้องตรวจสอบผลการค้นข้อมูลหลังการออกจากวนซ้ำ ว่าพบข้อมูลหรือไม่ อาจตรวจสอบด้วยเงื่อนไข การพบข้อมูล หรือการหมดข้อมูลก็ได้ ในที่นี้ใช้เงื่อนไขการหมดข้อมูล และถ้าพบข้อมูลจะแสดงตำแหน่งของข้อมูลที่พบ เขียนคำสั่งได้ดังนี้

```
If i>n Then Writeln ('** Not Found **')
Else Writeln ('Found at Element # ',i );
```

สรุปขั้นตอนวิธีการค้นแบบตามลำดับ โดยเขียนเป็นโปรซีเจอร์ภาษาปาสคาลได้

ดังรูป 2.10

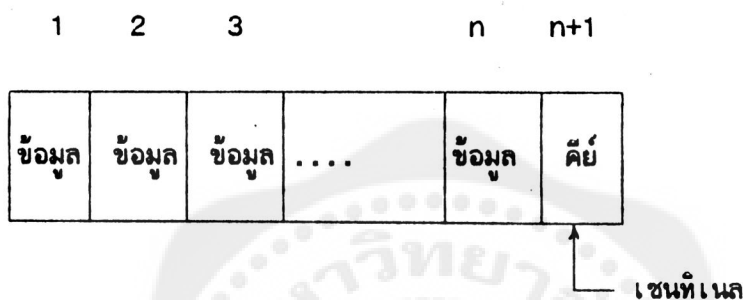
```
Procedure Sequential_search (Var A:Data_array; n :Integer;
                             Key : DataItem);
Var i :Integer;
Begin
    i := 1;
    While (A[i] <> Key) AND (i < n) Do
        i := i+1;
    If i > n Then
        Writeln('** Not Found **')
    Else Writeln('Found at Element # ',i);
End;
```

รูป 2.10 ขั้นตอนวิธีค้นข้อมูลแบบตามลำดับ

เทคนิคการใช้เซนทิเนล

การตรวจสอบด้วยเงื่อนไข ($key \neq item[i]$) AND ($i \leq n$) ทำให้การทำงานช้า เนื่องจากต้องตรวจสอบเงื่อนไขที่ซับซ้อน เราสามารถปรับปรุงขั้นตอนวิธีให้ทำงานเร็วขึ้นโดยใช้เพียงเงื่อนไขเดียว ทั้งนี้ต้องมีการปรับปรุงการใช้เนื้อที่ของข้อมูลเล็กน้อยคือจะต้องจัดเนื้อที่เพิ่มอีกหนึ่งที่ในส่วนท้ายข้อมูลเพื่อใช้เก็บค่าคีย์ที่ต้องการค้น เรียกว่า เซนทิเนล (sentinel)

จำนวนข้อมูลคือ n ใช้ตำแหน่งที่ $n+1$ เก็บค่าคีย์ที่ต้องการค้น



ดังนั้นเมื่อทำการค้นข้อมูลจะพบข้อมูลที่มีค่าเท่ากับคีย์เสมอ ซึ่งต้องตรวจสอบว่า ถ้าพบข้อมูลในตำแหน่งสุดท้ายหรือ $n+1$ แสดงว่าไม่มีข้อมูลที่ต้องการอยู่ในข้อมูลชุดนี้
ขั้นตอนวิธีการค้นปรับปรุงได้ดังนี้

```

Procedure Sequential_search2 (Var A:Data_array; n:Integer;
                               Key : DataItem);
Var i :Integer;
Begin
    A[n+1] := Key; {Sentinel}
    i := 1;
    While (A[i] <> Key) Do i:= i+1;
    If i = n+1 Then
        Writeln '** Not Found **')
    Else Writeln('Found at Element # ',i);
End;
```

รูป 2.11 ขั้นตอนวิธีค้นข้อมูลแบบตามลำดับใช้เซนทิเนล

พิจารณาจำนวนครั้งของการวนรอบสำหรับการค้นหาข้อมูลแบบตามลำดับ เป็นดังนี้

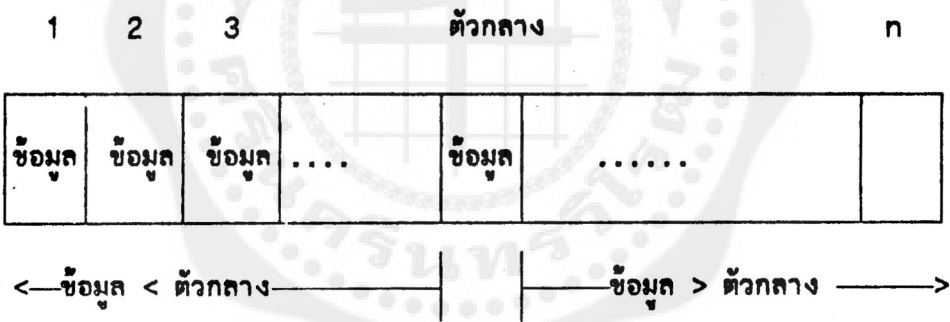
โดยเฉลี่ย : $n/2$

กรณีที่ดีที่สุด : 1

กรณีเลวที่สุด : n

2.10.2 การค้นหาข้อมูลแบบทวิภาค

ถ้าข้อมูลได้รับการเรียงลำดับตามลำดับของค่าเรียบร้อยแล้ว การค้นหาข้อมูลแบบทวิภาค จะช่วยให้ประสิทธิภาพการทำงานดีขึ้นมาก หลักการค้นหาแบบนี้จะใช้การเปรียบเทียบค่าที่ต้องการ ค้นหาข้อมูลตัวกลางของข้อมูลทั้งหมด ว่าเท่ากับค่าที่ต้องการหรือไม่ ถ้าไม่เท่าก็จะพิจารณาต่อว่า ถ้าค่าคีย์ น้อยกว่า ตัวกลางก็จะไปค้นหาข้อมูลในส่วนครึ่งที่น้อยกว่าตัวกลาง แต่ถ้าค่าคีย์มากกว่าตัวกลางก็จะไปค้นหาในส่วนครึ่งที่มากกว่าตัวกลางนั้น ดังรูป 2.12



รูป 2.12 ลำดับข้อมูลเมื่อเทียบกับค่าตัวกลาง

- กำหนด
- 1 : ตำแหน่งของข้อมูลตัวแรก
 - r : ตำแหน่งข้อมูลตัวสุดท้าย
 - mid : ตำแหน่งข้อมูลตัวกลาง
 - found : ตัวแปรแสดงผลการค้นหาข้อมูล

ตำแหน่งของตัวกลางคำนวณจาก $(1+r)/2$ หรือ $mid := (L+r) \text{ DIV } 2$

การเปรียบเทียบค่าคีย์ กับ ข้อมูลตัวกลางพิจารณาดังนี้

- ถ้า ข้อมูลตัวกลาง = ค่าคีย์ แสดงว่า พบข้อมูล
- ถ้า ข้อมูลตัวกลาง > ค่าคีย์ ไปค้นข้อมูลด้วยวิธีเดิมจากส่วนครึ่งที่น้อยกว่าตัวกลาง
(เปลี่ยนค่า $r = \text{mid} - 1$)
- ถ้า ข้อมูลตัวกลาง < ค่าคีย์ ไปค้นข้อมูลด้วยวิธีเดิมจากส่วนครึ่งที่มากกว่าตัวกลาง
(เปลี่ยนค่า $L = \text{mid} + 1$)

ขั้นตอนวิธีค้นข้อมูลแบบทวิภาค ต้องทำการวนซ้ำจนกว่าจะพบข้อมูล ซึ่งถ้าพบก็จะพบที่ข้อมูลตัวกลางเสมอ และกรณีไม่มีข้อมูลจะวนซ้ำจนหมดข้อมูล กล่าวคือไม่สามารถแบ่งส่วนข้อมูลได้อีก นั่นคือค่า L มากกว่า r หลังส่วนการวนซ้ำ ต้องตรวจสอบว่าพบข้อมูลหรือไม่ โดยตรวจสอบจากค่าของ `found` ถ้าค่าเป็นจริง (TRUE) แสดงว่า พบข้อมูลที่ตำแหน่ง `mid` นั้นเอง การเลือกเปรียบเทียบที่ตัวกลางจะช่วยเพิ่มประสิทธิภาพของการค้นข้อมูลได้เป็นอย่างดี เทียบกับการลดจำนวนข้อมูลที่ต้องเปรียบเทียบลงไปทีละครึ่งของจำนวนข้อมูลที่กำลังทำงานอยู่ ซึ่งจะประกันได้ว่า กรณีเลขที่สุดต้องทำการเปรียบเทียบไม่เกิน $\log_2 n$ ครั้ง

ขั้นตอนวิธีค้นข้อมูลแบบทวิภาค เขียนเป็นโปรซีเจอร์ได้ดังรูป 2.13

```

Procedure Binary_search (Var A:Data_array; n :Integer;
                        Key : DataItem);
Var L,r,mid :Integer;
    Found :Boolean;
Begin
    L:=1; r:=n, found := FALSE;
    While (L<=r) AND (NOT found) Do
        Begin
            mid := (L+r) DIV 2;
            If A[mid] = key Then found := TRUE
            Else If A[mid] < key Then L := mid+1
            Else R := mid-1;
        End; {While}
    If Found Then
        Writeln('Found at Element # ',mid)
    Else Writeln('** Not Found **');
End;

```

รูป 2.13 ขั้นตอนการค้นข้อมูลแบบทวิภาค

2.11 ตัวอย่างโปรแกรม

ในหัวข้อนี้จะแสดงตัวอย่างโปรแกรม การเรียงลำดับข้อมูลและการค้นข้อมูล เพื่อให้ผู้อ่านสามารถนำไปทดลองปฏิบัติได้ เพื่อให้เกิดความเข้าใจขั้นตอนวิธีมากขึ้น

ตัวอย่างที่ 1 โปรแกรม Selection_sort แสดงการเรียงลำดับแบบคัดเลือก โปรแกรมนี้อ่านข้อมูลเข้าจากแฟ้มข้อมูลในจานบันทึกที่ต้องสร้างไว้ล่วงหน้าแล้ว โดยข้อมูลที่บันทึกไว้เป็นเลขจำนวนเต็ม การจัดเรียงกระทำในแถวลำดับ และแสดงผลบนจอภาพ

```

Program Selection_Sort_Demo;
Uses    Crt;
{ This program demonstrates the selection sort algorithm }
{ by sorting data from disk file.                        }
{ Finally, sorted data is output on the screen.          }
Const  Max = 1000;
Type   DataItem = Integer;
       List      = Array [1..Max] of DataItem;
Var    finame    :string[15];
       fi        :File of DataItem;
       data      :List;
       i, n      :Integer;
       ch        :Char;

{-----}
Procedure Selection_sort(Var A:List; count :Integer);
Var  i,j,k,pass :Integer;
     temp :DataItem;
Begin
    pass := 0;
    For i := 1 to count-1 Do
        Begin
            pass := pass+1;
            k := i;
            temp := A[k];
            For j := i+1 to count Do {find smallest item}
                If A[j] < temp Then
                    Begin  k := j;
                           temp := A[j]
                    End;
            End;
            {Swap item(k) with item(i)}
            A[k] := A[i];
            A[i] := temp;
            {display output of each pass}
            Write('PASS ',pass:2,'  :');

```

```

        For k := 1 to count Do Write(A[k]:5); Writeln
    End
End; {Selection_sort}
{-----}

```

```

Begin {main}
    Clrscr;
    Writeln('*** SELECTION SORT ***':40); Writeln;
    Write('Enter data file name:'); Readln(finame);
    {read data from file into array}
    Assign(fi, finame);
    Reset(fi);
    Write('INPUT DATA:');
    i:=1;
    While NOT Eof(fi) Do
    Begin
        Read(fi, data[i]);
        Write(data[i]:5);
        i := i+1
    End;
    Writeln;
    Writeln('----- FOR K: -----');
    n := i-1; {number of data}
    Selection_sort(data, n); {call procedure}
    Writeln('-----');
    Write('OUTPUT   :');
    For i:=1 to n Do
        Write(data[i]:5);
    Writeln;
    Write('Press Any Key...');
    ch := Readkey;
    writeln
End.

```

ตัวอย่างผลการทำงานของโปรแกรม Selection_Sort_Demo

```
*** SELECTION SORT ***

Enter data file name:B:TEST.DAT
INPUT DATA:  66  23  43  69  29  90  82  46  70  26
-----
PASS 1  :  23  66  43  69  29  90  82  46  70  26
PASS 2  :  23  26  43  69  29  90  82  46  70  66
PASS 3  :  23  26  29  69  43  90  82  46  70  66
PASS 4  :  23  26  29  43  69  90  82  46  70  66
PASS 5  :  23  26  29  43  46  90  82  69  70  66
PASS 6  :  23  26  29  43  46  66  82  69  70  90
PASS 7  :  23  26  29  43  46  66  69  82  70  90
PASS 8  :  23  26  29  43  46  66  69  70  82  90
PASS 9  :  23  26  29  43  46  66  69  70  82  90
-----
OUTPUT   :  23  26  29  43  46  66  69  70  82  90

Press Any Key...._
```

หมายเหตุ ถ้าต้องการใช้วิธีทดลองโปรแกรมการเรียงลำดับแบบอื่นๆ ก็ทำได้โดยเปลี่ยนส่วน
โปรซีเจอร์ไปตามวิธีที่ต้องการ และปรับปรุงโปรแกรมเพียงเล็กน้อยเท่านั้น

ตัวอย่างที่ 2 โปรแกรม Quick_Sort_Demo แสดงการเรียงลำดับแบบควิก โปรแกรมนี้สร้าง
ข้อมูลทดสอบโดยใช้ฟังก์ชันเลขสุ่ม สร้างเลขจำนวนเต็มซึ่งมีค่าระหว่าง 0 ถึง 100 และ
ให้ผู้ใช้ระบุจำนวนข้อมูลทดสอบ โปรแกรมนี้ไม่มีการแสดงผลการทำงานแต่ละขั้นตอนเนื่องจาก
การเรียงลำดับใช้วิธีการเรียกซ้ำ เพียงแต่แสดงข้อมูลทดสอบและผลการเรียงลำดับบนจอภาพ

```

Program Quick_Sort_Demo;
{$R-,S-}
Uses Crt;

{ This program demonstrates the quicksort algorithm, which      }
{ provides an extremely efficient method of sorting arrays in    }
{ memory. The program generates a list of random numbers        }
{ between 0 and 100, and then sorts them using the QUICKSORT    }
{ procedure. Finally, the sorted list is output on the screen.   }
{ Note that stack and range checks are turn off (through the    }
{ compiler directive above) to optimize execution speed.        }

Const  Max = 1000;

Type   DataItem = Integer;
       List = Array[1..Max] of DataItem;
Var    data :List;
       i,num :Integer;
       ch :Char;

{-----}
Procedure Quicksort(Var a:List; Lo, Hi :Integer);
  Procedure Sort(l,r:Integer);
    Var  i,j  :Integer;
         x,y  :DataItem;
  Begin
    i := 1,j:=r;
    x := a[(1+r) DIV 2];
    Repeat
      While a[i] < x Do i:= i+1;
      While x < a[j] Do j:=j-1;
      If i<=j Then
        Begin
          y :=a[i]; a[i]:=a[j]; a[j]:= y;
          i := i+1; j:=j-1
        End;
      Until i>j;
  End;

```

```

        If l < j Then Sort(l,j);
        If i < r Then Sort(i,r);
    End; {Sort}
Begin
    Sort(Lo,Hi);
End; {Quicksort}
{-----}

Begin {Main}
    Clrscr;
    Writeln('*** QUICK SORT ***':40);
    Writeln;
    Write('Enter number of test data:');
    Readln(num);
    Writeln;
    Writeln('Now generating ',num,' random numbers...');
    Writeln;
    Randomize;

    For i := 1 to num Do
    Begin
        data[i] := Random(100);
        Write(data[i]:5)
    End;
    Writeln;
    Writeln('Now sorting random numbers...');
    Quicksort(data,1,num);
    Writeln;
    For i:=1 to num Do Write(data[i]); Writeln;
    Writeln;
    Write('Press Any Key...');
    ch := Readkey;
    Writeln
End.

```

ตัวอย่างการทำงานของโปรแกรม Quick_Sort_Demo

*** QUICK SORT ***

Enter number of test data:15

Now Gerating 15 random numbers...

89 33 89 69 26 61 43 99 49 89 15 97 14 79 75

Now Sorting random numbers...

14 15 26 33 43 49 61 69 75 79 89 89 89 97 99

Press Any Key..._

ตัวอย่างที่ 3 โปรแกรม Phone_sort เป็นตัวอย่างสำหรับเรียงลำดับข้อมูลในแฟ้มสุ่ม (Random File) โดยใช้การเรียงลำดับแบบควิก การอ้างอิงถึงข้อมูลแต่ละรายการใช้หมายเลขระเบียบในปาสกาลเริ่มตั้งแต่ 0,1,2,...,n ดังนั้นจึงดูเหมือนเป็นการเรียงลำดับข้อมูลแบบแถวลำดับในงานบันทึก ข้อมูลแต่ละระเบียบในแฟ้มข้อมูลประกอบด้วย ชื่อ และ หมายเลขโทรศัพท์

โปรแกรมนี้จะเรียงลำดับข้อมูลในแฟ้มชื่อ Phone.Dat ซึ่งบันทึกข้อมูลไว้แล้วอยู่ในงานบันทึก โดยเรียงลำดับข้อมูลตามอักขระของชื่อ และผลการเรียงลำดับบันทึกกลับลงบนแฟ้มเดิม ซึ่งจะเห็นจะว่าขั้นตอนวิธียังคงเป็นแบบเดิม เพียงแต่ปรับปรุงคำสั่งสำหรับอ้างอิงข้อมูลในแฟ้มเล็กน้อยเท่านั้น

```

Program Phonesort;
Uses  Crt;
Type  Phone_rec  = Record
        Name : String [15];
        Phone: String [15]
    End;

    Str15      = String [15];
    Dataitem = Phone_rec;
    Datafi    = File of Dataitem;
Var
    P      : Dataitem;
    Fi     : Datafi;
    n      : integer;

```

```
Function Find (Var fp : Datafi; i :Integer) : Str15;
Var t : Dataitem;
```

```
Begin
```

```
    i := i-1;      {record number begin with o}
```

```
    Seek (fp,i);
```

```
    Read (fp,t);
```

```
    Find := t.name {return name of i th person}
```

```
End;{Find}
```

```
Procedure Qsrand (Var fp :Datafi; count :Integer);
```

```
{--- Recursive Quick Sort ---}
```

```
Procedure Qs (l,r :Integer);
```

```
Var i,j,k : Integer;
```

```
    x,y,z : Dataitem;
```

```
Begin
```

```
    i := l; j:=r;
```

```
    k := (l+r) DIV 2;
```

```
    Seek (fp, k-1);
```

```
    Read (fp, x ); {get a record}
```

```
Repeat
```

```
    While find (fp,i) < x.name Do i:= i+1;
```

```
    While find (fp,j) > x.name Do j := j-1;
```

```
    If i < j Then
```

```
        Begin {swap records on disk}
```

```
            Seek (fp,i-1); Read (fp,y);
```

```
            Seek (fp,j-1); Read (fp,z);
```

```
            Seek (fp,j-1); Write(fp,y);
```

```
            Seek (fp,i-1); Write(fp,z);
```

```
            i := i+1; j=j-1;
```

```
        End;
```

```
Until i>j;
```

```
    If l < j Then Qs (l,j);
```

```
    If i < r Then Qs (i,r)
```

```
End; {Qs}
```

```

Begin
    Qs (1,count);
End; {Qsrand}
Begin {main}
    Clrscr;
    Writeln('*** SORT DATA RECORD IN RANDOM FILE ***');
    Assign (fi,'Phone.dat');
    Reset (fi);
    n := Filesize (fi);
    Writeln;
    Writeln('Please wait for file sorting');
    Qsrand (fi,n);
    Close (fi);
    Writeln('Now, file is sorted already');

    Writeln('*** List data from file ***');
    Reset(fi);
    while Not Eof(fi) Do
    Begin
        Read(fi,p);
        Writeln(p.name:15,p.phone:20);
    End;
    close(fi);
End.

```

ตัวอย่างที่ 4 โปรแกรม Phone_Search แสดงวิธีค้นข้อมูลในแฟ้มข้อมูลแบบตามลำดับ โดยใช้แฟ้ม Phone.Dat เช่นเดียวกับตัวอย่างที่ 3 โปรแกรมนี้ให้ผู้ใช้ระบุชื่อที่ต้องการ เพื่อค้นหาเลขโทรศัพท์มาแสดงบนจอภาพ

```

Program Phone_search;
Uses  Crt;
Type  Phone_rec  = Record
                                Name : String [15];
                                Phone: String [15]
                            End;
    Str15      : String [15];
    Dataitem   : Phone_rec;
    Datafi     : File of Dataitem;
Var  t : Dataitem;
    fi  : File of Dataitem;
    key : Str15;
    ch  : Char;
    found : Boolean;
Begin
Clrscr;
Assign(fi,'PHONE.DAT');
Writeln('*** SEARCH PHONE FILE ***');Writeln;
Repeat
    Write('NAME  :');Readln(key);
    Reset(fi);
    found := False;
    While (Not Found) AND (Not EOF(fi)) Do
        Begin Read (fi,t);
            If t.name = key Then  found := True;
        End;
    Close(fi);
    If Found Then writeln('PHONE :',t.phone)
    Else Writeln('PHONE :** Not Found **');
    Writeln; Write('Search Any More?(Y/N):');
    ch := Readkey; Writeln;
Until  ch IN ['n', 'N'];
Writeln('*** End of Search ***');
End.

```

ตัวอย่างการทำงานของโปรแกรม Phone_search

*** SEARCH PHONE FILE ***

NAME :PIRIYA {ผู้ใช้ระบุชื่อ}

PHONE :232-4451 {แสดงหมายเลขเมื่อค้นพบข้อมูล}

Search Any More?(Y/N)Y {ผู้ใช้ต้องการค้นหาต่อ}

NAME :ANN {ผู้ใช้ระบุชื่อ}

PHONE :** Not Found ** {ค้นไม่พบข้อมูลในแฟ้ม}

Search Any More?(Y/N)N {ผู้ใช้ไม่ต้องการค้นหาต่อ}

** End of Search ***

แบบฝึกหัด

1. ทำไมจึงจำเป็นต้องทำการเรียงลำดับข้อมูล
2. อธิบายความหมายและที่มาของฟังก์ชันบิก โอ
3. เรียงลำดับข้อมูลต่อไปนี้ 6,9,4,10,13,0,21,11 โดยวิธีเรียงลำดับต่อไปนี้
 - ก. แบบพอง
 - ข. แบบแทรก
 - ค. แบบคัดเลือก
4. ขั้นตอนวิธีของการเรียงลำดับแบบใดที่ทำการสลับค่าข้อมูลน้อยที่สุด
 - ก. แบบพอง ข. แบบแทรก ค. แบบคัดเลือก
5. เรียงลำดับข้อมูล 22,36,6,79,26,45,75,13,31,62,27 ด้วยวิธีเรียงลำดับแบบควิก
6. เขียนโปรแกรมเพื่อปรับปรุงการทำงานของการทำงานของเรียงลำดับแบบพอง โดยทำการตรวจสอบว่าถ้าในรอบใดไม่มีการสลับค่าข้อมูลให้สิ้นสุดการทำงานได้ (แสดงว่าข้อมูลเรียงลำดับแล้ว)
7. เขียนโปรแกรมการค้นแบบทวิภาค โดยใช้หลักการทํางานแบบเรียกซ้ำ (Recursion)
8. ปรับปรุงโปรแกรมตัวอย่างที่ 4 Phone_Search เพื่อใช้การค้นข้อมูลแบบทวิภาค

บทที่ 3

โครงสร้างข้อมูลแบบแถวลำดับ

- หัวข้อ : 3.1 ความหมายและการเรียกใช้
- 3.2 ลักษณะของตัวแปรแถวลำดับ
- 3.3 การเก็บตัวแปรแถวลำดับในหน่วยความจำ
และการฟังก์ชันเพื่อเข้าถึงตัวแปรแถวลำดับ
- 3.4 การประยุกต์ของแถวลำดับ

3.1 ความหมายและการเรียกใช้

โครงสร้างข้อมูลแบบแถวลำดับ (Array) เป็นลักษณะการจัดเก็บข้อมูลจำนวนหนึ่งซึ่งมีลักษณะเหมือนกัน นั่นคือมีประเภทข้อมูลแบบเดียวกัน เก็บไว้เรียงต่อเนื่องกันไปตามลำดับ การอ้างอิงถึงสมาชิกแต่ละตัวจะใช้ ชื่อเดียวกัน และใช้ดัชนีกำกับ (Subscript หรือ index) เพื่อบ่งชี้ว่าเป็นสมาชิกตัวใดในแถวลำดับ

รูปแบบ

$V(i)$

เมื่อ v : ชื่อของตัวแปรแถวลำดับ

i : ดัชนีกำกับ ระบุตำแหน่งสมาชิก

ตัวอย่าง ให้ x เป็นตัวแปรแถวลำดับที่ประกอบด้วยสมาชิกที่เป็นจำนวนเต็ม จำนวน 30 ตัว โดยมีขอบเขตของดัชนีเป็น $1, 2, 3, \dots, 30$ แสดงโครงสร้างได้ดังรูป 3.1

X(1) X(2) X(3) ... X(29) X(30)

--	--	--	--	--	--

X(1) หมายถึง สมาชิกที่อยู่ในลำดับที่ 1
X(29) หมายถึง สมาชิกที่อยู่ในลำดับที่ 29 เป็นต้น

รูป 3.1 ตัวอย่างตัวแปรแถวลำดับ

การใช้ตัวแปรแถวลำดับจะต้องมีการกำหนดเกี่ยวกับ จำนวนสมาชิกทั้งหมด ขอบเขตของดัชนี และประเภทข้อมูลของสมาชิกไว้อย่างชัดเจน และการใช้งานจะต้องสอดคล้องกับข้อกำหนดดังกล่าวที่ได้กำหนดไว้

3.1.1 การดำเนินการกับตัวแปรแถวลำดับ

การดำเนินการกับตัวแปรแถวลำดับนั้น จะไม่มีการใส่ข้อมูล (insert) หรือลบข้อมูล (delete) ที่ทำให้ขนาดของแถวลำดับเปลี่ยนแปลง เนื่องจากโครงสร้างแบบแถวลำดับจะมีขนาดคงที่ตลอดการทำงานของโปรแกรม การดำเนินการกับตัวแปรแถวลำดับ ได้แก่

- การเรียงลำดับข้อมูล
- การค้นข้อมูล
- การประมวลผลต่าง ๆ กับข้อมูลในแถวลำดับ เช่นการใส่ข้อมูล การนำข้อมูลไปใช้ในการคำนวณต่าง ๆ เป็นต้น

3.1.2 มิติ

มิติ (Dimension) ของตัวแปรแถวพิจารณาจากทิศทางของการจัดวางสมาชิกทั้งหมด ดังนั้นจึงมีตั้งแต่

- 1 มิติ (One dimension array)
- 2 มิติ (Two dimension array)
- 3 มิติ (Three dimension array)
- (และมากกว่า 3 มิติ)

แต่โดยทั่วไปมักนิยมใช้ไม่เกิน 3 มิติ เนื่องจากรูปแบบจะเข้าใจยาก รูป 3.2 แสดงตัวอย่างแถวลำดับแบบ 1 มิติ 2 มิติ และ 3 มิติ

$x(1) \ x(2) \ x(3) \ \dots \ x(n)$

--	--	--	--	--

(ก) แถวลำดับแบบ 1 มิติ

สดมภ์ 1 2 3

แถว

1	$x(1,1)$	$x(1,2)$	$x(1,3)$
2	$x(2,1)$	$x(2,2)$	$x(2,3)$
3	$x(3,1)$	$x(3,2)$	$x(3,3)$

(ข) แถวลำดับแบบ 2 มิติ

รหยาบ 1

รหยาบ 2

รหยาบ 3

$A(1,1,1)$	$A(1,1,2)$	$A(1,1,3)$	$A(2,1,1)$	$A(2,1,2)$	$A(2,1,3)$	$A(3,1,1)$	$A(3,1,2)$	$A(3,1,3)$
$A(1,2,1)$	$A(1,2,2)$	$A(1,2,3)$	$A(2,2,1)$	$A(2,2,2)$	$A(2,2,3)$	$A(3,2,1)$	$A(3,2,2)$	$A(3,2,3)$
$A(1,3,1)$	$A(1,3,2)$	$A(1,3,3)$	$A(2,3,1)$	$A(2,3,2)$	$A(2,3,3)$	$A(3,3,1)$	$A(3,3,2)$	$A(3,3,3)$

(ค) แถวลำดับแบบ 3 มิติ

รูป 3.2 แถวลำดับแบบ 1 มิติ 2 มิติ และ 3 มิติ

3.2 ลักษณะของตัวแปรแถวลำดับ

3.2.1 ตัวแปรแถวลำดับแบบหนึ่งมิติ

การจัดวางตำแหน่งของสมาชิกในตัวแปรแถวลำดับแบบหนึ่งมิติ จะไปในทิศทางเดียวต่อเนื่องกันไปตามลำดับ และใช้ดัชนีกำกับเพียงตัวเดียวในการอ้างอิงถึงสมาชิก ในลักษณะ

ชื่อตัวแปร (ดัชนี)

หรือ $V(i)$

ค่าของดัชนีอาจกำหนดในลักษณะ $1, 2, 3, \dots, n$ เมื่อ n เป็นจำนวนสมาชิกสูงสุด

ตัวอย่าง ให้ CLASS เป็นตัวแปรแถวลำดับซึ่งเก็บจำนวนนักเรียนในแต่ละห้องเรียน ซึ่งมีทั้งหมด 10 ห้อง การเขียนโปรแกรมคอมพิวเตอร์จะต้องมีส่วนของการประกาศเกี่ยวกับ ชื่อตัวแปร ขนาด และประเภทข้อมูลของสมาชิก ในภาษาปาสคาลจะกำหนดได้ดังนี้

VAR CLASS : Array [1..10] of Integer;

(หรือในภาษาเบสิกจะใช้คำสั่ง DIM ย่อมาจาก Dimension สำหรับประกาศตัวแปรแถวลำดับได้ดังนี้

DIM CLASS%(10) เป็นต้น)

การอ้างอิงถึงสมาชิกตัวที่ 5 เขียนได้เป็น CLASS[5] เป็นต้น ทั้งนี้ต้องระวังว่าสมาชิกในแถวลำดับต้องมีประเภทข้อมูลแบบเดียวกัน และจะอ้างอิงถึงตัวแปรเกินขอบเขตของดัชนีที่กำหนดไว้ไม่ได้ ในบางภาษาอาจยอมให้กำหนดขอบเขตของดัชนีได้โดยไม่ต้องเริ่มจาก 1, 2, 3, ... เช่นในภาษาปาสคาล เราอาจกำหนดค่าดัชนีเป็น -4, -3, -2, -1, 0, 1, 2, 3, 4, 5 เป็นต้น โดยเขียนในคำสั่งประกาศได้ดังนี้ สมมติให้ X เป็นชื่อตัวแปรแถวลำดับ

VAR X: Array [-4..5] of Real;

ซึ่งจะมีสมาชิกทั้งหมด 10 ตัว ($5 - (-4) + 1 = 10$)

นั่นคือสามารถระบุขอบเขตของดัชนีกำกับได้ในลักษณะ

[ค่าเริ่มต้นของดัชนี (1) .. ค่าสุดท้ายของดัชนี (u)]

และจำนวนสมาชิกทั้งหมดในแถวลำดับคำนวณได้จาก $(u - 1 + 1)$

3.2.2 ตัวแปรแถวลำดับแบบสองมิติ

การจัดวางตำแหน่งของสมาชิกในตัวแปรแถวลำดับแบบสองมิติ จะอยู่ในลักษณะของตารางสองทางนั่นเอง การอ้างอิงถึงสมาชิกต้องระบุถึงตำแหน่งของแถวและสดมภ์ ในรูปแบบ

ชื่อตัวแปร (ดัชนีแถว, ดัชนีสดมภ์)

หรือ $V(i, j)$

ตัวอย่างเช่น ให้ A เป็นตัวแปรแถวลำดับแบบ 2 มิติ ซึ่งประกอบด้วยสมาชิกที่เป็นจำนวนเต็ม มีดัชนีแถวเป็น 1, 2, 3, 4 และดัชนีสดมภ์เป็น 0, 1 เขียนประกาศได้ดังนี้

VAR A: Array [1..4, 0..1] of integer;

นั่นคือ การระบุขอบเขตของดัชนี เป็นดังนี้

$[l_1..u_1, l_2..u_2]$

เมื่อ l_1 : ค่าเริ่มต้นของดัชนีแถว

u_1 : ค่าสุดท้ายของดัชนีแถว

l_2 : ค่าเริ่มต้นของดัชนีสดมภ์

u_2 : ค่าสุดท้ายของดัชนีสดมภ์

จำนวนสมาชิกทั้งหมด คำนวณจาก (จำนวนแถว * จำนวนสดมภ์) หรือเท่ากับ

$$(u_1 - l_1 + 1)(u_2 - l_2 + 1)$$

3.2.3 ตัวแปรแถวลำดับแบบสามมิติ

ลักษณะการจัดวางตำแหน่งของสมาชิกในตัวแปรแถวลำดับแบบสามมิติ จะเพิ่มระนาบ (plane) ขึ้นมาจากแบบสองมิติ นั่นคือสามารถมองได้เป็นตารางแบบสองมิติ ซ้อนกันตั้งแต่สองตารางขึ้นไป

การอ้างอิงถึงสมาชิกต้องระบุดัชนีไว้กับ 3 ตัว ดังนี้

ชื่อตัวแปร (ดัชนีระนาบ, ดัชนีแถว, ดัชนีสดมภ์)

หรือ $V(i, j, k)$

จากตัวอย่างของตัวแปรแถวลำดับแบบสองมิติ จะกำหนดให้ B เป็นตัวแปรแถวลำดับแบบ 3 มิติ ซึ่งประกอบด้วย 3 ระนาบ คือ -1,0,1 เขียนประกาศด้วยปาสคาลได้ดังนี้

```
VAR B:Array [-1..1,1..4,0..1] of integer;
```

นั่นคือ การระบุขอบเขตของดัชนี จะเป็นดังนี้

$[1..u_1, 1..u_2, 1..u_3]$

เมื่อ $1..u_1$ เป็นขอบเขตของดัชนีระนาบ

$1..u_2$ เป็นขอบเขตของดัชนีแถว

$1..u_3$ เป็นขอบเขตของดัชนีสดมภ์

จำนวนสมาชิกทั้งหมด คำนวณจาก (จำนวนระนาบ * จำนวนแถว * จำนวนสดมภ์)

หรือเท่ากับ $(u_1 - l_1 + 1)(u_2 - l_2 + 1)(u_3 - l_3 + 1)$

จากตัวอย่างข้างต้นจำนวนสมาชิกทั้งหมดของแถวลำดับ B = $(1 - (-1) + 1) * (4 - 1 + 1) * (1 - 0 + 1)$
 $= 3 * 4 * 2 = 24$

3.3 การเก็บตัวแปรแถวลำดับในหน่วยความจำ และการคำนวณฟังก์ชันเพื่อเข้าถึง

ในที่นี้จะใช้สัญลักษณ์ A[ขอบเขตดัชนี] แทนการกำหนดตัวแปรแถวลำดับ เช่น A[1..10] แทนแถวลำดับแบบหนึ่งมิติขนาดสมาชิก 10 ตัว ค่าของดัชนีกำกับเป็น 1,2,3,...,10 A[1..5,0..99] แทนแถวลำดับแบบสองมิติขนาดสมาชิก 500 ตัว ค่าของดัชนีแถวเป็น 1,2,...,5 ดัชนีสดมภ์เป็น 0,1,2,3,...,99 เป็นต้น

3.3.1 การเก็บตัวแปรแถวลำดับแบบหนึ่งมิติ

การเก็บตัวแปรแถวลำดับแบบหนึ่งมิติ จะเก็บเรียงในเนื้อที่ที่ต่อเนื่องกันไปตามลำดับในหน่วยความจำ เช่น

A(1) A(2) A(3) A(4) A(5)

A[1..5] จะจัดเก็บแบบ

--	--	--	--	--

B(-1) B(0) B(1) B(2)

B[-1..2] จะจัดเก็บแบบ

--	--	--	--

3.3.2 ฟังก์ชันเพื่อเข้าถึงตัวแปรแถวลำดับแบบหนึ่งมิติ

ฟังก์ชันเพื่อเข้าถึง (Accessing function) เป็นฟังก์ชันเพื่อคำนวณหาเลขที่อยู่ของหน่วยความจำที่เก็บข้อมูลที่ต้องการ การเก็บแถวลำดับในหน่วยความจำจะใช้เนื้อที่ต่อเนื่องกันไปในหน่วยความจำ และสมาชิกแต่ละตัวจะใช้เนื้อที่เท่ากัน

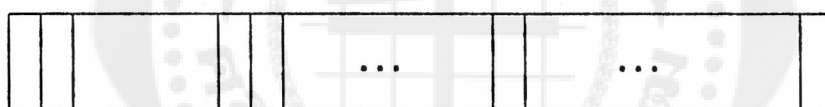
ถ้า $A[1..n]$ เป็นตัวแปรแถวซึ่งมีสมาชิก n ตัว และสมาชิกแต่ละตัวใช้เนื้อที่ s ไบต์ แล้วเนื้อที่ทั้งหมดสำหรับแถวลำดับนี้เท่ากับ ns ไบต์

ถ้าหน่วยความจำหลักมีขนาด M ไบต์ ดังนั้นเลขที่อยู่จะเป็น $0, 1, 2, \dots, (M-1)$ แล้วตัวแปรแถวลำดับ $A[1..n]$ จะอยู่ในเนื้อที่ระหว่างตำแหน่ง 0 ถึง $(M-1)$ เลขที่อยู่เริ่มต้นของแถวลำดับในหน่วยความจำ เรียกว่า เลขที่อยู่ฐาน (Base Address) ใช้สัญลักษณ์แทนว่า L_0

ดังนั้น $L_0 = \text{Loc}(A(1))$ หรือ เลขที่อยู่ของ $A(1)$

การเก็บแถวลำดับ $A[1..n]$ ในหน่วยความจำ แสดงได้ดังรูป 3.3

เลขที่อยู่ 0 1 2 ... L_0 ... (M-1)



$A(1) \ A(2) \ \dots \ A(n)$

← แถวลำดับ A →

รูป 3.3 การเก็บแถวลำดับในหน่วยความจำ

การคำนวณเลขที่อยู่ของ $A(i)$ ใน $A[1..n]$ พิจารณาจากเลขที่อยู่ของ

$$A(1) = L_0$$

$$A(2) = L_0 + s$$

$$A(3) = L_0 + 2s$$

.....

ดังนั้น

$$\text{Loc } A(i) = L_0 + (i-1)s$$

เมื่อ $i = 1, 2, \dots, n$

กรณีทั่วไป การคำนวณของ $A(i)$ ใน $A[1..u]$ เป็นดังนี้

$$\begin{aligned} A(1) &= L_0 \\ A(1+1) &= L_0 + s \\ A(1+2) &= L_0 + 2s \\ &\dots\dots\dots \\ A(u) &= L_0 + (u-1)s \end{aligned}$$

ดังนั้น $Loc\ A(i) = L_0 + (i-1)s$

เมื่อ $1 \leq i \leq u$

ตัวอย่าง หาเลขที่อยู่ของ $A(3)$ ใน $A[-1..4]$ โดยเลขที่อยู่ของ $A(-1)$ คือ 1000 สมาชิกแต่ละตัวใช้เนื้อที่ 4 ไบต์

$$\begin{aligned} \text{เลขที่อยู่ของ } A(i) &= L_0 + (i-1)s \\ \text{เลขที่อยู่ของ } A(3) &= 1000 + (3 - (-1)) * 4 \\ &= 1000 + 16 \\ &= 1016 \end{aligned}$$

แสดงโดยรูปได้ดังนี้

$A(-1)\ A(0)\ A(1)\ A(2)\ A(3)\ A(4)$

--	--	--	--	--	--

1000 1004 1008 1012 1016 1020

3.3.3 การเก็บตัวแปรแถวลำดับแบบสองมิติ

การเก็บตัวแปรแถว แบบ 2 มิติ จะถูกเก็บเรียงต่อไปในหน่วยความจำโดยใช้หลัก

การ 2 แบบ คือ

- 1. ลำดับแถวเป็นหลัก (Row Major Order)
- 2. ลำดับสดมภ์เป็นหลัก (Column Major Order)

การเก็บแบบลำดับแถวเป็นหลัก

ใช้หลักการเก็บแต่ละแถวเรียงลำดับกันไป ดังรูป

แถว 1 แถว 2 ... แถว N

			...		
--	--	--	-----	--	--

Lo

เช่น $A[1..n,1..3]$ การเก็บข้อมูลเรียงลำดับดังนี้

$A(1,1), A(1,2), A(1,3), A(2,1), A(2,2), A(2,3), \dots, A(n,1), A(n,2), A(n,3)$

|<— แถวที่ 1 —>| |<— แถวที่ 2 —>| ... |<— แถวที่ n —>|

สังเกต : ดัชนีสดมภ์จะเปลี่ยนก่อนดัชนีแถว

การเก็บแถวลำดับแบบสดมภ์เป็นหลัก

ใช้หลักการเก็บแต่ละสดมภ์เรียงลำดับกันไป ดังรูป 3.4

สดมภ์ 1 สดมภ์ 2 ... สดมภ์ K

			...		
--	--	--	-----	--	--

Lo

รูป 3.4 การเก็บแถวลำดับแบบสดมภ์เป็นหลัก

เช่น $A[1..3,1..K]$ จะเก็บเรียงลำดับดังนี้

$A(1,1), A(2,1), A(3,1), A(1,2), A(2,2), A(3,2), \dots, A(1,K), A(2,K), A(3,K)$

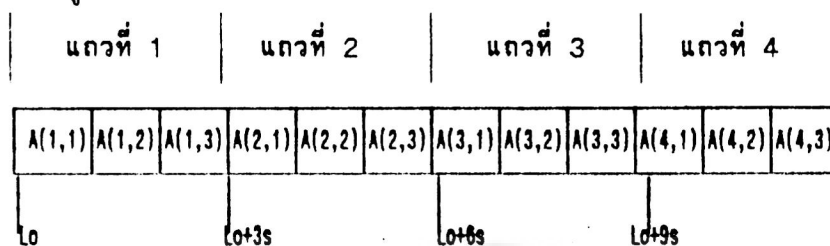
|<— สดมภ์ 1 —>| |<— สดมภ์ 2 —>| ... |<— สดมภ์ K —>|

3.3.4 ฟังก์ชันเพื่อเข้าถึงตัวแปรแถวลำดับสองมิติ

1. เก็บแบบลำดับแถวเป็นหลัก

ให้สมาชิกแต่ละตัวใช้เนื้อที่ s ไบต์ ตัวแปรแถวลำดับ $A[1..4, 1..3]$ ถูกเก็บ

ในลักษณะดังรูป 3.5



รูป 3.5 ตัวอย่างการเก็บแถวลำดับสองมิติแบบลำดับแถวเป็นหลัก

พิจารณาเลขที่อยู่ $A(1,1) = Lo$ $\longrightarrow Lo + 0*(3s)$

$A(2,1) = Lo + 3s$ $\longrightarrow Lo + 1*(3s)$

$A(3,1) = Lo + 6s$ $\longrightarrow Lo + 2*(3s)$

$A(4,1) = Lo + 9s$ $\longrightarrow Lo + 3*(3s)$

ดังนั้นในกรณีของ $A[1..u_1, 1..u_2]$ เลขที่อยู่ของ

$$A(i,1) = Lo + (i-1) u_2 s$$

ดังนั้นเลขที่อยู่ของ $A(i,j)$ คำนวณจาก

$$Loc(A(i,j)) = Lo + (i-1) u_2 s + (j-1) s$$

เมื่อ $1 \leq i \leq u_1$ และ $1 \leq j \leq u_2$

และกรณีทั่วไป $A[l_1..u_1, l_2..u_2]$

จำนวนแถว $= (u_1 - l_1 + 1)$

จำนวนสดมภ์ $= (u_2 - l_2 + 1)$

เลขที่อยู่ของ $A(l_1, l_2) = Lo$

เลขที่อยู่ของ $A(i,j)$ คำนวณจาก

$$\text{Loc}(A(i,j)) = L_0 + (i-1)(u_2-1)s + (j-1)s$$

เมื่อ $1 \leq i \leq u_1$ และ $1 \leq j \leq u_2$

2. แบบลำดับสดมภ์เป็นหลัก

สมมติ $A[1..4,1..3]$ ถูกจัดเก็บแบบลำดับสดมภ์เป็นหลัก จะอยู่ในลักษณะ

สดมภ์ 1				สดมภ์ 2				สดมภ์ 3			
$A(1,1)$	$A(2,1)$	$A(3,1)$	$A(4,1)$	$A(1,2)$	$A(2,2)$	$A(3,2)$	$A(4,2)$	$A(1,3)$	$A(2,3)$	$A(3,3)$	$A(4,3)$
L_0				$L_0 + 4s$ $= L_0 + 1 \cdot (4s)$				$L_0 + 8s$ $= L_0 + 2 \cdot (4s)$			

รูป 3.6 ตัวอย่างการเก็บแถวลำดับแบบสองมิติแบบลำดับสดมภ์เป็นหลัก

ดังนั้น สำหรับ $A[1..u_1, 1..u_2]$ เลขที่อยู่ของ $A(i,j)$ คำนวณจาก

$$\text{Loc}(A(1,j)) = L_0 + (j-1)u_1s$$

และ

$$\text{Loc}(A(i,j)) = L_0 + (j-1)u_1s + (i-1)s$$

เมื่อ $1 \leq i \leq u_1$ และ $1 \leq j \leq u_2$

ตัวอย่าง หาเลขที่อยู่ของ $A(4,3)$ ของตัวแปรแถวลำดับ $A[1..5,1..4]$ ซึ่งจัดเก็บแบบลำดับ
สดมภ์เป็นหลัก

$$\begin{aligned}\text{Loc } (A(4,3)) &= L_0 + (j-1)u_1s + (i-1)s \\ &= L_0 + (3-1)5s + (4-1)s \\ &= L_0 + 10s + 3s \\ &= L_0 + 13s\end{aligned}$$

กรณีทั่วไป ตัวแปรแถวลำดับ $A[l_1..u_1, l_2..u_2]$ สำหรับการจัดเก็บแบบลำดับสดมภ์เป็นหลัก

$$\text{จำนวนแถว} = (u_1 - l_1 + 1)$$

$$\text{จำนวนสดมภ์} = (u_2 - l_2 + 1)$$

$$\text{ให้ } \text{Loc } (A(l_1, l_2)) = L_0$$

เลขที่อยู่ของสมาชิกตัวแรกในสดมภ์ที่ j คือ

$$\text{Loc } (A(l_1, j)) = L_0 + (j-l_2)(u_1-l_1+1)s$$

ดังนั้นเลขที่อยู่ของ $A(i, j)$ คำนวณจาก

$$\text{Loc } A(i, j) = L_0 + (j-l_2)(u_1-l_1+1)s + (i-l_1)s$$

$$\text{เมื่อ } l_1 \leq i \leq u_1 \text{ และ } l_2 \leq j \leq u_2$$

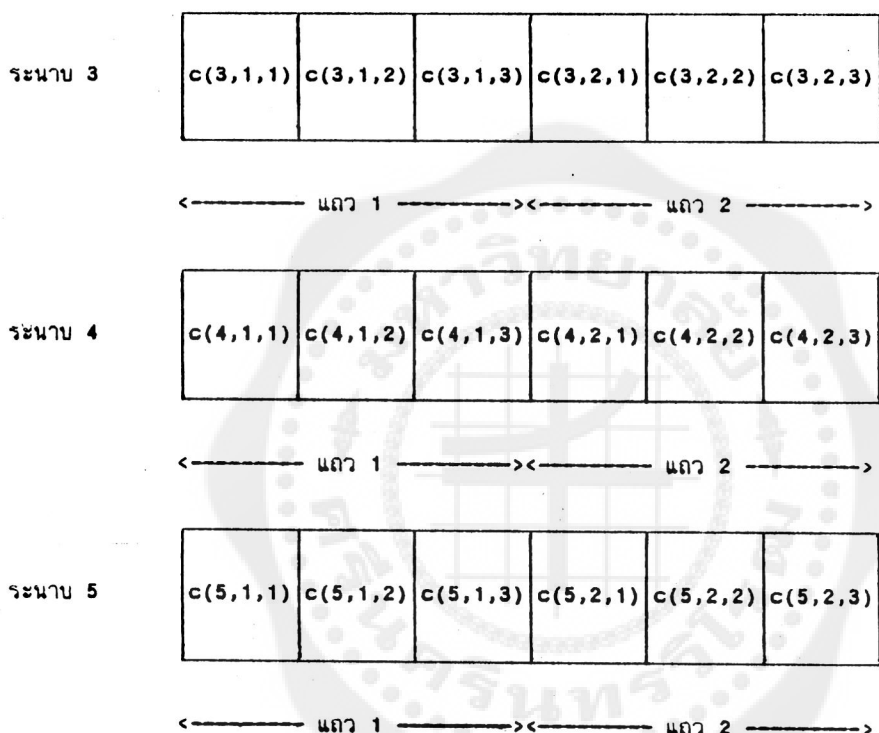
ตัวอย่าง หาเลขที่อยู่ของ $A(-1,0)$ ของตัวแปรแถวลำดับ $A[-2..2,-1..-1]$ ซึ่งเก็บแบบ
ลำดับสดมภ์เป็นหลัก

$$\text{Loc } (A(i, j)) = L_0 + (j-l_2)(u_1-l_1+1)s + (i-l_1)s$$

$$\begin{aligned}\text{Loc } (A(-1,0)) &= L_0 + (0-(-1))(2-(-2)+1)s + (-1-(-2))s \\ &= L_0 + 5s + s \\ &= L_0 + 6s\end{aligned}$$

3.3.5 การเก็บตัวแปรแถวลำดับแบบสามมิติ

การเก็บตัวแปรแถวลำดับแบบสามมิติ มีหลักการเช่นเดียวกับการเก็บตัวแปรแถวลำดับแบบสองมิติ กล่าวคือในแต่ละระนาบอาจใช้การเก็บแบบลำดับแถวเป็นหลัก หรือเก็บแบบลำดับแถวเป็นหลัก และเก็บเรียงลำดับของระนาบต่อเนื่องกันไป ตัวอย่างเช่น ตัวแปรแถวลำดับ $A[3..5, 1..2, 1..3]$ จัดเก็บแบบลำดับแถวเป็นหลัก การจัดเก็บเป็นดังรูป 3.7



รูป 3.7 ตัวอย่างการเก็บแถวลำดับสามมิติแบบลำดับแถวเป็นหลัก

ดังนั้น ตัวแปรแถวลำดับ $A[l_1..u_1, l_2..u_2, l_3..u_3]$ เมื่อใช้การจัดเก็บแบบลำดับแถวเป็นหลัก และ $\text{Loc}(A[l_1, l_2, l_3]) = L_0$ แล้วเลขที่อยู่ของ $A(i, j, k)$ สามารถคำนวณได้จากสูตร

$$\text{Loc}(A[l_1, l_2, l_3]) = L_0 + (i-l_1)(u_2 - l_2 + 1)(u_3 - l_3 + 1)s + (j-l_2)(u_3 - l_3 + 1)s + (k-l_3)s$$

เมื่อ $l_1 \leq i \leq u_1$, $l_2 \leq j \leq u_2$ และ $l_3 \leq k \leq u_3$

หมายเหตุ ให้เขียนรูปแสดงการจัดเก็บแถวลำดับข้างต้นแบบลำดับสดมภ์เป็นหลัก และเขียนสูตร
คำนวณเลขที่อยู่ของ $A(i,j,k)$ ด้วยตนเอง

3.4 การประยุกต์ของแถวลำดับ

จากลักษณะของการจัดเก็บแถวลำดับในหน่วยความจำดังกล่าวมาแล้ว โครงสร้าง
ข้อมูลแบบแถวลำดับสามารถนำไปประยุกต์ใช้ได้หลากหลายลักษณะ ในที่นี้จะกล่าวถึงการประยุกต์
สองแบบ คือ เมตริกซ์สามเหลี่ยม และ เมตริกซ์แบบสเปซ

3.4.1 เมตริกซ์สามเหลี่ยม

เมตริกซ์สามเหลี่ยม (Tridiagonal Matrix) เป็นเมตริกซ์ขนาด $N \times N$
มีลักษณะดังรูป 3.8

$$\begin{vmatrix} X & X & X & X \\ 0 & X & X & X \\ 0 & 0 & X & X \\ 0 & 0 & 0 & X \end{vmatrix}$$

Upper Triangular

หรือ

$$\begin{vmatrix} X & 0 & 0 & 0 \\ X & X & 0 & 0 \\ X & X & X & 0 \\ X & X & X & X \end{vmatrix}$$

Lower Triangular

รูป 3.8 เมตริกซ์สามเหลี่ยม

จะเห็นว่าจำนวนสมาชิกในเมตริกซ์สามเหลี่ยมที่ไม่เป็นศูนย์ มีอยู่ทั้งสิ้นไม่เกิน
 $1+2+3+...+N = [N(N+1)]/2$ ดังนั้นเราสามารถปรับปรุงรูปแบบการจัดเก็บเมตริกซ์ดังกล่าว
โดยการใช้แถวลำดับแบบหนึ่งมิติที่เก็บเฉพาะสมาชิกที่ไม่เป็นศูนย์ แทนการเก็บด้วยแถวลำดับแบบ
สองมิติ ซึ่งจะช่วยให้ประหยัดเนื้อที่ที่ใช้เก็บข้อมูล พิจารณาเมตริกซ์สามเหลี่ยม A

$$A = \begin{vmatrix} A_{11} & 0 & 0 & \dots & 0 \\ A_{21} & A_{22} & 0 & \dots & 0 \\ A_{31} & A_{32} & A_{33} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & A_{n3} & \dots & A_{nn} \end{vmatrix}$$

เมื่อเก็บในแถวลำดับแบบหนึ่งมิติ จะเป็นดังรูป 3.9

1 2 3 4 5 6 ...

[N(N+1)]/2

A ₁₁	A ₂₁	A ₂₂	A ₃₁	A ₃₂	A ₃₃	...	A _{n1}	A _{n2}	A _{n3}	...	A _{nn}
-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----	-----------------	-----------------	-----------------	-----	-----------------

รูป 3.9 การเก็บเมตริกซ์สามเหลี่ยมในแถวลำดับหนึ่งมิติ

เมื่อเปลี่ยนรูปแบบการจัดเก็บมาใช้แถวลำดับแบบหนึ่งมิติแล้ว การคำนวณตำแหน่งของ A_{ij} ในแถวลำดับแบบหนึ่งมิติ คำนวณได้จากสูตร

$k = [(i-1)*i]/2 + j$

เช่น A₃₂ เก็บอยู่ในตำแหน่งที่ [(3-1)*3]/2+2 =5 ของแถวลำดับแบบหนึ่งมิติ เป็นต้น

3.4.2 เมตริกซ์แบบสพาซ

เมตริกซ์แบบสพาซ (Sparse Matrix) หมายถึงเมตริกซ์ที่สมาชิกส่วนมากมีค่าเป็นศูนย์ ดังตัวอย่างในรูป 3.10

A =

0	0	0	1	0	2	0
0	0	0	0	0	0	0
1	0	0	0	2	0	0
0	0	0	0	0	0	0
0	0	4	0	0	0	0
2	0	0	0	0	0	0

รูป 3.10 ตัวอย่างเมตริกซ์แบบสพาซ

ถ้าเก็บเมตริกซ์นี้ด้วยแถวลำดับขนาด N×N จะสิ้นเปลืองเนื้อที่โดยไม่จำเป็น เนื่องจากมีสมาชิกที่ไม่เป็นศูนย์เพียง 6 ตัว จากทั้งหมด 42 ตัว ได้แก่

A[1,4] = 1, A[1,6] = 2, A[3,1] = 1, A[3,5] = 2, A[5,3] = 4, A[6,1] = 2

การจัดเก็บเมตริกซ์ A อาจใช้ตัวแปรแถวลำดับที่ช่วยแสดงตำแหน่ง และค่าของสมาชิกแต่ละตัว ดังรูป 3.11 ซึ่งจะช่วยให้ประหยัดเนื้อที่ในการเก็บข้อมูลลงได้มาก โดยเฉพาะเมื่อเมตริกซ์มีขนาดใหญ่และมีสมาชิกที่เป็นศูนย์อยู่เป็นจำนวนมาก

ลำดับที่	แถว	สดมภ์	ข้อมูล
1	1	4	1
2	1	6	2
3	3	1	1
4	3	5	2
5	5	3	4
6	6	1	2

รูป 3.11 การประยุกต์ใช้แถวลำดับสำหรับเก็บเมตริกซ์แบบสพาซ

แบบฝึกหัด

1. เขียนรูปแสดงการจัดเก็บตัวแปรแถวลำดับ $A[1..5, 1..4]$ ในหน่วยความจำ
 - (ก) แบบลำดับแถวเป็นหลัก
 - (ข) แบบลำดับสดมภ์เป็นหลัก
2. คำนวณเลขที่อยู่ของ $A(3,4)$ ของแถวลำดับในข้อ 1 เมื่อเลขที่อยู่ฐานคือ 2000 และสมาชิกแต่ละตัวใช้เนื้อที่ 2 ไบต์
 - (ก) แบบลำดับแถวเป็นหลัก
 - (ข) แบบลำดับสดมภ์เป็นหลัก
3. กำหนดแถวลำดับ $A[-10..10, 0..50]$ สมาชิกแต่ละตัวใช้เนื้อที่ 4 ไบต์ เลขที่อยู่ฐานคือ L_0
 - (ก) จำนวนสมาชิกทั้งหมดเป็นเท่าไร
 - (ข) เนื้อที่ทั้งหมดที่ใช้เก็บแถวลำดับเป็นเท่าใด
 - (ค) คำนวณเลขที่อยู่ของ $A(1,50)$ เมื่อจัดเก็บแบบลำดับแถวเป็นหลัก
4. จากค่าของเมตริกซ์ ต่อไปนี้

1	2	3	4
5	6	7	8
9	10	11	12

ซึ่งถูกแทนด้วยตัวแปรแถวลำดับ 2 มิติ และจัดเก็บแบบลำดับแถวเป็นหลัก โดยมีเลขที่อยู่ฐานคือ 2000 และข้อมูลแต่ละตัวใช้เนื้อที่ 4 ไบต์

- (ก) แสดงค่าของสมาชิกตัวที่ 7 ในตัวแปรแถวลำดับที่เก็บในหน่วยความจำ
- (ข) แสดงเลขที่อยู่ของสมาชิกที่มีค่า 10
- (ค) แสดงค่าของสมาชิกที่อยู่ในตำแหน่ง 2020
- (ง) เขียนเลขที่อยู่ของสมาชิกแถวลำดับ เมื่อมีการพิมพ์ค่าตามลำดับแบบลำดับแถวเป็นหลัก
- (จ) เขียนเลขที่อยู่ของสมาชิกแถวลำดับ เมื่อมีการพิมพ์ค่าตามลำดับแบบลำดับสดมภ์เป็นหลัก

5. แทนเมตริกซ์ในข้อ 2 ด้วยตัวแปรแถวลำดับ 2 มิติ และจัดเก็บแบบลำดับสดมภ์เป็นหลัก และตอบคำถาม ข้อ (ก) ถึง (จ)

6. เขียนโปรแกรมโดยใช้หลักการเก็บเมตริกซ์สามเหลี่ยมในแถวลำดับแบบหนึ่งมิติ เพื่อแก้ระบบสมการดังรูป

$$\begin{aligned} A_{11}X_1 &= b_1 \\ A_{21}X_1 + A_{22}X_2 &= b_2 \\ A_{31}X_1 + A_{32}X_2 + A_{33}X_3 &= b_3 \\ &\dots\dots\dots \\ A_{n1}X_1 + A_{n2}X_2 + A_{n3}X_3 + \dots + A_{nn}X_n &= b_n \end{aligned}$$

7. ให้ B เป็นแถวลำดับแบบหนึ่งมิติ เก็บสมาชิกบนเส้นทแยงมุมของเมตริกซ์ A ซึ่งเป็น tridiagonal matrix ดังรูป

a ₁₁	a ₁₂	0	0	0
a ₂₁	a ₂₂	a ₂₃	0	0
0	a ₃₂	a ₃₃	a ₃₄	0
0	0	a ₄₃	a ₄₄	a ₄₅
0	0	0	a ₅₄	a ₅₅

โดยเก็บ A(1,1) ใน B(1), A(1,2) ใน B(2), A(2,1) ใน B(3) ไปตามลำดับ
จงแสดงฟังก์ชันคำนวณตำแหน่งที่เก็บค่าของ A(i,j) ในแถวลำดับ B

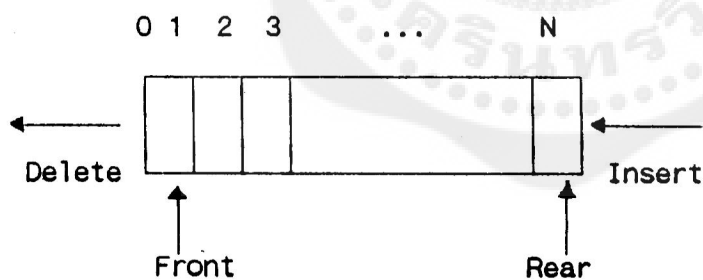
บทที่ 4

โครงสร้างข้อมูลแบบคิว

- หัวข้อ: 4.1 ลักษณะของคิว
- 4.2 การดำเนินงานในคิว
 - 4.3 การกำหนดคิวและขั้นตอนดำเนินงาน
 - 4.4 คิววงกลม
 - 4.5 การประยุกต์ของคิว

4.1 ลักษณะของคิว

คิว (Queue) เป็นโครงสร้างข้อมูลเชิงเส้นประเภทหนึ่ง ที่กำหนดให้การใส่ข้อมูลเข้าในคิวและการนำข้อมูลออกจากคิวกระทำที่ปลายคนละด้าน โดยข้อมูลที่เข้ามาก่อนจะถูกนำออกจากคิวก่อน (First-In, First-out หรือ FIFO) ตัวชี้สมาชิกตัวแรกของคิว เรียกว่า Front และตัวชี้สมาชิกตัวหลังสุดของคิวเรียกว่า Rear ดังรูป 4.1



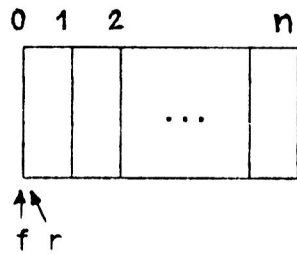
รูป 4.1 โครงสร้างข้อมูลแบบคิว

ถ้ากำหนดค่าสมาชิกในคิวมีได้สูงสุด N ตัว

ค่าของ Front และ Rear จะเป็นได้ $0, 1, 2, \dots, N$

ถ้า $Rear = N$ แสดงว่า คิวเต็ม

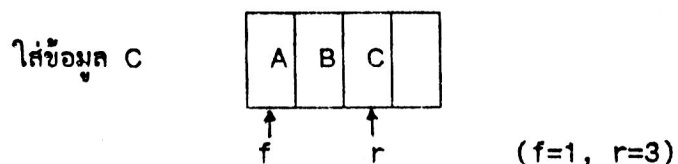
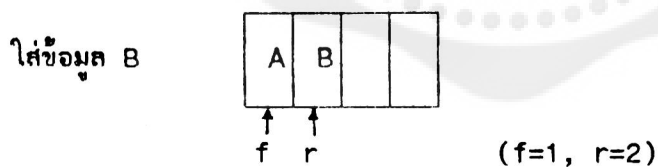
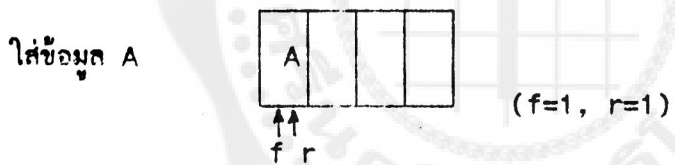
ถ้า $\text{Front}=0$ และ $\text{Rear}=0$ แสดงว่า คิวว่าง (Empty queue) ดังรูป 4.2



รูป 4.2 แสดงคิวว่าง ($f = \text{Front}$, $r = \text{Rear}$)

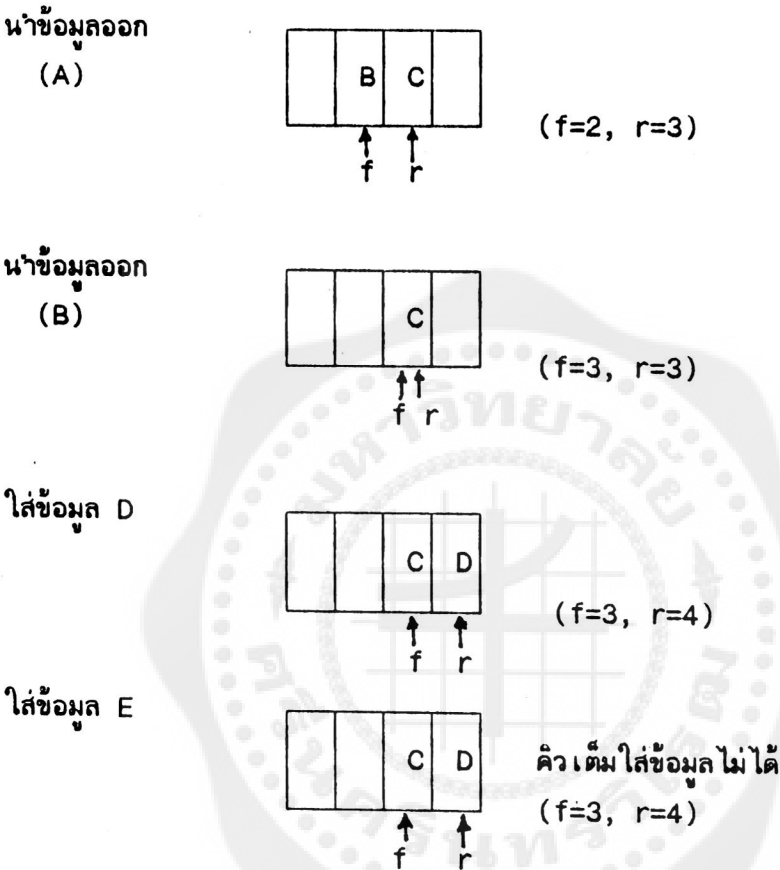
4.2 การดำเนินการในคิว

เมื่อมีการใส่ข้อมูลเข้าในคิวต้องวางต่อท้ายจากตำแหน่งของ Rear โดยปรับค่าของ Rear ให้ขยับไป 1 ตำแหน่งแล้วจึงใส่ข้อมูล ซึ่งจะใส่ข้อมูลเข้าในคิวได้เรื่อยๆจนกว่าคิวจะเต็ม การใส่ข้อมูลเข้าในคิว แสดงในรูป 4.3



รูป 4.3 การใส่ข้อมูลเข้าในคิว

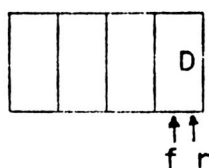
เมื่อต้องการนำข้อมูลออกจากคิวต้องนำออกจากตำแหน่ง Front และต้องตรวจสอบว่า
คิวไม่ว่างจึงจะนำข้อมูลออกได้ และต้องปรับค่า Front ให้ชี้ข้อมูลตัวถัดไปในคิว รูป 4.4 แสดง
การใส่ข้อมูลเข้าและนำข้อมูลออกจากคิว



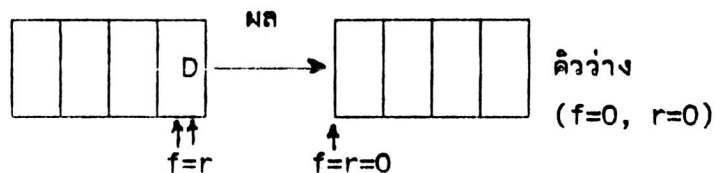
ข้อสังเกต มีที่ว่างส่วนหน้าคิว แต่ใส่ข้อมูลไม่ได้เป็นการเสียเนื้อที่โดยเปล่าประโยชน์

รูป 4.4 การใส่ข้อมูลเข้าและนำข้อมูลออกจากคิว

ลบข้อมูล
(C)



ลบข้อมูล
(D)



รูป 4.4 (ต่อ) การใส่ข้อมูลเข้าและนำข้อมูลออกจากคิว

กรณีของการนำข้อมูลออก ถ้าข้อมูลในคิวมีอยู่เพียงตัวเดียว ($\text{Front} = \text{Rear}$) เมื่อนำข้อมูลออกแล้วจะต้องกำหนดค่า Front และ Rear ให้เป็นศูนย์

4.3 การกำหนดคิวและขั้นตอนวิธีดำเนินการ

กำหนดโครงสร้างแบบคิวด้วยแถวลำดับ ได้ดังรูป 4.5

```

CONST    N    = 100;
TYPE     Itemtype = CHAR;
         Queue  = ARRAY [1..N] OF Itemtype;
VAR      Q : Queue;
         Front, Rear : 0..N;
  
```

รูป 4.5 การกำหนดโครงสร้างแบบคิวด้วยแถวลำดับ

แบบฝึกหัด

- อธิบายสถานการณ์ในการใส่ข้อมูลเข้าและลบข้อมูลออก ในคิวและคิววงกลมเมื่อ
 - Front = 0
 - Front = Rear
 - Rear = N
- อธิบายความแตกต่างระหว่างคิวกับคิววงกลม โดยแสดงให้เห็นถึงลักษณะข้อดี ข้อเสียของคิว ทั้ง 2 แบบ
- เขียนโปรแกรมเพื่อค้นหาว่าในคิววงกลมที่กำหนดให้ มีสมาชิกที่มีค่าเท่ากับค่าที่เก็บในตัวแปร X หรือไม่
 - ถ้า ไม่มี ให้นำค่าของ X เก็บต่อท้ายคิว
 - ถ้า มี ให้แสดงตำแหน่งว่าเป็นตัวที่เท่าใดในคิว

ใช้ข้อมูลต่อไปนี้เพื่อตอบแบบฝึกหัดข้อ 4 และข้อ 5

ระบบคอมพิวเตอร์ระบบหนึ่ง กำหนดลำดับความสำคัญของงานเป็น 10 ระดับ ตามหมายเลขประจำตัวผู้ใช้ ดังนี้

0 - 99	ลำดับความสำคัญสูงสุด
100 - 199	ลำดับความสำคัญสูงสุดรอง
200 - 299	.
.....	.
900 - 999	ลำดับความสำคัญสูงสุดต่ำสุด (ประมวลผลในช่วงวันสุดสัปดาห์)

* งานที่มีลำดับความสำคัญในระดับเดียวกัน จะถูกประมวลผลตามลำดับที่เข้ามาในระบบ

- * งานในคิวที่มีลำดับความสำคัญสูงสุด จะถูกนำมาประมวลผลก่อน ถ้าหมดแล้วก็จะเลือกงานในคิวที่มีลำดับความสำคัญรองลงมา ไปประมวลผลตามลำดับ งานในคิวที่มีลำดับความสำคัญต่ำจะถูกนำไปประมวลผลก็ต่อเมื่อไม่มีงานในคิวที่มีลำดับความสำคัญสูงกว่ารออยู่เท่านั้น

- * ระบบนี้กำหนดคิวของงานตามลำดับความสำคัญไว้ดังนี้

Jobs : ARRAY[0..9] OF QueueType;

ข้อมูลของแต่ละงานที่เก็บในคิวประกอบด้วย เลขประจำตัวผู้ใช้ และ ชื่อโปรแกรม

- เขียนโปรซีเจอร์ AddJob ซึ่งรับค่า เลขประจำตัวผู้ใช้ และ ชื่อโปรแกรม แล้วใส่ข้อมูลดังกล่าวลงในคิวงานตามลำดับความสำคัญตามเลขประจำตัวผู้ใช้
- เขียนโปรซีเจอร์ GetNextJob เพื่อเลือกงานที่จะถูกนำไปประมวลผลจากคิวในลำดับความสำคัญสูงสุดที่มีอยู่ในขณะนั้น โดยให้คำตอบเป็นชื่อของโปรแกรม (และลบข้อมูลของงานนั้นออกจากคิวด้วย)

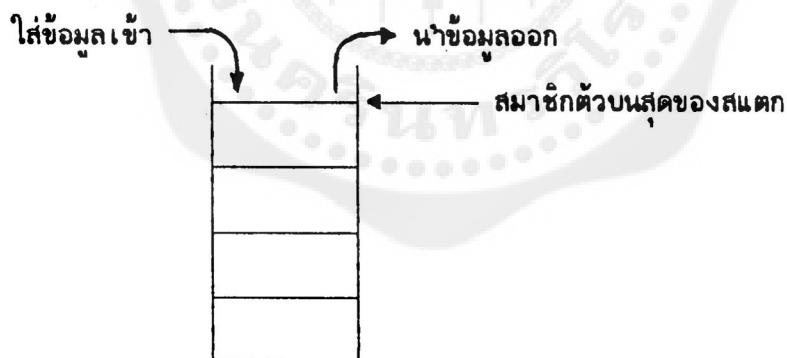
บทที่ 5

โครงสร้างข้อมูลแบบสแตก

- หัวข้อ :
- 5.1 ลักษณะของสแตก
 - 5.2 การดำเนินงานในสแตก
 - 5.3 การกำหนดสแตกและขั้นตอนวิธีดำเนินงาน
 - 5.4 การประยุกต์ใช้สแตก

5.1 ลักษณะของสแตก

สแตกเป็นโครงสร้างข้อมูลเชิงเส้นที่มีข้อกำหนดว่า การเพิ่มข้อมูลเข้าหรือนำข้อมูลออกในโครงสร้างแบบสแตกจะกระทำที่ปลายข้างเดียวกัน โดยสมาชิกที่เข้ามาหลังสุดจะถูกนำออกจากสแตกก่อน (Last-in, First-out หรือ LIFO) ดังแสดงในรูป 5.1



รูป 5.1 โครงสร้างข้อมูลแบบสแตก

โครงสร้างข้อมูลแบบสแตกอาจแทนด้วยตัวแปรแถวลำดับ หรืออาจใช้รายการโยงทั้งนี้สามารถพิจารณาเลือกใช้ได้ตามความเหมาะสมของงาน ในที่นี้จะกล่าวถึงการใช้ตัวแปรแถวลำดับแทนโครงสร้างข้อมูลแบบสแตก

5.2 การดำเนินงานในสแตก

การดำเนินงานในสแตก ได้แก่

- การเพิ่มข้อมูลในสแตก
- การนำข้อมูลออกจากสแตก

การเก็บข้อมูลของสแตก

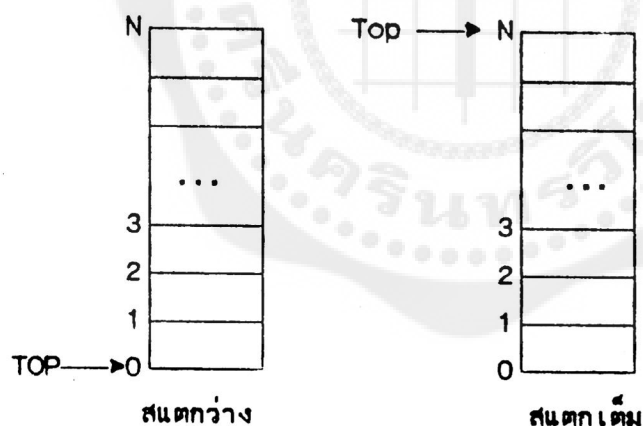
กำหนดให้สแตกสามารถเก็บข้อมูลได้สูงสุด N ตัว

Top เป็นตัวชี้สมาชิกตัวบนสุดของสแตก ค่าของ Top จะเป็นได้ตั้งแต่

$0, 1, 2, \dots, N$

ถ้าสแตกว่าง (stack empty) ค่า Top จะมีค่าเป็น 0

ถ้าสแตกเต็ม (stack full) ค่า Top จะมีค่าเท่ากับ N ดังรูป 5.2



รูป 5.2 แสดงสแตกว่าง และสแตกเต็ม

5.2.1 การใส่ข้อมูลเข้าในสแตก

การใส่ข้อมูลเข้าในสแตก (Pushing Stack) จะเป็นการนำเอาข้อมูลมาใส่ในสแตกโดยวางต่อจากข้อมูลตัวสุดท้ายในสแตก ซึ่งข้อมูลตัวใหม่จะกลายเป็นตัวที่อยู่บนสุดของสแตก

ซึ่งการใส่ข้อมูลจะกระทำได้เรื่อย ๆ จนกว่าข้อมูลจะเต็มสแตก นั่นคือเมื่อ Top มีค่าเท่ากับ N

5.2:2 การนำข้อมูลออกจากสแตก

การนำข้อมูลออกจากสแตก (Popping Stack) เป็นการนำข้อมูลตัวบนสุดของสแตกออกไป ทำให้ตัวถัดลงไปกลายเป็นตัวบนสุดแทน การนำข้อมูลออกจากสแตกจะกระทำได้เรื่อย ๆ จนกว่าสแตกว่าง นั่นคือเมื่อ Top มีค่าเท่ากับ 0

5.3 การกำหนดสแตกและขั้นตอนวิธีดำเนินการ

กำหนดตัวแปรแถวลำดับแทนสแตกด้วยภาษาปาสคาล ได้ดังรูป 5.3

```
CONST  Upperbound = 100;
TYPE   Itemtype = Integer;
       stack      = ARRAY [1..Upperbound] of Itemtype;
VAR    S:Stack;
       top : 0..Upperbound;
```

รูป 5.3 การแทนสแตกด้วยตัวแปรแถวลำดับ

กำหนดฟังก์ชัน empty ในรูป 5.4 สำหรับตรวจสอบว่าสแตกว่างหรือไม่ โดยกำหนดเป็นฟังก์ชันแบบบูล ซึ่งจะให้ค่าเป็นเท็จเมื่อสแตกไม่ว่าง และให้ค่าเป็นจริงเมื่อสแตกว่าง ฟังก์ชันนี้จะใช้สำหรับการตรวจสอบสแตกทุกครั้งที่จะนำข้อมูลออกจากสแตก

การนำข้อมูลออกจากสแตกจะเป็นการนำค่าจากสมาชิกตัวบนสุดของสแตกออกไป เพื่อกำหนดให้เป็นค่าของข้อมูลตัวใดตัวหนึ่งในการทำงาน ดังนั้นจึงกำหนดไว้เป็นฟังก์ชันชื่อ POP ดังรูป 5.5 เมื่อต้องการนำค่าจากสแตกมาให้กับตัวแปร X การเรียกใช้จะเขียนได้ดังนี้ $X := \text{Pop}(s)$

```

FUNCTION  empty(s:stack):boolean;
BEGIN
    IF top <= 0 THEN empty := true
    ELSE  empty := false;
END;

```

รูป 5.4 ขั้นตอนวิธีทดสอบสแตกว่าว่างหรือไม่

```

FUNCTION  pop (VAR s:stack) : itemtype;
BEGIN
    IF empty (s) THEN
        BEGIN  WRITELN('empty stack');
                pop :=0
            END
    ELSE
        BEGIN  pop:= s[top];
                top:= top-1
            END;
    END;
END;

```

รูป 5.5 ขั้นตอนวิธีนำข้อมูลออกจากสแตก

ข้อสังเกต กรณีที่สแตกว่าว่างฟังก์ชัน pop จะเขียนข้อความว่า empty stack และกำหนดให้ pop มีค่าเป็นศูนย์

การใส่ข้อมูลเข้าในสแตก แสดงขั้นตอนวิธีด้วยโปรซีเจอร์ push ดังรูป 5.6

```

PROCEDURE push (VAR s:stack; x:itemtype);
BEGIN
  IF top >= upperbound
    THEN WRITELN ('stack full')
  ELSE
    BEGIN
      top:=top+1;
      s[top]:=X
    END;
  END;
END;

```

รูป 5.6 ขั้นตอนวิธีใส่ข้อมูลเข้าสแตก

การทำงานของโปรซีเจอร์ push จะเป็นการนำค่าที่ส่งมาให้เก็บเข้าเป็นสมาชิกตัวบนสุดของสแตก เช่น ส่งค่า X มาเก็บในสแตก การเรียกใช้จะอยู่ในรูป Push(S,X)

5.4 การประยุกต์ใช้สแตก

สแตกสามารถนำมาประยุกต์ใช้กับงานทางด้านคอมพิวเตอร์ได้หลากหลายกรณี ในที่นี้จะยกตัวอย่างการใช้สแตกในงาน 2 อย่าง คือ

1. การเรียกซ้ำ
2. สัญกรณ์เติมกลาง และสัญกรณ์เติมหลัง

5.4.1 การเรียกซ้ำ

การเรียกซ้ำ (Recursion) เป็นเทคนิคการประมวลผล โดยให้โปรแกรมสามารถเรียกตัวเองมาใช้งานได้ โปรแกรมที่มีสมบัติดังกล่าว เรียกว่า โปรแกรมแบบเรียกซ้ำ (recursive procedure) ซึ่งสามารถใช้ได้ในภาษาคอมพิวเตอร์บางภาษา เช่น ปาสคาล ซี ฟอธรวัน เป็นต้น

งานที่สามารถประมวลผลแบบเรียกซ้ำ จะได้งานที่มีลักษณะประมวลผลซ้ำ ๆ ด้วยวิธีเดียวกัน และใช้ค่าที่ต่อเนื่องกันจนกระทั่งบรรลุถึงค่าสุดท้ายตามกฎเกณฑ์ที่ตั้งไว้ ซึ่งเป็นเงื่อนไขสิ้นสุดการทำงาน โดยทั่วไปอาจเขียนโปรแกรมด้วยวิธีวนซ้ำธรรมดาก็ได้ ตัวอย่างเช่น การคำนวณค่าแฟกทอเรียล (Factorial) ซึ่งคำนวณได้ดังนี้

$$N! = N*(N-1)*(N-2)*...*3*2*1$$

หรือ $N! = N*(N-1)!$

โดยที่ $0!$ มีค่าเท่ากับ 1

พิจารณาฟังก์ชัน $N! = N*(N-1)!$ ส่วนของ $(N-1)!$ เป็นการซ้ำกระบวนการเดิมนั่นเอง ดังนั้น ถ้า $N=5$ จะเห็นว่าสามารถแจกแจงได้ดังรูป 5.7

$$\begin{array}{rcll}
 5! & = & 5*4! & \leftarrow (24) & = 5*24 & = 120 \\
 & \swarrow & & & & \\
 & 4! & = & 4*3! & \leftarrow (6) & = 4*6 & = 24 \\
 & \swarrow & & & & \\
 & & 3! & = & 3*2! & \leftarrow (2) & = 3*2 & = 6 \\
 & & \swarrow & & & & \\
 & & & 2! & = & 2*1! & \leftarrow (1) & = 2 \\
 & & & \swarrow & & & \\
 & & & & 1! & = & 1*0! & \leftarrow (1) & = 1 \\
 & & & & \swarrow & & \\
 & & & & & 0! & = & 1
 \end{array}$$

รูป 5.7 การคำนวณค่าแฟกทอเรียลแบบเรียกซ้ำ

หมายเหตุ ลูกศรลงแสดงการเรียกซ้ำ ลูกศรขึ้นแสดงคืนค่า

คำตอบที่ได้จากการทำงานนี้คือ $5! = 120$ ขั้นตอนวิธีคำนวณค่าแฟกทอเรียลแบบเรียกซ้ำ แสดงในรูป 5.8

```

FUNCTION factorial(n:integer) : integer;
BEGIN
    IF n=0 THEN factorial :=1
    ELSE factorial := n*factorial(n-1);
END;

```

รูป 5.8 ขั้นตอนวิธีคำนวณค่าแฟกทอเรียลแบบเรียกซ้ำ

การเรียกใช้ฟังก์ชัน factorial อาจเขียนดังนี้

```

READLN (n); { n:LongInt}
WRITELN (n,'!=',factorial(n));

```

โปรแกรมย่อยแบบเรียกซ้ำอาจถูกเรียกใช้โดยโปรแกรมอื่น หรือเรียกใช้ตัวเองก็ได้ หลักการทำงานสำหรับการเรียกใช้โปรแกรมย่อยโดยปกตินั้น ก่อนที่จะเริ่มไปทำงานยังโปรแกรมย่อย จะต้องมีการเก็บบันทึกข้อมูลเกี่ยวกับสถานะของโปรแกรมที่เป็นอยู่ขณะนั้นไว้ก่อน เพื่อให้การคืนกลับเมื่อเสร็จจากการทำงานของโปรแกรมย่อยเป็นไปอย่างถูกต้อง คือสามารถกลับมาประมวลผลโปรแกรมที่เรียกใช้ ณ จุดต่อจากคำสั่งที่เรียกใช้นั่นเอง

ข้อมูลเกี่ยวกับสถานะปัจจุบันของโปรแกรมที่เรียกใช้ที่ต้องเก็บบันทึกไว้ ได้แก่ค่าของ

- พารามิเตอร์
- ตัวแปรเฉพาะที่

และ - เลขที่อยู่กลับ

พารามิเตอร์ (formal parameters) ได้แก่ พารามิเตอร์ที่ส่งไปสำหรับใช้ในการประมวลผลในโปรแกรมย่อย

ตัวแปรเฉพาะที่ (local variables) ได้แก่ ตัวแปรต่าง ๆ ที่กำหนดขึ้นใช้เฉพาะในโปรแกรมนั้น

เลขที่อยู่กลับ (return address) หมายถึง เลขที่อยู่ของคำสั่งที่จะถูกกลับมาประมวลผลต่อไป

การประมวลผลโปรแกรมย่อยแบบเรียกซ้ำ ประกอบด้วย ลำดับของการเรียกใช้ และเข้าประมวลผลเป็นลำดับไปจนกว่าจะถึงจุดสิ้นสุดจึงมีการคืนกลับ ดังนั้น เมื่อมีการเรียกใช้โปรแกรมย่อยแต่ละครั้ง ข้อมูลที่เกี่ยวข้องกับสภาวะของโปรแกรมหากกล่าวจะต้องถูกบันทึกเก็บไว้ในลักษณะเรียงต่อเนื่องกันไปตามลำดับของการเรียกใช้ และเมื่อประมวลผลจนจบโปรแกรมย่อยที่ถูกเรียกใช้ครั้งสุดท้าย ที่ทำให้เงื่อนไขสิ้นสุดเป็นจริง ก็จะเริ่มการคืนกลับ (return) และย้อนกลับมาประมวลผลยังจุดต่อจากคำสั่งเรียกใช้ของโปรแกรมที่เรียกใช้ครั้งหลังสุด จนจบแล้วคืนกลับในทางนองเดียวกันเป็นลำดับไป ขณะที่มีการคืนกลับมาทำงานนั้น จะต้องนำค่าข้อมูลเกี่ยวกับสภาวะปัจจุบันของโปรแกรมที่เก็บบันทึกไว้กลับมายังหน่วยความจำ แล้วจึงเริ่มการประมวลผลต่อ

การเก็บบันทึกข้อมูลที่แสดงสภาวะปัจจุบันของโปรแกรม ณ จุดที่เรียกใช้โปรแกรมย่อย เพื่อที่จะสามารถนำกลับมาใช้เมื่อมีการคืนกลับมาทำงานต่อนี้ สามารถนำโครงสร้างข้อมูลแบบสแตกมาใช้ ซึ่งจะช่วยให้การประมวลผลเป็นไปได้อย่างถูกต้อง การประมวลผลโปรแกรมย่อยแบบเรียกซ้ำ ประกอบด้วยขั้นตอน ดังรูป 5.9

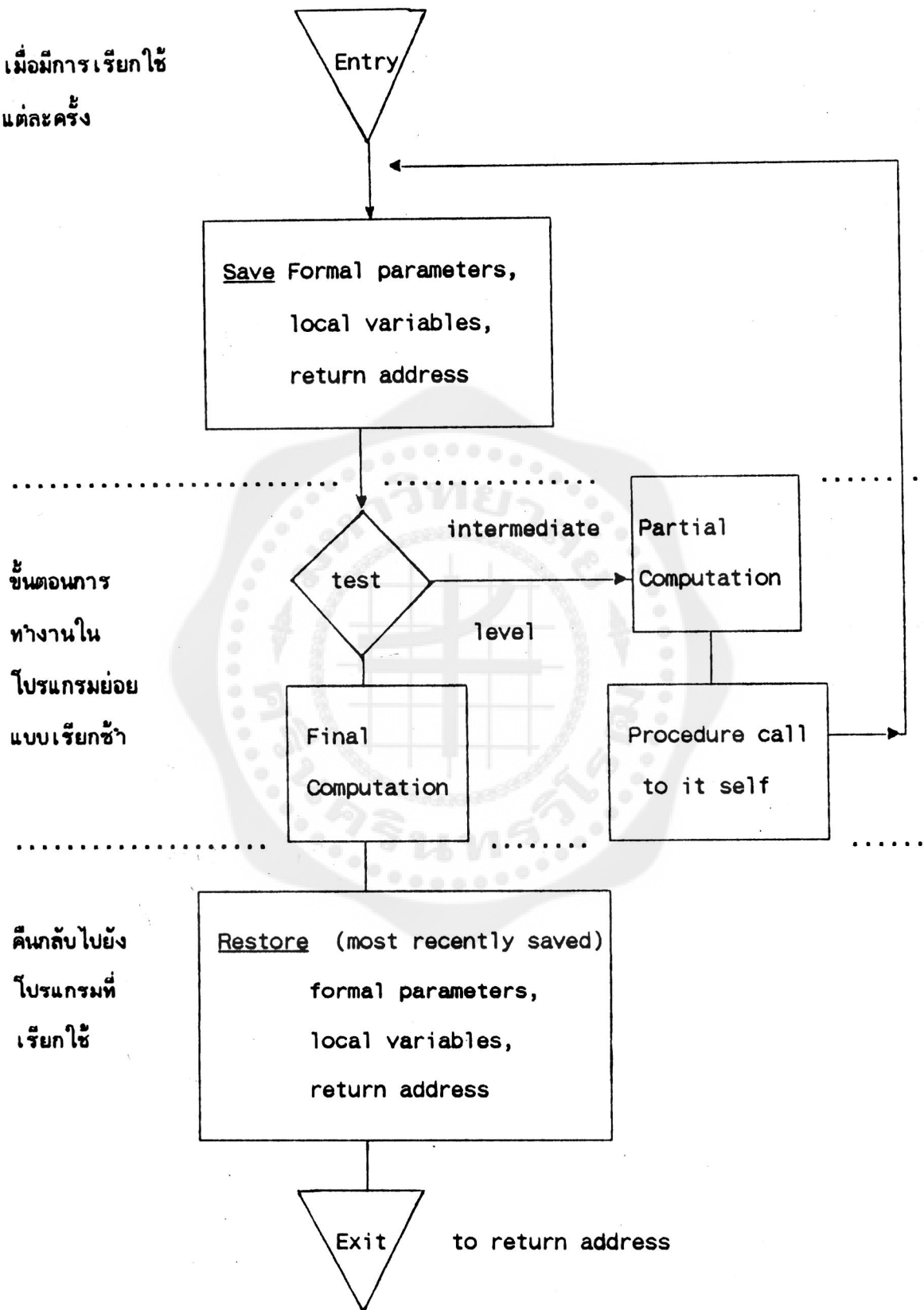
การเขียนโปรแกรมย่อยแบบเรียกซ้ำในภาษาที่อ่านง่าย เช่นภาษาปาสคาล จะเห็นว่าการเขียนโปรแกรมจะง่าย สั้น และเข้าใจง่าย โดยเราไม่เห็นถึงการใช้สแตกสำหรับบันทึกข้อมูลเกี่ยวกับสภาวะของโปรแกรมหากกล่าวแล้ว เนื่องจากเราไม่ต้องเขียนคำสั่งจัดการในส่วนนี้เอง เพราะตัวแปลชุดคำสั่งจะสร้างชุดคำสั่งสำหรับการจัดการสแตกให้มันเอง แต่อย่างไรก็ดีเราสามารถเขียนการทำงานแบบเรียกซ้ำ โดยเขียนส่วนคำสั่งจัดการสแตกเองก็ได้ โดยใช้ขั้นตอนดังที่ได้อธิบายไว้แล้ว

โปรแกรมย่อยแบบเรียกซ้ำ สามารถใช้ได้กับฟังก์ชันการคำนวณ หรือโครงสร้างข้อมูลที่มีนิยามเป็นแบบเรียกซ้ำ เช่น โครงสร้างแบบต้นไม้ ซึ่งจะกล่าวถึงในบทที่ 7 ตัวอย่างฟังก์ชันคำนวณเลขลำดับแบบ Fibonacci ก็สามารถเขียนเป็นโปรแกรมย่อยแบบเรียกซ้ำได้ พิจารณาจากนิยาม

$$\text{FIBONACCI}(N) = \begin{cases} 0 & \text{เมื่อ } N=1 \\ 1 & \text{เมื่อ } N=2 \\ \text{FIBONACCI}(N-1)+\text{FIBONACCI}(N-2) & \text{เมื่อ } N>2 \end{cases}$$

จะเห็นว่านิยามแบบเรียกซ้ำ ให้ทดลองเขียนโปรแกรมคำนวณเลขลำดับดังกล่าว

เมื่อมีการเรียกใช้
แต่ละครั้ง

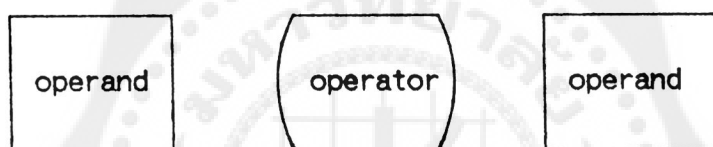


รูป 5.9 ขั้นตอนการประมวลผลโปรแกรมย่อยแบบเรียกซ้ำ

ถึงแม้ว่าการเขียนโปรแกรมแบบเรียกซ้ำ จะทำให้เราสามารถเขียนโปรแกรมได้สั้นและง่ายก็ตาม โปรแกรมที่ได้จะมีประสิทธิภาพในการประมวลผลไม่ด้นัก ถ้าไม่จำเป็นก็ควรหลีกเลี่ยงการเขียนโปรแกรมในลักษณะดังกล่าว เพราะส่วนมากมักใช้การทำงานแบบวนซ้ำแทนได้ เพียงแต่ต้องปรับปรุงขั้นตอนวิธีให้เหมาะสม

5.4.2 สัญกรณ์เติมกลางและสัญกรณ์เติมหลัง

การเขียนนิพจน์การคำนวณทางคณิตศาสตร์โดยทั่วไปจะอยู่ในรูปแบบ ที่เรียกว่า สัญกรณ์เติมกลาง (Infix Notation) ดังรูป 5.10



รูป 5.10 นิพจน์คณิตศาสตร์ในรูปสัญกรณ์เติมกลาง

โดยตัวถูกดำเนินการ (operand) อาจเป็นตัวแปร หรือค่าคงที่ และตัวกระทำ (operator) ได้แก่ เครื่องหมายสำหรับการคำนวณต่าง ๆ มักประกอบด้วย

- เครื่องหมายยกกำลัง (จะแทนด้วยสัญลักษณ์ ^)
- คูณ,หาร (*) , (/)
- บวก,ลบ (+, -)

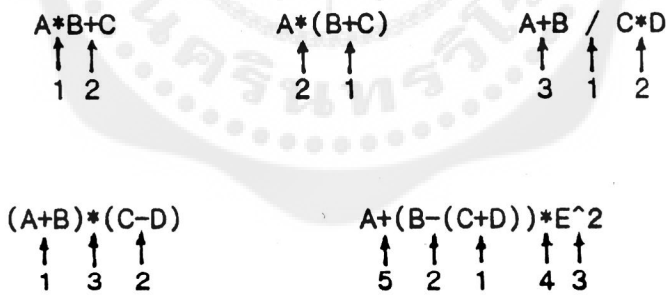
โดยมีการกำหนดลำดับการหาค่าก่อน (precedence) ของการดำเนินการไว้อย่างแน่นอนว่า ถ้านิพจน์ประกอบด้วยหลายเครื่องหมายคำนวณแล้วเครื่องหมายใดจะถูกทำงานก่อน โดยทั่วไปกำหนดไว้ดังรูป 5.11

เครื่องหมาย	ความสำคัญ
$\wedge$ $*$ / $+$ -	สูง ↓ ต่ำ

รูป 5.11 ลำดับการทำงานของเครื่องหมายคำนวณ

และข้อกำหนดในการคำนวณค่าทางนิพจน์จะเป็นดังนี้

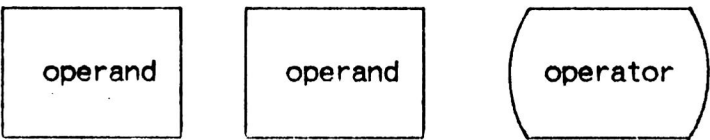
- เครื่องหมายที่มีลำดับการทำงานสูงกว่าจะถูกดำเนินการก่อน แต่ถ้ามีเครื่องหมายที่มีลำดับเท่ากันจะดำเนินการจากซ้ายไปขวา
- ถ้ามีเครื่องหมายวงเล็บ () จะดำเนินการนิพจน์ในวงเล็บ จากวงเล็บในสุดก่อน ดังตัวอย่างในรูป 5.12



รูป 5.12 ลำดับการดำเนินการนิพจน์คำนวณ

การคำนวณค่าของนิพจน์แบบเติมกลาง (Infix expression) นี้ ต้องตรวจหาลำดับความสำคัญ (priority) ของเครื่องหมายก่อนจึงสามารถทำงานได้ ซึ่งไม่สะดวกสำหรับการดำเนินการด้วยคอมพิวเตอร์ ตัวแปลชุดคำสั่งจะเปลี่ยนรูปคำสั่งคำนวณจากรูปแบบที่เขียนในภาษาระดับสูงซึ่งใช้นิพจน์แบบเติมกลาง เป็นชุดคำสั่งภาษาเครื่องในรูปนิพจน์แบบเติมหลัง (Postfix

expression) ซึ่งเป็นนิพจน์ที่ใส่เครื่องหมายไว้หลังตัวถูกดำเนินการทั้ง 2 ตัว ดังรูป 5.13 และตัวอย่างนิพจน์ทั้ง 2 แบบดังรูป 5.14



รูป 5.13 นิพจน์คณิตศาสตร์ในรูปสัญลักษณ์เติมหลัง

นิพจน์แบบเติมกลาง	นิพจน์แบบเติมหลัง
A+B	AB+
A-B	AB-
A*B	AB*
A/B	AB/
A^B	AB^

รูป 5.14 ตัวอย่างนิพจน์แบบเติมกลางและแบบเติมหลัง

ลักษณะที่สำคัญของนิพจน์แบบเติมหลัง คือ

1. เครื่องหมายทุกตัวจะต้องอยู่หลังตัวถูกดำเนินการ 2 ตัว เสมอ
2. เครื่องหมายทุกตัวจะถูกวางอยู่ในลำดับที่ถูกต้องสำหรับการคำนวณ (ตัวซ้ายสุดคำนวณตัวแรก จนถึงตัวขวาสุดจะถูกคำนวณสุดท้าย)
3. ไม่มีเครื่องหมายวงเล็บในนิพจน์

การใช้นิพจน์แบบเต็มหลัง ช่วยลดปัญหา 2 ประการคือ

1. ลดความยุ่งยากในการใส่วงเล็บเพื่อแสดงลำดับการทำงานก่อน-หลังของเครื่องหมาย
2. ไม่ต้องตรวจหาลำดับความสำคัญของเครื่องหมายในนิพจน์

วิธีการเปลี่ยนนิพจน์แบบเต็มกลางเป็นนิพจน์แบบเต็มหลัง (ด้วยมือ)

1. นำนิพจน์นิพจน์แบบเต็มกลางมาใส่เครื่องหมายวงเล็บคร่อมการทำงานตามลำดับให้ถูกต้อง
2. เปลี่ยนรูปจากนิพจน์แบบเต็มกลาง มาเป็นนิพจน์แบบเต็มหลัง โดยนำเครื่องหมายมาแทนที่เครื่องหมายวงเล็บปิด ทีละตัวจากวงเล็บในสุดจนหมดทั้งนิพจน์
3. ถอดวงเล็บทั้งหมดออก

ตัวอย่าง

$$(1) A+B*C \longrightarrow (A+(B*C)) \longrightarrow (A+(BC*)) \longrightarrow (ABC*+) \longrightarrow ABC*+$$

$$(2) A/B^{\wedge}C+D*E-A*C$$

$$(((A/(B^{\wedge}C)))+(D*E))-(A*C))$$

(ตอบ $ABC^{\wedge}/DE*+AC*-)$)

$$(3) A/B^{\wedge}(C+D)*(E-A)/C+G$$

$$((((A/(B^{\wedge}(C+D))))*(E-A))/C)+G)$$

(ตอบ $ABCD+^{\wedge}/EA-*C/G+$)

สังเกต 1. ลำดับของตัวถูกดำเนินการไม่เปลี่ยนแปลง

2. ลำดับเครื่องหมาย จะเรียงตามลำดับการทำงานที่ถูกต้องจากซ้ายไปขวา

ขั้นตอนวิธีสำหรับการเปลี่ยนนิพจน์แบบเต็มกลางเป็นนิพจน์แบบเต็มหลัง โดยใช้สแต็ก

ขั้นตอนวิธีนี้จะอ่านนิพจน์แบบเต็มกลางเข้ามา และตรวจสอบค่าที่เข้ามาว่า เป็นเครื่องหมายคำนวณ หรือเป็นตัวถูกดำเนินการ หรือเครื่องหมายวงเล็บเปิด-ปิด ซึ่งจะมีการทำงานที่แตกต่างกันไป จะใช้สแต็กสำหรับเก็บเครื่องหมายคำนวณและเครื่องหมายวงเล็บเปิดเท่านั้น และพิมพ์ผลลัพธ์เป็นนิพจน์แบบเต็มหลัง

กำหนดให้ ข้อมูลนำเข้าเป็นนิพจน์ ซึ่งตัวถูกดำเนินการเป็นอักขระเดี่ยวๆ เช่น X เครื่องหมายคำนวณประกอบด้วย: +, -, *, /, ^ อาจใช้วงเล็บ () คร่อมนิพจน์ได้ และกำหนดให้ใช้ ";" แสดงการจบนิพจน์

ตัวอย่างการแปลงนิพจน์แบบเต็มกลางเป็นเต็มหลัง โดยใช้สแต็ก

ข้อมูลเข้า : ((A+B)*C/D+E^F)/G;

ลำดับการตรวจสอบนิพจน์	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
ข้อมูล	(	(	A	+	B	)	*	C	/	D	+	E	^	F	)	/	G	;
Top ---->	(	(	(	+	+	(	*	*	/	/	+	+	^	^		/	/	
สแต็ก		(				(								(				
ผลลัพธ์			A		B	+		C	*	D	/	E		F	^		G	/

ผลลัพธ์ : A B + C * D / + E F ^ + G /

รูป 5.15 ตัวอย่างการแปลงนิพจน์แบบเต็มกลางเป็นนิพจน์เต็มหลังโดยใช้สแต็ก

แบบฝึกหัด

- เขียนโปรแกรมพิมพ์สามเหลี่ยมพาสกาล (Pascal's triangle of binomial coefficients) ซึ่งมีลักษณะดังรูป

```

      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
  .....
  
```

โดยที่ค่าของสมาชิกในแต่ละแถวจะได้จากผลบวกของสมาชิก 2 ตัว ของแถวบนที่อยู่ตรงกับสมาชิกตัวนั้น จงเขียนโปรแกรมเพื่อหาค่า 10 บรรทัดแรกของสามเหลี่ยมพาสกาล

- เขียนโปรแกรมแปลงเลขฐานจากค่าในฐานสิบเป็นฐาน r ใดๆ ($r \geq 2$) โดยใช้สแต็กเก็บค่าเศษเหลือจากการหาร และพิมพ์ผลลัพธ์
- เขียนโปรแกรมแปลงนิพจน์แบบเต็มกลางเป็นนิพจน์แบบเต็มหลัง
- วัตถุประสงค์ของการใช้สแต็กสำหรับการทำงานของโปรแกรมย่อยแบบเรียกซ้ำ คืออะไร
- จากวิธีการหา หรม. (Greates common divisor) ของเลข 2 จำนวนโดยวิธี Euclid's algorithm กำหนดนิยามได้ดังนี้

$$\text{GCD}(M, N) = \begin{cases} \text{GCD}(N, M) & \text{เมื่อ } N > M \\ M & \text{เมื่อ } N = 0 \\ \text{GCD}(N, M \bmod N) & \text{เมื่อ } N > 0 \end{cases}$$

เขียนฟังก์ชันแบบเรียกซ้ำสำหรับคำนวณหา หรม. ตามวิธีข้างต้น

6. การคำนวณค่าของนิพจน์แบบเต็มหลังต่างจากการคำนวณค่าของนิพจน์แบบเต็มกลางอย่างไร

7. แปลงนิพจน์แบบเต็มกลางต่อไปนี้ไปเป็นนิพจน์แบบเต็มหลัง

(ก) $A + (B * C)^D / (M - N)$ (ข) $A^N + (B - C) / X * Y - K / D$

(ค) $A / B + (X / Y)^N$

8. แปลงนิพจน์แบบเต็มหลังต่อไปนี้ไปเป็นนิพจน์แบบเต็มกลาง

(ก) $AB + C -$ (ข) $ABC * /$

(ค) $AB - C + DEF - + ^$ (ง) $ABCDE - + ^ * EF * -$

(จ) $AB + C *$

9. จงหาค่านิพจน์แบบเต็มหลังต่อไปนี้ เมื่อ $A=1$, $B=2$ และ $C=3$

(ก) $AB + C *$ (ข) $ABC * /$

(ค) $ABC * /$ (ง) $AB + C - BA + C ^ -$

(จ) $ABC + * CBA - + *$

บทที่ 6

โครงสร้างข้อมูลแบบรายการโยง

หัวข้อ : 6.1 บทนำ

6.2 รายการโยง

6.3 การดำเนินงานในรายการโยง

6.4 ประเภทของรายการโยง

6.5 รายการโยงเดี่ยว

6.6 การแทนรายการโยงด้วยตัวแปรแถวลำดับ

6.7 รายการโยงคู่

6.8 การดำเนินงานในรายการโยงคู่

6.9 ตัวอย่างโปรแกรม

6.1 บทนำ

จากการศึกษาเรื่อง โครงสร้างข้อมูลเชิงเส้นแบบแถวลำดับ คิว และสแตก ที่ผ่านมาเป็นการใช้เนื้อที่แบบสถิตทั้งหมดเนื่องจากเราแทนคิวและสแตกด้วยแถวลำดับ ในบทนี้จะกล่าวกล่าวถึงโครงสร้างข้อมูลแบบรายการโยง (Linked List) ซึ่งเป็นโครงสร้างข้อมูลเชิงเส้นอีกแบบหนึ่ง ที่มีการใช้เนื้อที่ของหน่วยความจำแบบไดนามิก เราสามารถพิจารณาความแตกต่างของการใช้เนื้อที่แบบสถิตและไดนามิก ได้ดังนี้

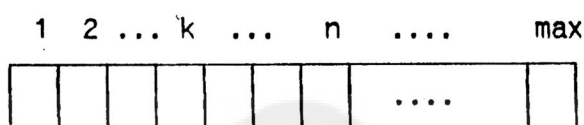
6.1.1 การใช้เนื้อที่แบบสถิต

การใช้เนื้อที่สำหรับเก็บข้อมูลจะอยู่เรียงต่อเนื่องกันไปตามลำดับในหน่วยความจำ

ข้อดี การเข้าถึงข้อมูลทำได้ง่ายและรวดเร็ว

ข้อจำกัด

1. ต้องประกาศขนาดเนื่อก่อนการประมวลผล ในบางครั้งเราไม่รู้จำนวนข้อมูลเข้าล่วงหน้าก็ต้องประกาศเผื่อเนื้อที่ไว้มากพอสมควร
2. การใส่ข้อมูลเข้าและการนำข้อมูลออกที่ต้องการรักษาลำดับของข้อมูลไว้ ต้องมีการขยับเนื้อที่ พิจารณารูป 6.1



รูป 6.1 การเก็บข้อมูลในแถวลำดับ

ถ้าจำนวนข้อมูลที่เก็บไว้ = n จำนวน

ต้องการใส่ข้อมูลเข้าในตำแหน่งที่ k ต้องขยับทุกตัวตั้งแต่ตัวที่ k ถึง n ออกไป 1

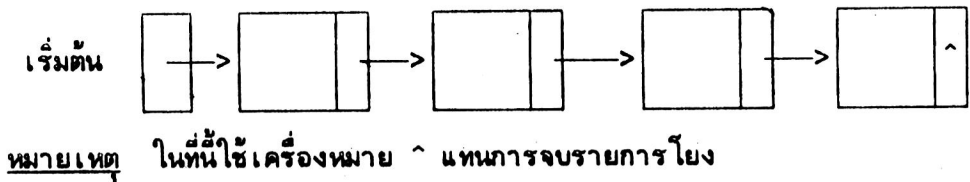
ตำแหน่ง จำนวนครั้งที่ขยับ = $n - k + 1$

ถ้าต้องการนำข้อมูลในตำแหน่งที่ k ออก ต้องขยับตัวที่ $k+1$ ถึง n เข้ามาทดแทน
จำนวนครั้งที่ขยับ = $n - k$ ครั้ง ซึ่งการขยับเนื้อที่ดังกล่าวเป็นการเสียเวลาประมวล โดยเฉพาะ
เมื่อมีการใส่ข้อมูลเข้าและนำข้อมูลออกบ่อย ๆ

6.1.2 การใช้เนื้อที่แบบไดนามิก

การเก็บข้อมูลที่ใช้การจัดสรรเนื้อที่แบบไดนามิกนั้น ตำแหน่งที่เก็บข้อมูลไม่จำเป็นต้องอยู่เรียงต่อเนื่องกัน การกำหนดลำดับข้อมูลทำโดยใช้ตัวชี้ (Pointer) หรือลิงค์ (link) เพื่อโยงลำดับของข้อมูล การอ้างอิงถึงข้อมูลแบบนี้ต้องใช้ตัวชี้เสมอ รูป 6.2 แสดงการเก็บข้อมูลโดยจัดสรรเนื้อที่ในหน่วยความจำแบบไดนามิก

การใส่ข้อมูลเข้าและนำข้อมูลออก ทำได้โดยการปรับค่าของตัวชี้ให้ถูกต้องโดยไม่ต้องเปลี่ยนตำแหน่งของข้อมูลตัวอื่นๆ การเข้าหาข้อมูลที่ต้องการจะต้องผ่านตามลำดับตั้งแต่สมาชิกตัวแรกไปตามลำดับ และการจัดเตรียมเนื้อที่จะกระทำการระหว่างการประมวลผล



รูป 6.2 รูปแผนการเก็บข้อมูลโดยจัดสรรเนื้อที่แบบไดนามิก

ข้อดี

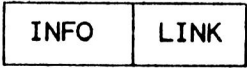
- 1. การใช้เนื้อที่เป็นไปอย่างมีประสิทธิภาพ เนื้อที่ที่ใช้จะพอดีกับข้อมูลในขณะนั้นเท่านั้น
- 2. จำนวนข้อมูลไม่จำกัดโดยโปรแกรม (แต่จำกัดโดยขนาดของหน่วยความจำ)
- 3. การใส่ข้อมูลเข้าและนำข้อมูลออก ทำได้โดยไม่ต้องเปลี่ยนตำแหน่งของข้อมูลตัวอื่น

ข้อจำกัด ใช้ได้เฉพาะในภาษาคอมพิวเตอร์บางภาษา เช่น ปาสคาล ฟอแทรน ซี เป็นต้น

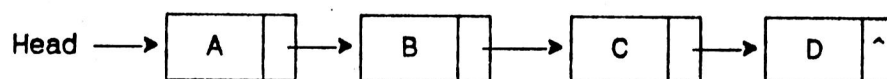
6.2 รายการโยง

รายการโยง (Linked List) เป็นโครงสร้างข้อมูลเชิงเส้น ที่ใช้หลักการจัดสรรเนื้อที่ของหน่วยความจำแบบไดนามิก ตำแหน่งที่เก็บข้อมูลของสมาชิกแต่ละตัว ไม่จำเป็นต้องอยู่ต่อเนื่องกัน การเชื่อมโยงลำดับของข้อมูลทำได้โดยใช้ตัวชี้ เนื้อที่สำหรับสมาชิกแต่ละตัว เรียกว่า โหนด (node) แต่ละโหนดประกอบด้วยเนื้อที่ 2 ส่วนคือ ส่วนสำหรับเก็บข้อมูล และส่วนสำหรับเก็บตำแหน่งของสมาชิกตัวถัดไป ดังรูป 6.3

การอ้างถึงข้อมูลในรายการโยงต้องเริ่มจากสมาชิกตัวแรกก่อนเสมอ และต้องมีตัวชี้ตัวหนึ่งเป็นตัวชี้จุดเริ่มต้นของรายการโยง ดังรูป 6.4



รูป 6.3 โหนดของรายการโยง



รูป 6.4 ตัวอย่างรายการโยง

การกำหนดโครงสร้างของโหนด

การกำหนดโครงสร้างของโหนดและตัวชี้โหนดในรายการโยง เขียนประกาศในภาษาปาสคาลได้ดังรูป 6.5

```

Type Itemtype = Char;
   Pointer = ^ Node;
   Node = Record
       Info :Itemtype;
       Link :Pointer
   End;
  
```

รูป 6.5 การกำหนดโครงสร้างของโหนด

6.3 การดำเนินงานในรายการโยง

การดำเนินงานในรายการโยง ได้แก่

1. การสร้างรายการโยง
2. การใส่ข้อมูลเข้า
3. การนำข้อมูลออก
4. การแฉะผ่านรายการโยง

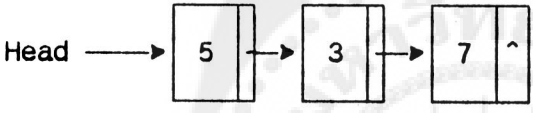
6.4 ประเภทของรายการโยง

ประเภทของรายการโยง แบ่งออกเป็น 2 ประเภท ได้แก่

1. รายการโยงเดี่ยว
2. รายการโยงคู่

6.5 รายการโยงเดี่ยว

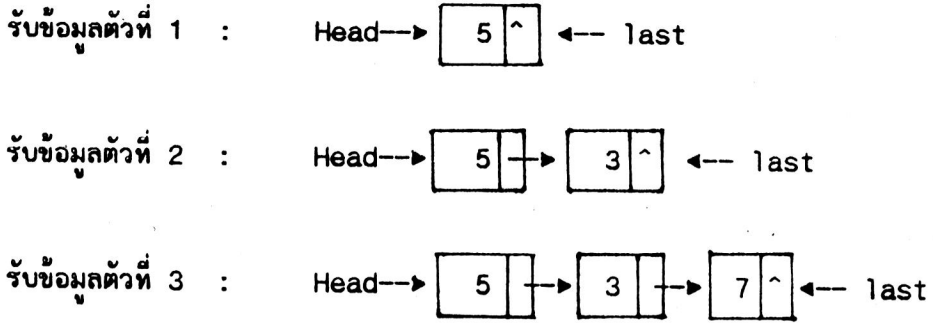
รายการโยงเดี่ยว (Singly Linked List) เป็นรายการโยงซึ่งแต่ละโหนดจะมีเพียงหนึ่งตัวชี้เท่านั้นสำหรับชี้สมาชิกตัวถัดไป ตัวชี้ของโหนดสุดท้ายจะเป็น nil และมีตัวชี้ตัวหนึ่งสำหรับชี้โหนดแรกของรายการโยง ดังรูป 6.6



รูป 6.6 ตัวอย่างรายการโยงเดี่ยว

6.5.1 การสร้างรายการโยง

การสร้างรายการโยง ได้แก่การนำข้อมูลจำนวนหนึ่งเข้ามาเก็บในรายการโยงตามลำดับ เช่น ข้อมูลเป็น 5, 3 และ 7 ต้องการเก็บด้วยรายการโยงดังรูป 6.6
 วิธีการสร้าง: ในที่นี้จะใช้ตัวชี้ 2 ตัว คือ Head ชี้โหนดแรกและ Last ชี้โหนดสุดท้าย ในขณะที่เริ่มต้นกำหนดให้ Head และ Last เป็น nil รูป 6.7 แสดงการสร้างรายการโยง



รูป 6.7 แสดงการสร้างรายการโยง

6.5.2 การแหว่ม่านรายการโยง

การแหว่ม่านรายการโยง (Travel List) เป็นการเข้าถึงข้อมูลในรายการโยง โดยเข้าแหว่เยียมทีละโหนดจากต้นรายการโยงไปตามลำดับ โดยไปตามทางของตัวชี้จนกว่าพบว่า เป็น nil

วิธีการ สมมติให้ t เป็นตัวชี้ของโหนดที่กำลังเยียมอยู่ในขณะนั้น

เริ่มต้นจากโหนดแรก :

t := head;

เยียมไปทีละโหนดจนกว่าตัวชี้เป็น nil :

```
While t <> nil then
begin visit node t;
      t := t^.link {ไปโหนดถัดไป}
end;
```

ตัวอย่างที่ 1 ขั้นตอนวิธีแหว่ม่านรายการโยงและพิมพ์ข้อมูลของแต่ละโหนดออกมาตามลำดับ แสดง
ในรูป 6.8

```
Procedure Printlist (Head:pointer);
Var      t :pointer;
Begin
    t := head;
    While t <> nil Do
        Begin
            writeln (t^.info);
            t := t^.link
        End
    End;
End;
```

รูป 6.8 ขั้นตอนวิธีแหว่ม่านรายการโยงและพิมพ์ข้อมูลของแต่ละโหนดออกมาตามลำดับ

ตัวอย่างที่ 2 การค้นข้อมูลที่ต้องการในรายการโยง โดยใช้ฟังก์ชัน Search ค้นหาว่ามีข้อมูลที่เท่ากับค่า x หรือไม่ อยู่ที่ตำแหน่งใด ถ้าไม่พบข้อมูลคืนค่าเป็น nil ขั้นตอนวิธีแสดงในรูป 6.9

```
Function Search (head:pointer; x:integer):pointer;
Var    t : pointer;
        found : boolean;

Begin
    t := head;
    found := false;
    While (t <> nil) AND (Not found) Do
        Begin
            if t^.info = x
            then found := true
            else t := t^.link;
        End;
    Search := t;
End;
```

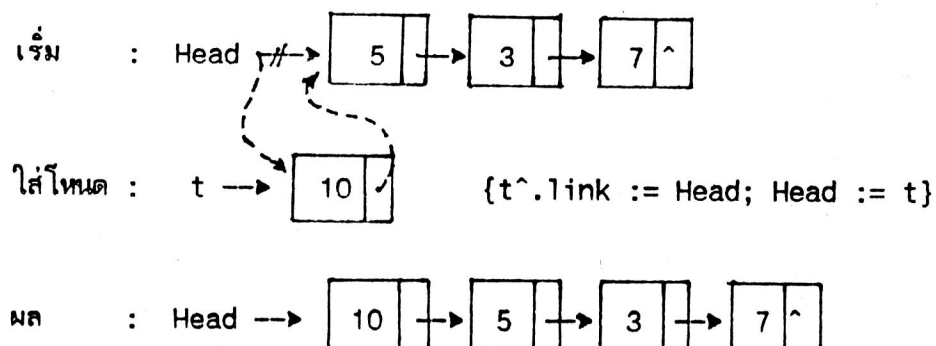
รูป 6.9 ขั้นตอนวิธีการค้นข้อมูลในรายการโยง

ถ้าพบจะคืนค่า Search ชี้ตำแหน่งของโหนดที่พบ ถ้าไม่พบจะคืนค่าเป็น nil

6.5.3 การใส่ข้อมูลเข้าในรายการโยง

การใส่ข้อมูลเข้าในรายการโยง อาจแยกพิจารณาเป็น 3 วิธี ได้แก่ การใส่ข้อมูลเข้าต้นรายการโยง การใส่ข้อมูลเข้ากลางรายการโยง การใส่ข้อมูลเข้าต่อท้ายรายการโยง

(1) การใส่ข้อมูลเข้าต้นรายการโยง มีวิธีการดังตัวอย่างรูป 6.10



รูป 6.10 ตัวอย่างการใส่ข้อมูลเข้าต้นรายการโยง

ขั้นตอนวิธีสำหรับการใส่ข้อมูลเข้าต้นรายการโยง แสดงในรูป 6.11

```

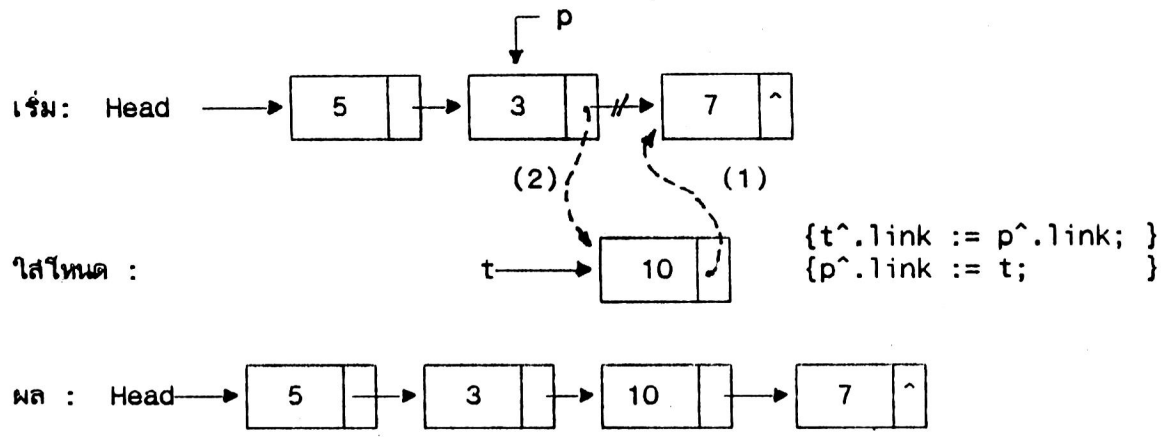
Procedure Inserthead (Var head:pointer;x:integer);
Var    t: pointer;
Begin
    new (t);
    t^.info := x;
    t^.link := head;
    head    := t
End;

```

รูป 6.11 ขั้นตอนวิธีการใส่ข้อมูลเข้าต้นรายการโยง

(2) การใส่ข้อมูลเข้ากลางรายการโยง

การใส่ข้อมูลเข้ากลางรายการโยงเป็นการใส่ข้อมูลโหนดใหม่แทรกกลางรายการโยง หลังตำแหน่งของโหนดใดโหนดหนึ่ง ดังรูป 6.12



รูป 6.12 ตัวอย่างการใส่ข้อมูลเข้ากลางรายการโยง

ขั้นตอนการใส่ข้อมูลเข้ากลางรายการโยง แสดงในรูป 6.13 เป็นการใส่ข้อมูลต่อจากโหนดที่ตัวชี้ p ชี้อยู่

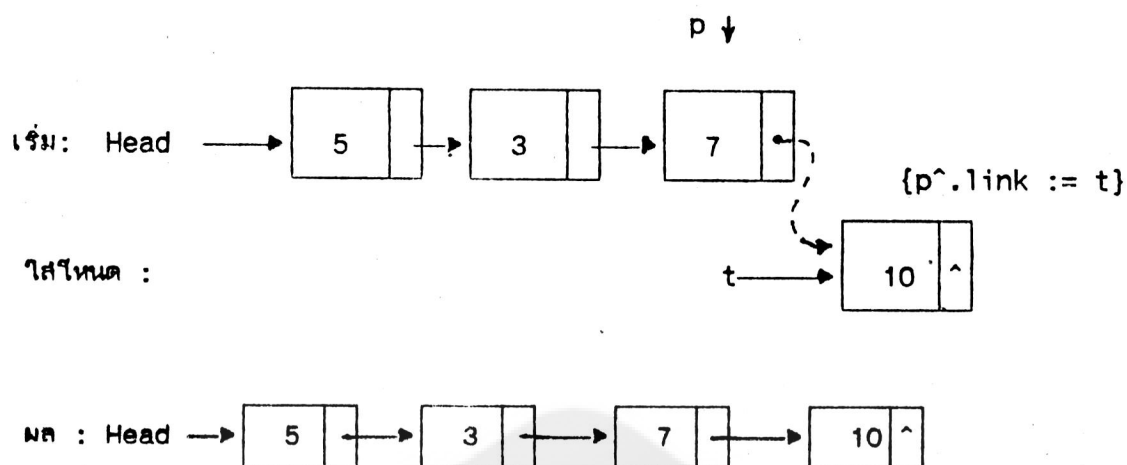
```

Procedure Insert_after (p:pointer; x:integer);
Var    t : pointer;
Begin
    new(t);
    t^.info:=x;
    t^.link:=p^.link;
    p^.link:=t
End;

```

รูป 6.13 ขั้นตอนวิธีการใส่ข้อมูลเข้ากลางรายการโยง

(3) การใส่ข้อมูลต่อท้ายรายการโยง ตัวอย่างการใส่ข้อมูลต่อท้ายรายการโยงดังรูป 6.14



รูป 6.14 ตัวอย่างการใส่ข้อมูลต่อท้ายรายการโยง

การใส่ข้อมูลต่อท้ายรายการโยง ต้องค้นหาตำแหน่งของโหนดสุดท้ายก่อนแล้วจึงเชื่อมโหนดขั้นตอนวิธีการใส่ข้อมูลต่อท้ายรายการโยง แสดงในรูป 6.15

6.5.4 การนำข้อมูลออกจากรายการโยง

การนำข้อมูลออกจากรายการโยง ต้องกำหนดว่าจะนำข้อมูลที่อยู่ต่อจากโหนดในตำแหน่งใดออกไป ดังตัวอย่างในรูป 6.16 ซึ่งจะลบโหนดที่อยู่ต่อจากโหนดที่ p อี้ออกไปขั้นตอนวิธีการนำข้อมูลออกจากรายการโยง แสดงในรูป 6.17

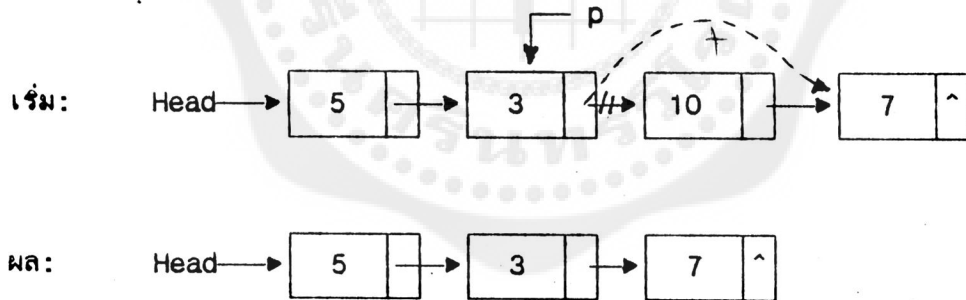
ข้อสังเกต การใส่ข้อมูลเข้าและการนำข้อมูลออกจากรายการโยง มักต้องกระทำหลังตำแหน่งที่ระบุเสมอ เนื่องจากข้อจำกัดของตัวชี้ที่ชี้ไปทางด้านหลังทางเดียว กรณีที่ต้องการใส่ข้อมูลหรือลบข้อมูลหน้าโหนดที่ระบุ ต้องใช้วิธีค้นหาตำแหน่งของโหนดข้างหน้าโหนดที่ระบุก่อน ซึ่งต้องใช้ตัวชี้อีกตัวหนึ่งมาช่วยจำตำแหน่งของโหนดข้างหน้า

```

Procedure Insert_tail (Head:pointer;x:integer);
Var  p, t : pointer;
Begin
    new(t);
    t^.info:=x;
    t^.link:=nil;
    {find last node}
    p:=head;
    While p^.link <> nil do
        p:=p^.link;
    {link with node t^}
    p^.link:=t;
End;

```

รูป 6.15 ขั้นตอนวิธีการใส่ข้อมูลต่อท้ายรายการโยง



รูป 6.16 ตัวอย่างการนำข้อมูลออกจากรายการโยง

```

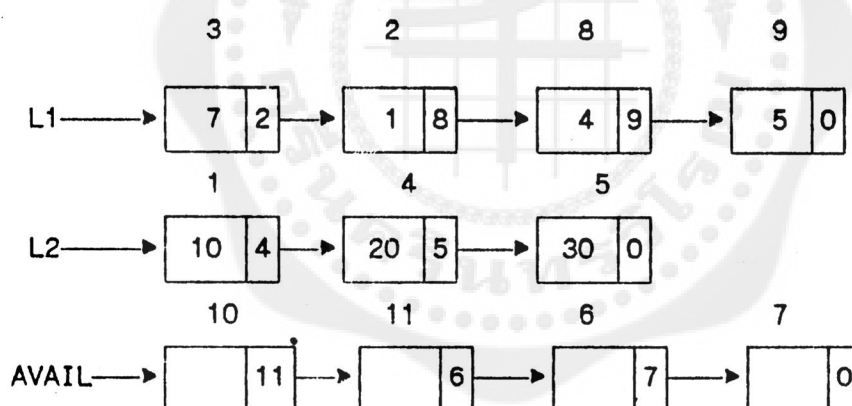
Procedure Delete_node(p:pointer);
Var t :pointer;
Begin {delete after node p}
    t := p^.link;
    p^.link := t^.link;
    Dispose(t)
End;

```

รูป 6.17 ขั้นตอนวิธีการนำข้อมูลออกจากรายการโยง

6.6 การแทนรายการโยงด้วยตัวแปรแถวลำดับ

บางกรณีเราอาจใช้ตัวแปรแถวลำดับแทนรายการโยง เช่น ในภาษาที่ไม่สามารถใช้ตัวแปรไดนามิกได้ แสดงได้ดังรูป 6.18 ซึ่งใช้ตัวแปรแถวลำดับเก็บข้อมูล 3 รายการโยง ดังนี้

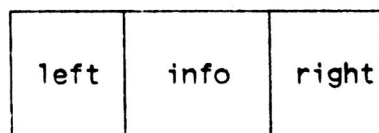


		INFO	LINK
L2	1	1	10
		2	1
L1	3	3	7
		4	20
		5	30
		6	
AVAIL	10	7	
		8	4
		9	5
		10	
		11	

รูป 6.18 ตัวอย่างการแทนรายการโยงด้วยแถวลำดับ

6.7 รายการโยงคู่

รายการโยงคู่ (Doubly Linked List) เป็นรายการโยงที่ประกอบด้วยโหนดที่มีตัวชี้ 2 ตัว ดังรูป 6.19 left เป็นตัวชี้โหนดก่อนหน้า และ right เป็นตัวชี้โหนดถัดไป



รูป 6.19 โหนดของรายการโยงคู่

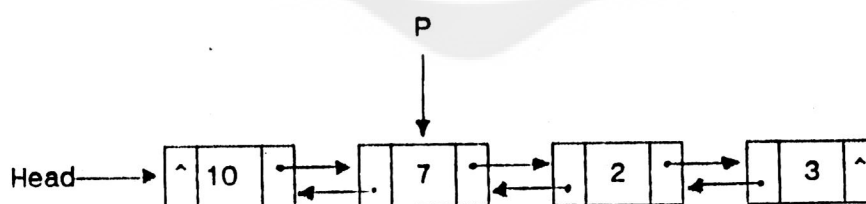
การกำหนดโครงสร้างของโหนด เขียนส่วนการประกาศได้ดังรูป 6.20

```

TYPE pointer = ^node
      node   = Record
                left, right: pointer;
                info = integer
            End;
  
```

รูป 6.20 การกำหนดโครงสร้างของโหนดในรายการโยงคู่

ลักษณะของรายการโยงคู่ เป็นดังรูป 6.21



รูป 6.21 ตัวอย่างรายการโยงคู่

ข้อดีของรายการโยงคู่ คือ การอ้างอิงถึงโหนดก่อนหน้า และ โหนดถัดไปของโหนดใดโหนดหนึ่ง สามารถกระทำได้ไปเรื่อย ๆ ถึงต้นและท้ายรายการโยง ต่างจากรายการโยงเดี่ยวซึ่งที่อ้างอิงได้เฉพาะโหนดถัดไปเท่านั้น

ตัวอย่าง ให้ left และ right หมายถึงตัวชี้ทางซ้ายและตัวชี้ทางขวาของโหนดตามลำดับ และ P เป็นตัวชี้โหนด ดังรูป 6.21 การอ้างอิงข้อมูลเป็นดังนี้

$p^{\wedge}.right^{\wedge}.info$ มีค่า 2
 $p^{\wedge}.right^{\wedge}.right^{\wedge}.info$ มีค่า 3
 $p^{\wedge}.right^{\wedge}.right^{\wedge}.right$ มีค่า nil
 $p^{\wedge}.left^{\wedge}.info$ มีค่า 10
 $p^{\wedge}.left^{\wedge}.left$ มีค่า nil
 $p^{\wedge}.right^{\wedge}.left$ ชี้มาที่โหนด p
 $p^{\wedge}.left^{\wedge}.right$ ชี้มาที่โหนด p

6.8 การดำเนินงานในรายการโยงคู่

การดำเนินงานในรายการโยงคู่ ในที่นี้จะกล่าวถึงเฉพาะการใส่ข้อมูลเข้าและนำข้อมูลออกเท่านั้น

6.8.1 การใส่ข้อมูลเข้าในรายการโยงคู่

การใส่ข้อมูลเข้าในรายการโยงคู่ สามารถกระทำได้ทั้งหน้าโหนดและหลังโหนดที่ระบุ โดยรับตัวชี้ทางซ้ายและทางขวาให้ถูกต้อง

(1) การใส่ข้อมูลเข้าในรายการโยงคู่หลังโหนดที่ระบุ สามารถกระทำได้ดังตัวอย่างรูป 6.22 ซึ่งใส่โหนดที่ชี้โดย t เข้าต่อท้ายโหนดที่ p ช้อยู่ โดยขั้นตอนวิธีการใส่ข้อมูลเป็นดังนี้

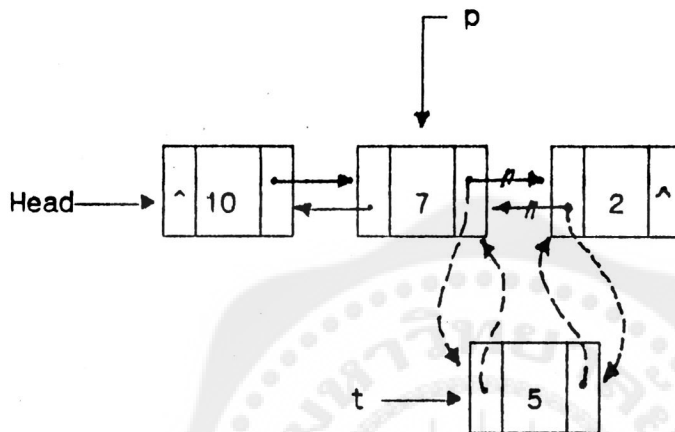
การใส่ข้อมูลเข้าในรายการโยงคู่

```
{เชื่อมตัวชี้ทางซ้ายและขวาของ  $t^{\wedge}$  }
 $t^{\wedge}.left := p;$ 
 $t^{\wedge}.right := p^{\wedge}.right;$ 
```

{ปรับตัวชี้ของโหนดในรายการโยงชี้มาที่ t}

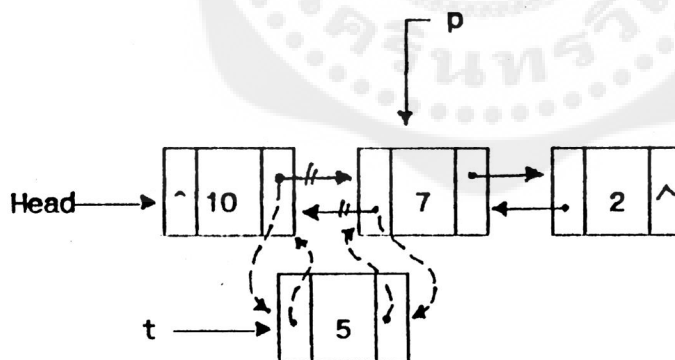
$p^{\wedge}.right^{\wedge}.left := t;$

$p^{\wedge}.right := t;$



รูป 6.22 ตัวอย่างการใส่ข้อมูลในรายการโยงคูลังโหนดที่ระบุ

(2) การใส่ข้อมูลเข้าในรายการโยงคูลหน้าโหนดที่ระบุ รูป 6.23 แสดงตัวอย่างการใส่ข้อมูลเข้าในรายการโยงคูลหน้าโหนด p



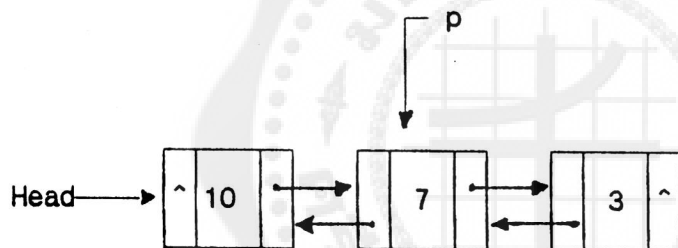
รูป 6.23 ตัวอย่างการใส่ข้อมูลในรายการโยงคูลหน้าโหนดที่ระบุ

ขั้นตอนวิธีการใส่ข้อมูลในรายการโยงคู่หน้า โหนดที่ระบุ เป็นดังนี้

```
{ เชื่อมตัวชี้ทางซ้ายและขวาของ  $t^{\wedge}$  }
 $t^{\wedge}.\text{right} := p;$ 
 $t^{\wedge}.\text{left} := p^{\wedge}.\text{left};$ 
{ ปรับตัวชี้ของโหนดในรายการโยงชี้มาที่  $t$  }
 $p^{\wedge}.\text{left}^{\wedge}.\text{right} := t;$ 
 $p^{\wedge}.\text{left} := t;$ 
```

6.8.2 การนำข้อมูลออกจากรายการโยงคู่

การนำข้อมูลออกจากรายการโยงคู่ทำได้โดยการปรับตัวชี้ที่ชี้โหนดที่ต้องการลบออกให้ชี้ข้ามไปยังโหนดถัดไป ดังตัวอย่างรูป 6.24



รูป 6.24 ตัวอย่างการลบโหนด

ขั้นตอนวิธีการนำข้อมูลโหนดที่ p ชี้อยู่ออกจากรายการโยง

```
{ ปรับตัวชี้ที่ชี้  $p$  ให้ข้ามไปชี้โหนดถัดไป }
 $p^{\wedge}.\text{left}^{\wedge}.\text{right} := p^{\wedge}.\text{right};$ 
 $p^{\wedge}.\text{right}^{\wedge}.\text{left} := p^{\wedge}.\text{left};$ 
{ อาจตัดตัวชี้ของโหนดที่ลบออกให้เป็น nil และคืนเนื้อที่ของโหนด }
 $p^{\wedge}.\text{left} := \text{nil};$ 
 $p^{\wedge}.\text{right} := \text{nil};$ 
Dispose ( $p$ );
```

พิจารณาการปรับตัวชี้ในการนำโหนดออกจากรายการโยง ควรมีการตรวจสอบด้วยว่าเป็นการลบโหนดต้นรายการโยงหรือท้ายรายการโยงหรือไม่ เนื่องจากถ้าเป็นโหนดต้นหรือท้ายรายการโยงจะมีตัวชี้ข้างหนึ่งของโหนดเป็น nil ดังนั้นปรับปรุงคำสั่งสำหรับโยงตัวชี้ได้ดังนี้

- ถ้าไม่ใช่โหนดท้ายรายการโยง ให้ปรับตัวชี้ทางซ้ายของโหนดถัดไปให้ชี้ข้ามโหนด p

```
if p^.right <> nil
```

```
then p^.right^.left := p^.left;
```

- ถ้าไม่ใช่โหนดต้นรายการโยง ให้ปรับตัวชี้ทางขวาของโหนดถัดไปให้ชี้ข้ามโหนด p

```
if p^.left <> nil
```

```
then p^.left^.right := p^.right;
```

6.9 ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรมต่อไปนี้ แสดงขั้นตอนวิธีดำเนินงานต่าง ๆ ในรายการโยงเดี่ยว ได้แก่

- การสร้างรายการโยง
- การแสดงข้อมูลในรายการโยง
- การใส่ข้อมูลเข้า
- การลบข้อมูลออก
- และ - การค้นข้อมูลในรายการโยง

โดยผู้ใช้สามารถเลือกงานจากเมนู การป้อนข้อมูลผ่านทางแป้นพิมพ์และแสดงผลทางจอภาพ การทดลองปฏิบัติงานโปรแกรมนี้จะช่วยให้เข้าใจโครงสร้างแบบรายการโยง ได้ดียิ่งขึ้น

```

PROGRAM LINK_DEMO;
{ This program demonstrates linked list operations,      }
{ such as create linked list, traverse linked list,      }
{ delete node, insert node, and search linked list.      }
{ This program was tested on Turbo Pascal version 5.0    }
{ and 6.0                                                }
USES      CRT;
TYPE      ITEMTYPE = STRING[10];
          POINTER  = ^NODE;
          NODE     = RECORD
                        INFO : ITEMTYPE;
                        LINK  : POINTER
                      END;
VAR CHOICE      : CHAR;
    HEAD, P     : POINTER;
    X           : ITEMTYPE;

PROCEDURE UNDERLINE;
BEGIN
  WRITELN('-----');
END;

PROCEDURE DISPLAYLIST (HEAD :POINTER);
VAR   T : POINTER;
BEGIN
  UNDERLINE;
  T := HEAD;
  WHILE T <> NIL DO
    BEGIN
      WRITE ('-->[' ,T^.INFO,']');
      T := T^.LINK
    END;
  WRITELN;
  UNDERLINE;
END;

PROCEDURE PRINTLIST (HEAD :POINTER);
VAR   T : POINTER;
      CH:CHAR;
BEGIN
  CLRSCR; WRITELN;
  WRITELN('** DISPLAY LINKED LIST ***');
  DISPLAYLIST(HEAD);
  WRITE('Press Any Key....');
  CH := READKEY;
  WRITELN
END;

PROCEDURE INSERT_HEAD (VAR HEAD:POINTER);
VAR T:POINTER;
    X :ITEMTYPE;
    CH:CHAR;
BEGIN

```

```

CLRSOR; WRITELN;
WRITELN('*** INSERT NEW HEAD NODE ***');
UNDERLINE;
WRITE('ENTER NEW ELEMENT VALUE : ');
READLN(X);
NEW(T);
T^.INFO := X;
T^.LINK := HEAD;
HEAD := T;
UNDERLINE;
WRITELN;
WRITELN('Already Inserted, Press Any Key...');
CH:=READKEY;
WRITELN
END;

PROCEDURE INSERT_AFTER_KEY(VAR HEAD:POINTER);
VAR KEY,X :ITEMTYPE;
    T,P :POINTER;
    CH :CHAR;
    FOUND :BOOLEAN;
BEGIN
    CLRSOR; WRITELN;
    WRITELN('*** INSERT NEW NODE AFTER SPECIFIED NODE ***');
    WRITELN('=====');
    WRITELN;
    WRITELN('*** LINKED LIST BEFORE INSERTION ***');
    DISPLAYLIST(HEAD);
    WRITELN;
    WRITELN('Look at the linked list...');
    WRITELN('Now, tell me you wish to Enter New element After Which Node');
    WRITE('Node Which Key ='); READLN(KEY);
    { Search node position }
    P:= HEAD;
    FOUND :=FALSE;
    WHILE (P <> NIL) AND (NOT FOUND) DO
        BEGIN
            IF P^.INFO = KEY THEN
                FOUND := TRUE
            ELSE BEGIN
                T := P;
                P := P^.LINK;
            END;
        END; {WHILE}
    IF NOT FOUND THEN
        BEGIN
            WRITELN; UNDERLINE;
            WRITELN(CHR(7),'-->Not Found, Invalid key...');
        END
    ELSE
        BEGIN
            WRITELN('OK! I found it, Now insert New Element value');
            READLN(X);
            NEW(T);

```

```

        T^.INFO :=X;
        T^.LINK := P^.LINK;
        P^.LINK := T;
        WRITELN; UNDERLINE;
        WRITELN('*** LINKED LIST AFTER INSERTION ***');
        DISPLAYLIST(HEAD)
    END;

    WRITELN;
    WRITE('Press Any Key...');
    CH := READKEY;
    WRITELN
END;

```

```

PROCEDURE INSERT_TAIL (HEAD:POINTER);
VAR P, T :POINTER;
    X      :ITEMTYPE;
    CH      :CHAR;
BEGIN
    CLRSCR; WRITELN;
    WRITELN('*** INSERT NEW TAIL NODE ***');
    UNDERLINE;
    WRITE('ENTER NEW ELEMENT VALUE : ');
    READLN(X);
    NEW(T);
    T^.INFO := X;
    T^.LINK := NIL;
    P := HEAD;
    WHILE P^.LINK <> NIL DO
        P := P^.LINK;
    P^.LINK := T;
    UNDERLINE;
    WRITELN('Already Inserted, Press Any Key...');
    CH:=READKEY;
    WRITELN
END;

```

```

PROCEDURE INS_MENU;
VAR C :CHAR;
BEGIN
    CLRSCR;
    GOTOXY(25,5);
    WRITELN ('INSERTION A NODE AT SPECIFIED POSITION ');
    GOTOXY(25,7);
    WRITELN ('<1> INSERT_HEAD ');
    GOTOXY(25,8);
    WRITELN ('<2> INSERT_AFTER_SPECIFIED_KEY ');
    GOTOXY(25,9);
    WRITELN ('<3> INSERT_TAIL ');
    GOTOXY(25,10);
    WRITE('SELECT <1-3> : ');
    READLN (C);
    CASE C OF
        '1' : INSERT_HEAD(HEAD);
        '2' : INSERT_AFTER_KEY(HEAD);

```

```

    '3' : INSERT_TAIL(HEAD)
END
END;

PROCEDURE DELETE(VAR HEAD:POINTER);-
VAR KEY :ITEMTYPE;
    T,P :POINTER;
    CH :CHAR;
    FOUND :BOOLEAN;
BEGIN
    CLRSCR; WRITELN;
    WRITELN('*** LINKED LIST BEFORE DELETION ***');
    DISPLAYLIST(HEAD);
    WRITELN;
    WRITE('ENTER KEY OF NODE TO BE DELETED: ');
    READLN(KEY);
    { Search for node to be deleted }
    P:= HEAD; FOUND := FALSE;
    IF (HEAD^.INFO = KEY) THEN
        BEGIN {Delete head node}
            FOUND := TRUE;
            HEAD := HEAD^.LINK
        END
    ELSE
        BEGIN
            FOUND :=FALSE;
            WHILE (P <> NIL) AND (NOT FOUND) DO
                BEGIN
                    IF P^.INFO = KEY THEN
                        FOUND := TRUE
                    ELSE BEGIN
                        T := P;
                        P := P^.LINK;
                    END;
                END; {WHILE}
            IF FOUND THEN
                T^.LINK := P^.LINK; {Delete non head node}
            END;
            IF NOT FOUND THEN
                BEGIN
                    WRITELN; UNDERLINE;
                    WRITELN(CHR(7),'-->Not Found, Invalid key...');
                END
            ELSE
                BEGIN
                    WRITELN; UNDERLINE;
                    WRITELN('*** LINKED LIST AFTER DELETION ***');
                    DISPLAYLIST(HEAD)
                END;
            WRITELN;
            WRITE('Press Any Key...');
            CH := READKEY;
            WRITELN
        END;
END;

```

```

PROCEDURE SEARCH( HEAD:POINTER);
VAR P,T      :POINTER;
    KEY      :ITEMTYPE;
    I        :INTEGER;
    CH       :CHAR;
    FOUND    :BOOLEAN;
BEGIN
    CLRSCR;  WRITELN;
    WRITELN('*** SEARCH LINKED LIST ***');
    UNDERLINE;
    WRITELN('*** DISPLAY LINKED LIST ***');
    DISPLAYLIST(HEAD);
    WRITE('ENTER SEARCH KEY: ');
    READLN(KEY);
    FOUND := FALSE;
    I := 1;
    P := HEAD;
    WHILE (NOT FOUND) AND (P<>NIL) DO
        BEGIN {Search node of specified key}
            IF P^.INFO = KEY THEN
                BEGIN FOUND := TRUE;
                    T := P
                END
            ELSE
                BEGIN
                    I := I + 1;
                    P := P^.LINK
                END
            END;
        IF FOUND THEN
            BEGIN
                WRITELN('Found At Node # ',i);
                WRITELN('VALUE OF NODE ',I,' IS ',T^.INFO);
            END
        ELSE
            WRITELN(CHR(7),'--> Not Found, Invalid Key...');
        UNDERLINE; WRITELN;
        WRITE('Press Any Key..');
        CH := READKEY;
        WRITELN
    END;

PROCEDURE CREATE (VAR HEAD :POINTER);
VAR T, LAST :POINTER;
    X       :ITEMTYPE;
BEGIN {Create linked list, insert at tail}
    CLRSCR;
    WRITELN ('*** CREATE LINKED LIST *** ');
    UNDERLINE;
    WRITELN ('ENTER ELEMENT VALUE  [ * FOR END] ');
    HEAD:=NIL;
    READLN (X);
    WHILE X <> '*' DO

```

```

BEGIN
  NEW (T);
  T^.INFO := X;
  T^.LINK := NIL;
  IF HEAD = NIL THEN
    BEGIN
      HEAD := T;
      LAST := T
    END
  ELSE
    BEGIN
      LAST^.LINK := T;
      LAST := T
    END;
  READLN (X)
END
END;

BEGIN {MAIN}
  HEAD := NIL; {Initial head pointer}
  REPEAT
    CLRSCR;
    GOTOXY(28,5);
    WRITELN('< LINKED LIST DEMO > ');
    GOTOXY(25,7);
    WRITELN('<1> CREATE LINKED LIST ');
    GOTOXY(25,8);
    WRITELN('<2> DISPLAY LINKED LIST ');
    GOTOXY(25,9);
    WRITELN('<3> INSERT NODE ');
    GOTOXY(25,10);
    WRITELN('<4> DELETE NODE ');
    GOTOXY(25,11);
    WRITELN('<5> SEARCH LINKED LIST ');
    GOTOXY(25,12);
    WRITELN('<6> QUIT ');
    GOTOXY(25,14);
    WRITE ('SELECT <1-6>: ');READLN(CHOICE);
    CASE CHOICE OF
      '1' : CREATE (HEAD);
      '2' : PRINTLIST (HEAD);
      '3' : INS_MENU;
      '4' : DELETE(HEAD);
      '5' : SEARCH (HEAD)
    END;
  UNTIL CHOICE='6';
  CLRSCR;
END.

```

ตัวอย่างการทำงานของโปรแกรม

โปรแกรมปรากฏเมนูให้เลือกงาน ดังนี้

```
< LINKED LIST DEMO >

<1> CREATE LINKED LIST
<2> DISPLAY LINKED LIST
<3> INSERT NODE
<4> DELETE NODE
<5> SEARCH LINKED LIST
<6> QUIT
SELECT<1-6>:_
```

ตัวอย่างการทำงานของเมื่อเลือกงาน <1> CREATE LINKED LIST

*** CREATE LINKED LIST ***

ENTER ELEMENT VALUE [* FOR END]

CAT↵

RAT↵

TIGER↵

*↵

ตัวอย่างการทำงานเมื่อเลือกงาน <2> DISPLAY LINKED LIST

*** DISPLAY LINKED LIST ***

-->[CAT]-->[RAT]-->[TIGER]

Press Any Key...↵

ตัวอย่างการทำงานเมื่อเลือกงาน <3> INSERT NODE ปรากฏเมนูย่อยดังนี้

INSERT A NODE AT SPECIFIED POSITION

<1> INSERT_HEAD

<2> INSERT_AFTER_SPECIFIED_KEY

<3> INSERT_TAIL

SELECT<1-3>:_

ตัวอย่างการทำงานเมื่อเลือกงาน <1> INSERT_HEAD จากเมนูย่อย

*** INSERT NEW HEAD NODE ***

ENTER NEW ELEMENT VALUE:BIRD<←

Already Inserted, Press Any Key...↵

ตัวอย่างการทำงานเมื่อเลือกงาน <3> INSERT_TAIL จากเมนูย่อย

*** INSERT NEW TAIL NODE ***

ENTER NEW ELEMENT VALUE: ZEBRA↵

Already Inserted, Press Any Key...↵

ตัวอย่างการทำงานเมื่อเลือกงาน <2> INSERT_AFTER_SPECIFIED_KEY จากเมนูย่อย

*** INSERT NEW NODE AFTER SPECIFIED KEY ***

=====

*** LINKED LIST BEFORE INSERTION ***

-->[BIRD]-->[CAT]-->[RAT]-->[TIGER]-->[ZEBRA]

Look at the linked list

Now, tell me you wish to enter new element after which node

Node which key = CAT↵

Ok! I found it, Now insert new element value

DOG↵

*** LINKED LIST AFTER INSERTION ***

-->[BIRD]-->[CAT]-->[DOG]-->[RAT]-->[TIGER]-->[ZEBRA]

Press Any Key...↵

ตัวอย่างการทำงานเมื่อเลือกงาน <4> DELETE NODE จากเมนูหลัก

*** LINKED LIST BEFORE DELETETION ***

-->[BIRD]-->[CAT]-->[RAT]-->[TIGER]-->[ZEBRA]

ENTER KEY OF NODE TO BE DELETED:TIGER↵

*** LINKED LIST AFTER DELETETION ***

-->[BIRD]-->[CAT]-->[RAT]-->[ZEBRA]

Press Any Key...↵

กรณีที่คำคีย์ที่ระบุไม่มีในรายการโยง จะปรากฏข้อความ

-->Not Found, Invalid Key...

ตัวอย่างการทำงานเมื่อเลือกงาน <5> SEARCH LINKED LIST จากเมนูหลัก

*** SEARCH LINKED LIST ***

*** DISPLAY LINKED LIST ***

-->[BIRD]-->[CAT]-->[RAT]-->[TIGER]-->[ZEBRA]

ENTER SEARCH KEY :TIGER↵

Found at Node # 4

Value of node 4 is TIGER

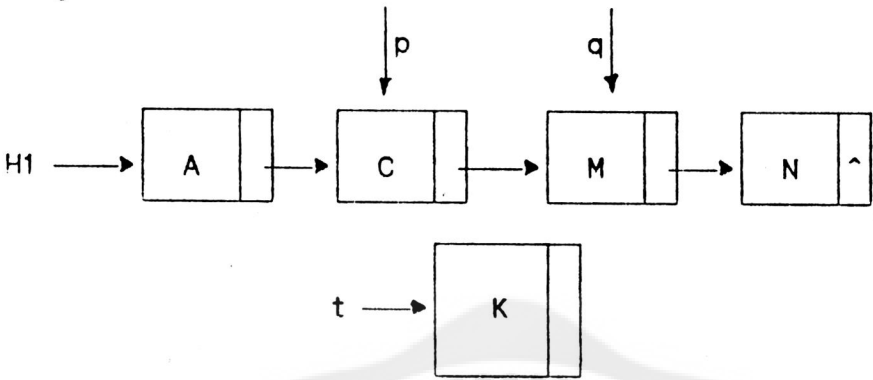
Press Any Key...↵

กรณีที่คำคีย์ที่ระบุไม่มีในรายการโยง จะปรากฏข้อความ

-->Not Found, Invalid Key...

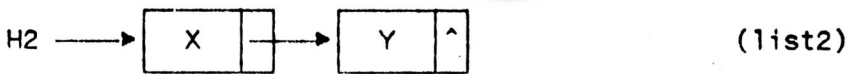
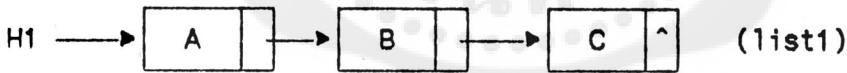
แบบฝึกหัด

1. จากรูป

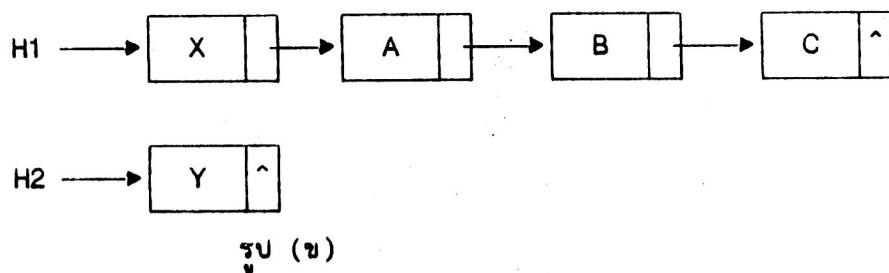


- (ก) เขียนคำสั่งใส่โหนด t เข้าในรายการโยงหลังโหนดที่ p ช้อยู่
- (ข) เขียนคำสั่งลบโหนดที่ q ช้อยู่

2. จากรูป (ก) ต้องการนำโหนดแรกออกจากรายการโยง list2 ไปใส่ไว้ต้นรายการโยง list1 ดังรูป (ข) เขียนคำสั่งสำหรับการทำางานดังกล่าว



รูป (ก)



3. กำหนดตัวแปร X : integer;
 Head, t : pointer;

จงวาดรูปจากการทำงานต่อไปนี้

```

Head := nil;
Read (x);
While x <> 999 do
  Begin
    new(t);
    t^.info := x;
    IF head = nil then
      Begin
        t^.link := nil;
        Head := t
      End
    Else
  
```

```

Begin
    t^.link := head;
    Head    := t
End;
Read (x)
End;

```

เมื่อข้อมูลที่ป้อนเข้ามาเป็น

10 20 30 999

4. (ก) วาดรูปแสดงการแทนคิวด้วยรายการโยงเดี่ยว
 (ข) เขียนโปรซีเจอร์สำหรับการใส่ข้อมูลเข้าในคิว เมื่อแทนโครงสร้างข้อมูลแบบคิวด้วยรายการโยง
 (ค) เขียนโปรซีเจอร์สำหรับการนำข้อมูลออกจากคิว เมื่อแทนโครงสร้างของคิวด้วยรายการโยง
5. ปรับปรุงรูปการแทนรายการโยงด้วยแถวลำดับในหัวข้อ 6.6 เมื่อมีการนำโหนดแรกของรายการโยง L2 ไปใส่เข้าเป็นโหนดแรกของรายการโยง L1 และลบโหนดสุดท้ายของ L2 แล้วคืนเป็นที่ว่างลงต้นรายการโยง AVAIL
6. (ก) วาดรูปแสดงการแทนสแตคด้วยรายการโยงเดี่ยว
 (ข) เขียนโปรซีเจอร์สำหรับใส่ข้อมูลลงในสแตคซึ่งแทนด้วยรายการโยงเดี่ยว
 (ค) เขียนโปรซีเจอร์สำหรับนำข้อมูลออกจากสแตค ซึ่งแทนด้วยรายการโยงเดี่ยว
7. (ก) วาดรูปแสดงการแทนสแตคด้วยรายการโยงคู่
 (ข) เขียนโปรซีเจอร์สำหรับใส่ข้อมูลลงในสแตคซึ่งแทนด้วยรายการโยงคู่
 (ค) เขียนโปรซีเจอร์สำหรับนำข้อมูลออกจากสแตค ซึ่งแทนด้วยรายการโยงคู่

8. เขียนโปรซีเจอร์ลบโหนดในรายการโยงเดี่ยว โดยลบโหนดที่อยู่หน้าโหนดที่ p ซ้ำๆ สมมติให้โหนดที่ p ซ้ำๆ ไม่ใช่โหนดแรกของรายการโยง
9. เขียนฟังก์ชันสำหรับนับจำนวนโหนดในรายการโยงเดี่ยว
10. วาดรูปแสดงการแทนด้วยวงกลมด้วยรายการโยงเดี่ยว
11. กำหนดโครงสร้างของโหนดเก็บข้อมูลนักศึกษา ซึ่งประกอบด้วยรหัสและคะแนน ดังนี้

```

Type pointer = ^node;
node = Record
    id      :string[5];
    score  :integer
End

```

เขียนโปรแกรมสำหรับสร้างรายการโยงเดี่ยว ซึ่งประกอบด้วยโหนดดังที่กำหนดไว้ข้างต้น โดยรับข้อมูลทางแป้นพิมพ์ และแสดงข้อมูลของรายการโยงดังกล่าวบนจอภาพ พร้อมทั้งพิมพ์สรุปจำนวนข้อมูลทั้งหมดในรายการโยง และคะแนนเฉลี่ย

12. เขียนโปรซีเจอร์สำหรับค้นคะแนนของนักศึกษาตามรหัสที่ผู้ใช้ระบุ จากรายการโยงในข้อ 11

บทที่ 7

โครงสร้างข้อมูลแบบต้นไม้

- หัวข้อ : 7.1 นิยามของต้นไม้
- 7.2 การแทนรูปต้นไม้
 - 7.3 ต้นไม้แบบทวิภาค
 - 7.4 การกำหนดโครงสร้างแบบต้นไม้
 - 7.5 ตัวอย่างการสร้างต้นไม้
 - 7.6 การแวะผ่านต้นไม้แบบทวิภาค
 - 7.7 ต้นไม้แบบค้นทวิภาค
 - 7.8 ต้นไม้แบบได้ดูล
 - 7.9 ตัวอย่างโปรแกรม

โครงสร้างข้อมูลแบบต่าง ๆ ที่กล่าวถึงมาก่อนหน้านี้ เป็นโครงสร้างข้อมูลแบบเชิงเส้น โครงสร้างข้อมูลแบบต้นไม้ (Tree) เป็นโครงสร้างข้อมูลแบบไม่เป็นเชิงเส้นแบบหนึ่งซึ่งเป็นโครงสร้างที่มีประโยชน์มากและถูกนำไปใช้อย่างกว้างขวาง

7.1 นิยามของต้นไม้

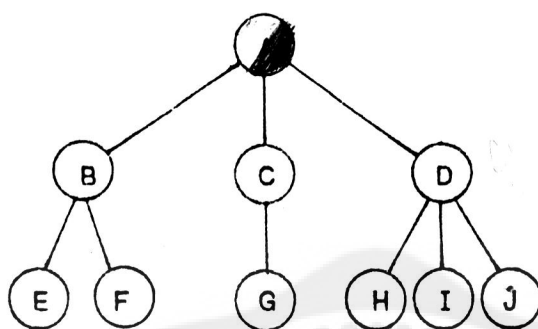
โครงสร้างข้อมูลแบบต้นไม้ กำหนดนิยามแบบเรียกซ้ำได้ดังรูป 7.1

ต้นไม้ ประกอบด้วยสมาชิกที่เรียกว่า โหนด ซึ่งมีสมบัติดังนี้

1. ว่าง (ไม่มีโหนดในต้นไม้) หรือ
2. มีโหนดหนึ่งถือว่าเป็น ราก (root) ส่วนที่เหลือเป็นต้นไม้ย่อย (Subtree) $T_1, T_2, \dots, T_k$ ($k \geq 0$) โดยต้นไม้ย่อยมีสมบัติเป็นต้นไม้เช่นกัน

รูป 7.1 นิยามของต้นไม้

โครงสร้างแบบต้นไม้แสดงความสัมพันธ์ระหว่างข้อมูลแบบเป็นลำดับชั้น (hierarchical relationship) โดยกำหนดความสัมพันธ์ระหว่างโหนดในระดับกับระดับถัดลงมาเป็นแบบพ่อกับลูก (father and son) ดังรูป 7.2



รูป 7.2 โครงสร้างข้อมูลแบบต้นไม้

โหนด A ซึ่งเป็นโหนดระดับบนสุด เรียกว่า ราก หรือ ราก (root node)

โหนด B,C,D ซึ่งอยู่ถัดลงมาจาก A เรียกว่า ลูกของโหนด A (children) หรือ direct descendant ของ A

ดังนั้นโหนด A เป็นพ่อของ B,C,D (father หรือ direct ancestor)

แต่ละโหนดในต้นไม้ที่อยู่ภายใต้รากเดียวกัน เรียกว่า โหนดพี่น้อง (sibling)

โหนดในระดับบนจะถือเป็น โหนดบน (ancestor หรือ predecessor) ของโหนดในระดับล่าง

โหนดในระดับล่างเรียกว่าเป็น โหนดล่าง (descendant หรือ successor) ของโหนดข้างบน

โหนดที่เป็นพ่อ-ลูกโดยตรง เรียกเป็น direct ancestor และ direct descendant

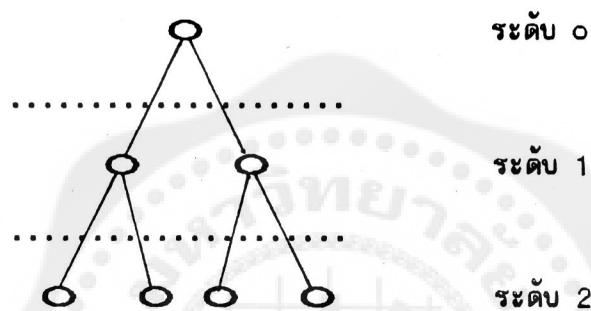
โหนดพี่น้อง (brother node) ได้แก่โหนดที่มีพ่อเดียวกัน เช่น B,C,D / E,F / H,I,J

กิ่งหรือเส้นเชื่อม (branch หรือ edge) ได้แก่เส้นเชื่อมระหว่างโหนดพ่อ-ลูก

ใบ (leaf node) ได้แก่ โหนดที่ไม่มีลูก เช่น E,F,G,H,I

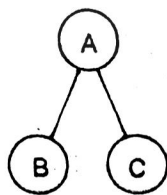
ดีกรีของโหนด (degree) ได้แก่ จำนวนลูกของโหนดนั้น เช่น ดีกรีของโหนด A เป็น 3, ดีกรีของโหนด B เป็น 2

ระดับของโหนด (level) เป็นการกำหนดค่าให้กับโหนดแต่ละระดับจากบนสุดคือราก เป็นระดับ 0, และระดับของลูกถัดลงไปเป็น 1,2,... ไปตามลำดับชั้น ดังรูป 7.3

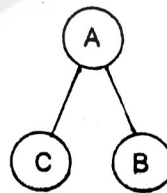


รูป 7.3 ระดับของต้นไม้

ความสูง (height หรือ dept) เป็นค่าที่วัดจากระดับสูงสุดของต้นไม้บวกด้วยหนึ่ง
ต้นไม้แบบอันดับ (ordered tree) เป็นต้นไม้ที่ถือว่าลำดับของลูกมีความหมาย เช่น

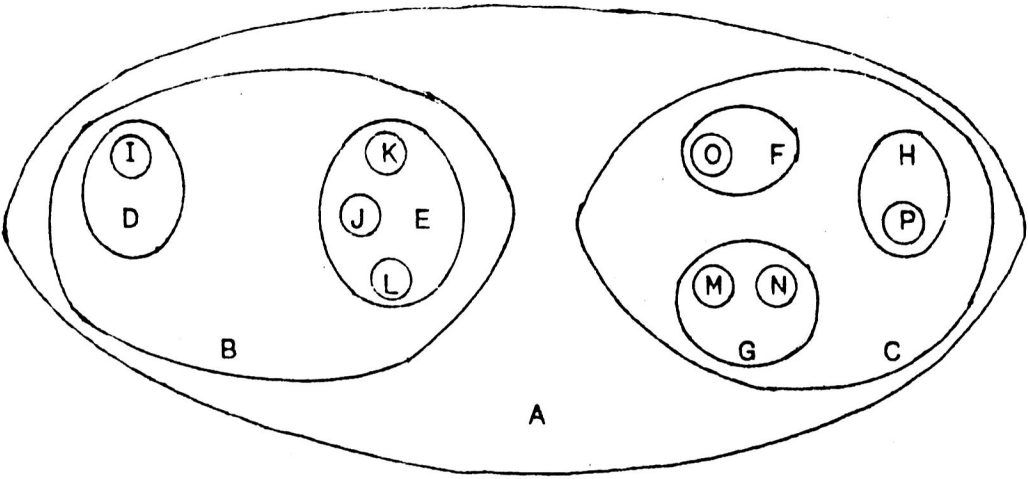


ต่างกับ



7.2 การแทนรูปต้นไม้

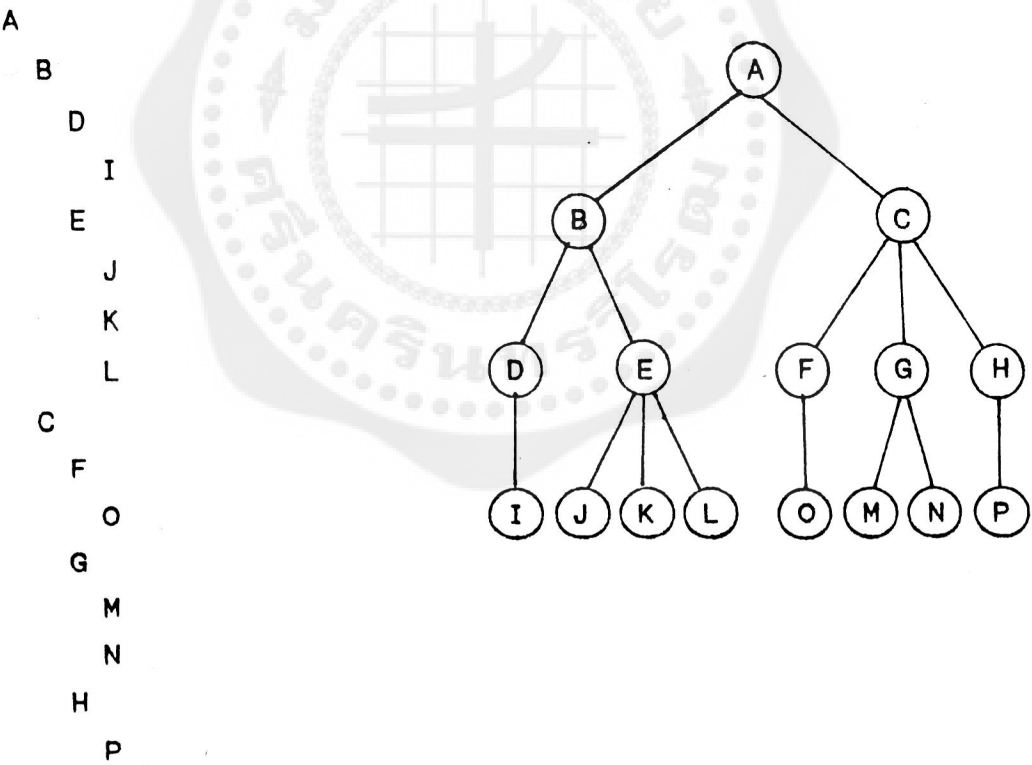
การแทนรูปของต้นไม้แทนได้หลายรูปแบบ เช่น ดังรูป 7.4



(ก) Nested Sets

$(A(B(D(I),E(J,K,L)),C(F(O),G(M,N),H(P))))$

(ข) วงเล็บซ้อน



(ค) การย่อหน้าเยื้องกัน(Indentation)

(ง) รูปกราฟ (Graph)

รูป 7.4 การแทนโครงสร้างของต้นไม้แบบต่าง ๆ

7.3 ต้นไม้แบบทวิภาค

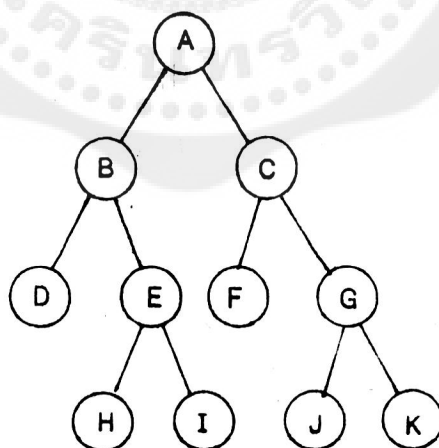
ต้นไม้แบบทวิภาค (Binary tree) เป็นต้นไม้ที่มีข้อจำกัดว่าแต่ละโหนดมีจำนวนลูกได้ไม่เกินสอง นิยามของต้นไม้แบบทวิภาคเป็นดังรูป 7.5

ต้นไม้แบบทวิภาค เป็นต้นไม้ที่มีสมบัติดังนี้

1. ว้าง หรือ
2. มีโหนดหนึ่งเป็นราก ที่เหลือแยกเป็นต้นไม้ย่อยซ้าย (left subtree) และต้นไม้ย่อยขวา (right subtree) โดยต้นไม้ย่อยทั้งสองมีสมบัติเป็นต้นไม้แบบทวิภาคด้วย

รูป 7.5 นิยามของต้นไม้แบบทวิภาค

ต้นไม้แบบทวิภาค เป็นต้นไม้แบบอันดับซึ่งมีดีกรีเท่ากับ 2 นั่นเอง ต้นไม้ซึ่งดีกรีมากกว่า 2 จะเป็นต้นไม้แบบหลายทาง (Multiway tree) ตัวอย่างต้นไม้แบบทวิภาค แสดงในรูป 7.6



รูป 7.6

ต้นไม้แบบทวิภาค

ต้นไม้แบบทวิภาคสมบูรณ์ (Complete Binary Tree) เป็นต้นไม้แบบทวิภาคที่ทุกโหนดที่ไม่ใช่ใบมีจำนวนลูกเท่ากับ 2 และจำนวนโหนดในระดับ L ใด ๆ ของต้นไม้เป็น 2^L ดังนั้น จำนวนโหนดทั้งหมดของต้นไม้แบบทวิภาคแบบสมบูรณ์ที่ ความสูงเป็น h คือ

$$n = 2^0 + 2^1 + 2^2 + \dots + 2^h = 2^{h+1} - 1$$

และความสูงของต้นไม้จะมีค่าเท่ากับ

$$h = \log_2 (n+1)$$

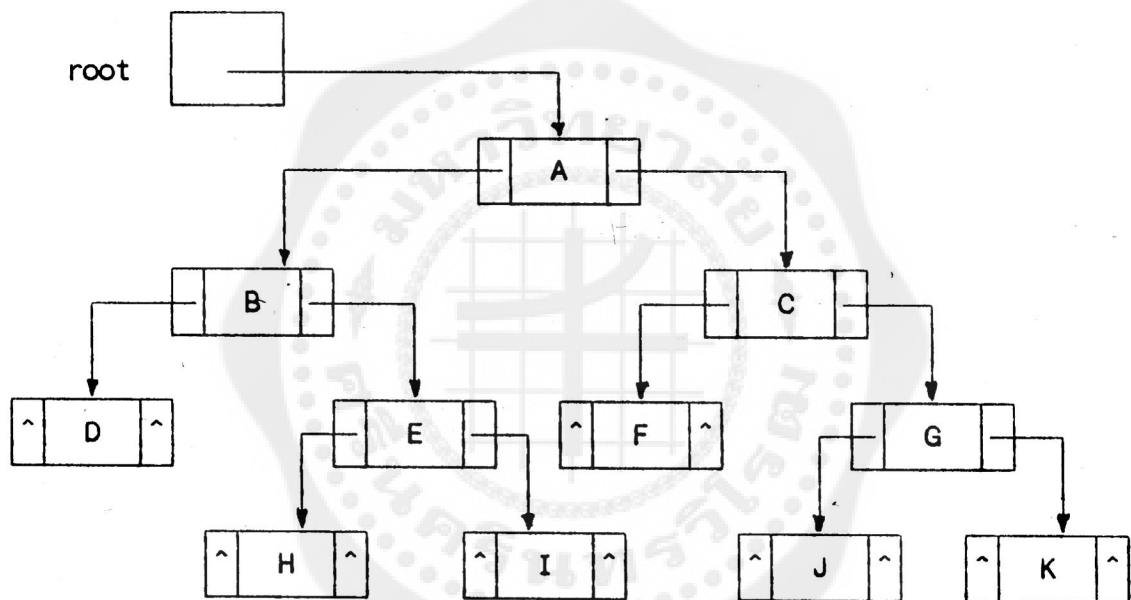
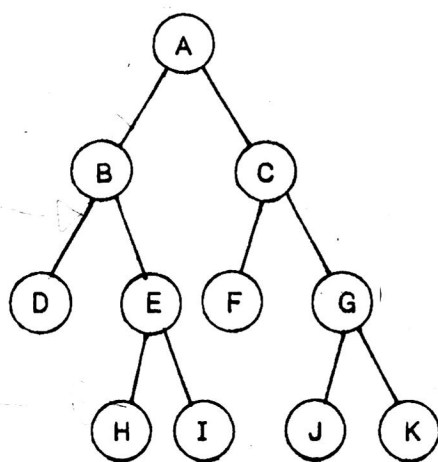
7.4 การกำหนดโครงสร้างแบบต้นไม้

กำหนดโครงสร้าง โหนดในต้นไม้แบบทวิภาค โดยใช้เนื้อที่แบบไดนามิกได้ดังรูป 7.7

```
Type  Itemtype = Char;
      Pointer  = ^ Node;
      Node     = Record
                  Info : Itemtype;
                  left, right : Pointer
      End;
```

รูป 7.7 การกำหนดโหนดในต้นไม้แบบทวิภาค

จากตัวอย่างต้นไม้แบบทวิภาคในรูป 7.6 จะเขียนโครงสร้างได้ดังรูป 7.8

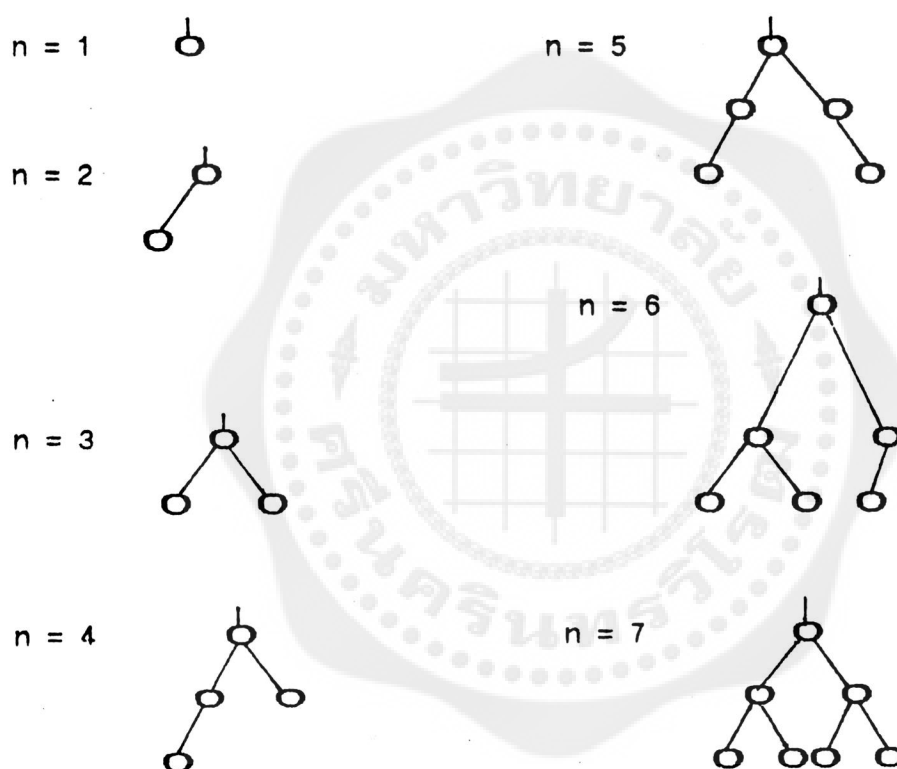


รูป 7.8 ตัวอย่างการแทนต้นไม้แบบทวิภาค

7.5 ตัวอย่างการสร้างต้นไม้

ในที่นี้จะแสดงตัวอย่างการสร้างต้นไม้ขนาด n โหนด เพื่อให้ได้ต้นไม้แบบทวิภาคที่มีความสูงน้อยที่สุด ซึ่งจะใช้หลักการของต้นไม้ได้ดุล (Balanced tree)

ต้นไม้จะเป็นต้นไม้ได้ดุล ถ้าจำนวนโหนดในต้นไม้ย่อยซ้ายและต้นไม้ย่อยขวาต่างกันสูงสุด 1 โหนด ดังตัวอย่างในรูป 7.9



รูป 7.9 ตัวอย่างต้นไม้ได้ดุล

วิธีการสร้างต้นไม้ได้ดุลขนาด n โหนด โดยใช้วิธีการแบบเรียกซ้ำ

1. ใช้หนึ่งโหนดเป็นราก
2. สร้างต้นไม้ย่อยซ้าย จำนวน n_l โหนด : $n_l = n \text{ div } 2$
3. สร้างต้นไม้ย่อยขวา จำนวน n_r โหนด : $n_r = n - n_l - 1$

ฟังก์ชันในรูป 7.10 แสดงขั้นตอนวิธีสร้างต้นไม้ได้ดุล

กำหนดโครงสร้างโหนดไว้ดังนี้

```
Type  Itemtype = integer;
      pointer   = ^node;
      node      = record
          key : itemtype;
          left,right : pointer
      end;
```

```
Function Bal_tree (n : integer):pointer;
Var  newnode : pointer;
     n1,nr : integer;
     x      :itemtype;
Begin {construct balanced tree with n nodes}
    if n = 0 then Bal_tree :=nil
    else begin n1:= div 2;
              nr := n-n1-1;
              Readln(x);
              new (newnode);
              with newnode ^ do
              begin  key := x;
                      left := Bal_tree (n1); {construct left subtree}
                      right:= Bal_tree (nr) {construct right subtree}
              end;
              Bal_tree := newnode;
              end;
    End; {Bal_tree}
```

รูป 7.10 ขั้นตอนวิธีสร้างต้นไม้ได้ดุล

7.6 การแหว่ม่านต้นไม้แบบทวรกาค

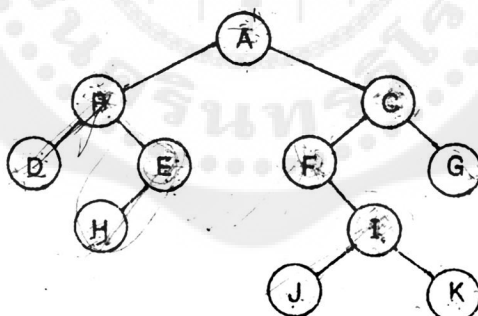
การแหว่ม่านต้นไม้แบบทวรกาค (Binary Tree Traversal) หมายถึงวธีการเข้าถึงข้อมูลแต่ละโหนดในต้นไม้แบบทวรกาคอย่างมีระเบียบแบบแผน

การแหว่ม่านต้นไม้แบบทวรกาค มี 3 วธี ดังนี้

1. วธีก่อนลำดับ (Preorder)
2. วธีตามลำดับ (Inorder)
3. วธีหลังลำดับ (วธีหลังลำดับ)

วธีการแหว่ม่านต้นไม้แบบทวรกาคทั้ง 3 แบบเป็นดังนี้

วธีก่อนลำดับ	วธีตามลำดับ	วธีหลังลำดับ
๑ เยี่ยมราก	๑ เยี่ยมต้นไม้ย่อยซ้าย	๑ เยี่ยมต้นไม้ย่อยซ้าย
๒ เยี่ยมต้นไม้ย่อยซ้าย	๒ เยี่ยมราก	๒ เยี่ยมต้นไม้ย่อยขวา
๓ เยี่ยมต้นไม้ย่อยขวา	๓ เยี่ยมต้นไม้ย่อยขวา	๓ เยี่ยมราก



รูป 7.11 ตัวอย่างต้นไม้แบบทวรกาค

ตัวอย่างการแหว่เยี่ยมโหนดในต้นไม้รูป 7.11 เป็นดังนี้

1. วธีก่อนลำดับ : A B D E H C F I J K G
2. วธีตามลำดับ : D B H E A F J I K C G
3. วธีหลังลำดับ : D H E B J K I F G C A

ขั้นตอนวิธีการแหว่ม่านต้นไม้วิธีก่อนลำดับ วิธีตามลำดับ และวิธีหลังลำดับ แสดงดังรูป 7.12, 7.13 และ 7.14 ส่วนการทำงานของ process (root) แทนการทำงานไต่กับโหนดที่เป็นรากของต้นไม้ย่อยนั้น และรูป 7.15 แสดงตัวอย่างการพิมพ์ข้อมูลในต้นไม้แบบทวิภาคด้วยวิธีตามลำดับ

```

Procedure Preorder (root : pointer);
begin
    if root <> nil then
    begin { *} Process (root);
        preorder (root^.left);
        preorder (root^.right)
    end
end; {Preorder}

```

รูป 7.12 ขั้นตอนวิธีการแหว่ม่านต้นไม้วิธีก่อนลำดับ

```

Procedure Inorder (root : pointer);
begin
    if root <> nil then
    begin Inorder (root^.left);
        { *} Process (root);
        Inorder (root^.right)
    end
end; {Inorder}

```

รูป 7.13 ขั้นตอนวิธีการแหว่ม่านต้นไม้วิธีตามลำดับ

```

Procedure Postorder (root : pointer);
begin
    if root <> nil then
        begin Postorder(root^.left);
              Postorder(root^.right);
              {*} Process (root)
        end;
    end; {Postorder}

```

รูป 7.14 ขั้นตอนวิธีการแหวะผ่านต้นไม้วิธีหลังลำดับ

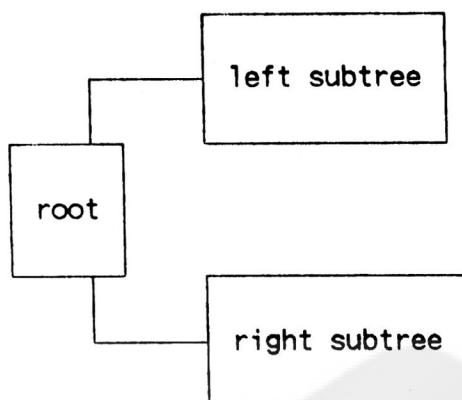
```

Procedure Printtree( t: pointer; n : integer);
Var      i : integer;      {n:indent level}
begin
    if t<> nil then
        begin
            Printtree (t^.left,n+1);
            for i:= 1 to n do write(' ');
            writeln (t^.data); {print root}
            Printtree (t^.right, n+1);
        end
    end; {printtree}

```

รูป 7.15 การพิมพ์ข้อมูลในต้นไม้แบบทวิภาคด้วยวิธีตามลำดับ

การพิมพ์ข้อมูลของต้นไม้ตามขั้นตอนวิธีในรูป 7.15 จะพิมพ์ในรูปแบบดังนี้



คำสั่งเรียกใช้จะส่งพารามิเตอร์ 2 ตัว ตัวแรกเป็นตัวชี้รากของต้นไม้และตัวที่สองเป็นค่าของระดับการเยื้อง ซึ่งจะเพิ่มขึ้นตามระดับของต้นไม้ การเรียกใช้จากโปรแกรมหลักอาจกำหนดระดับการเยื้องเป็น 1 หรือ 0 เช่น `printtree (root,0) {n=0}`

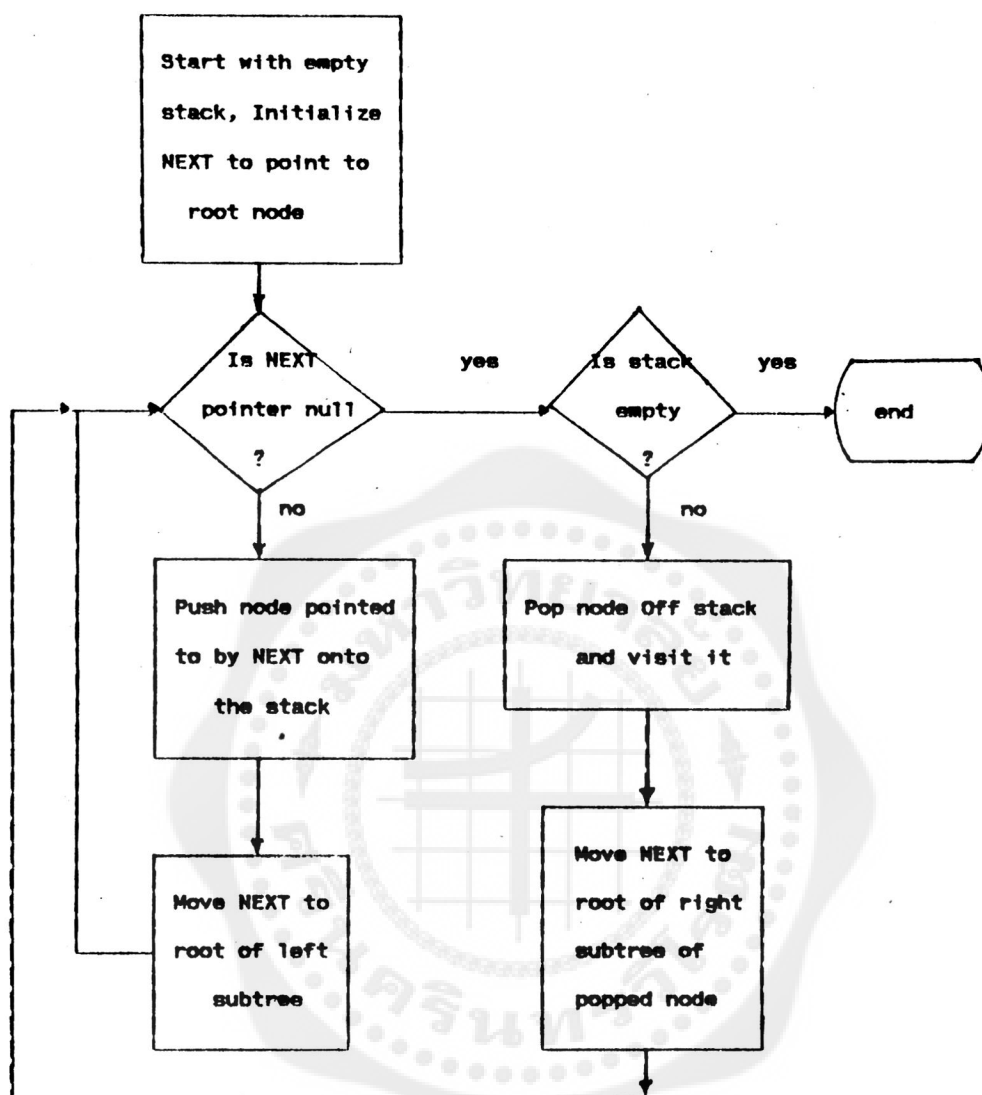
การแวะผ่านต้นไม้ตามขั้นตอนวิธีข้างต้นเป็นวิธีแบบเรียกซ้ำ จะเห็นว่าการเขียนขั้นตอนวิธีดังกล่าวจะสั้นและตรงตามนิยามที่กำหนดไว้ กรณีที่เขียนขั้นตอนวิธีแวะผ่านต้นไม้โดยไม่ใช้วิธีเรียกซ้ำก็สามารถกระทำได้ โดยต้องกำหนดสแตกขึ้นมาช่วยเก็บตำแหน่งของโหนดที่แวะผ่าน ในขณะที่ยังไม่เสร็จสิ้น ขั้นตอนวิธีแวะผ่านต้นไม้ที่กล่าวถึงวิธีตามลำดับที่ไม่ใช้วิธีเรียกซ้ำทำ แสดงด้วยผังงานได้ดังรูป 7.16 และเขียนเป็นโปรแกรมได้ดังรูป 7.17 โดยแทนสแตกด้วยแถวลำดับที่เก็บตัวชี้โหนด เขียนการประกาศสแตกไว้ดังนี้

```
Type Stackstruc = Record
```

```
Stack : Array[1..500] of pointer;
```

```
top   : 0..500
```

```
End;
```



รูป 7.16 มังงานแสดงขั้นตอนวิธีการแะผ่านต้นไม้ทวิภาคด้วยวิธีตามลำดับ
แบบไม่ใช้วิธีเรียกซ้ำ

```

Procedure Inorder(Root:Pointer);
Type Stackstruc = Record
    Stack : Array[1..500] of pointer;
    top   : 0..500
End;
Var Next :Pointer;
    Goflag:Boolean;
    S     :Stackstruc;
Begin
    S.top := 0;
    Next := Root; {start at root}
    Goflag := True;
    While Goflag Do
        Begin {build stack of pointers to nodes}
            {stop when encounter null left pointer}
            While (Next <> nil) Do
                Begin
                    S.top := S.top + 1;
                    S.Stack[S.top] := Next;
                    Next := Next^.left
                End;
            {visit node on top of stack}
            If (S.top > 0) Then
                Begin Next := S.Stack[S.top];
                    S.top := S.top - 1;
                    Writeln(Next^.Info);
                    Next := Next^.right
                End
            Else Goflag := False; {stack is empty}
        End;
    End;
End;

```

รูป 7.17 ขั้นตอนวิธีการแฉะผ่านต้นไม้ทวิภาคด้วยวิธีตามลำดับแบบไม่ใช้วิธีเรียกซ้ำ

7.7 ต้นไม้แบบค้นหา

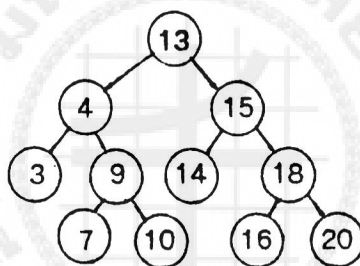
ต้นไม้แบบค้นหา (Binary Search Tree) เป็นต้นไม้แบบทวิภาคซึ่งมีการกำหนดตำแหน่งของโหนดในต้นไม้ตามลำดับค่าคีย์ของโหนด ซึ่งจะช่วยให้การเข้าถึงข้อมูลได้อย่างรวดเร็ว

7.7.1 นิยามของต้นไม้แบบทวิภาค

กำหนดนิยามของต้นไม้แบบค้นหาได้ดังรูป 7.18

ต้นไม้แบบค้นหา เป็นต้นไม้แบบทวิภาคซึ่งค่าคีย์ของโหนดในต้นไม้ย่อยซ้าย มีค่าน้อยกว่าค่าคีย์ของโหนด และค่าคีย์ของโหนดในต้นไม้ย่อยขวามีค่ามากกว่าหรือเท่ากับค่าคีย์ของราก

รูป 7.18 นิยามของต้นไม้แบบค้นหา



รูป 7.19 ตัวอย่างต้นไม้แบบค้นหา

การเข้าถึงข้อมูลในต้นไม้แบบค้นหา ใช้หลักการเปรียบเทียบค่าคีย์ที่ต้องการค้นหา กับค่าคีย์ของรากเสมอ โดยตรวจสอบเงื่อนไขต่อไปนี้

- ถ้าคีย์ที่ต้องการค้นหา เท่ากับ คีย์ของราก แสดงว่าพบข้อมูลที่ต้องการค้นหาแล้ว
- ถ้าคีย์ที่ต้องการค้นหา น้อยกว่า คีย์ของราก จะไปค้นหาต่อในต้นไม้ย่อยซ้าย
- ถ้าคีย์ที่ต้องการค้นหา มากกว่า ค่าคีย์ของราก จะไปค้นหาต่อในต้นไม้ย่อยขวา

ตัวอย่างเช่น ต้องการค้นหาว่ามีโหนดที่มีค่าคีย์เท่ากับ 7 หรือไม่จากต้นไม้ในรูป 7.19 จะเห็นว่าทำการเปรียบเทียบเพียง 4 ครั้งเท่านั้น โดยเปรียบเทียบกับโหนด 13 , 4 , 9 และ 7 ซึ่งพบข้อมูลที่ต้องการค้นหา

ถ้าเป็นต้นไม้แบบค้นหาที่สมดุล (Balanced Binary Search Tree) จะประกันได้ว่าจำนวนครั้งของการเปรียบเทียบสำหรับการค้นข้อมูลจะไม่เกิน $\log_2 n$

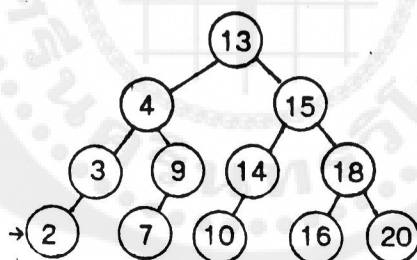
การแวะผ่านต้นไม้แบบค้นหาด้วยวิธีตามลำดับ จะได้ค่าคีย์ที่เรียงลำดับจากน้อยไปมาก เช่นการแวะผ่านต้นไม้ รูป 7.19 ด้วยวิธีตามลำดับจะได้ลำดับคีย์ดังนี้ 3, 4, 7, 9, 10, 13, 14, 15, 16, 18, 20

โหนดทางซ้ายสุดของต้นไม้แบบทวิภาค มีค่าน้อยที่สุด และโหนดทางขวาสุด จะเป็นโหนดที่มีค่ามากที่สุด

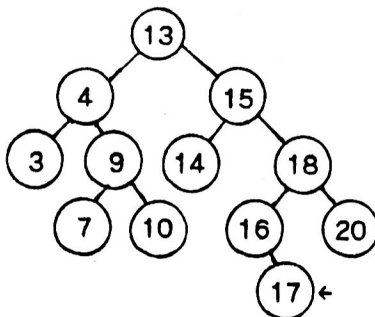
7.7.2 การใส่ข้อมูลเข้าในต้นไม้แบบทวิภาค

การใส่ข้อมูลเข้าในต้นไม้แบบทวิภาคจะใส่ข้อมูลลงในโหนดที่เป็นใบเสมอ ซึ่งต้องค้นหาตำแหน่งที่จะใส่ข้อมูลโดยการเปรียบเทียบคีย์กับคีย์ของราก จนกว่าจะพบตำแหน่งที่จะใส่ข้อมูลเพื่อรักษาสมบัติของต้นไม้แบบทวิภาค พิจารณาตัวอย่างการใส่ข้อมูลเข้าในต้นไม้ในรูป 7.19

ก. เพิ่มโหนดที่มีคีย์ = 2



ข. เพิ่มโหนดที่มีคีย์ = 17



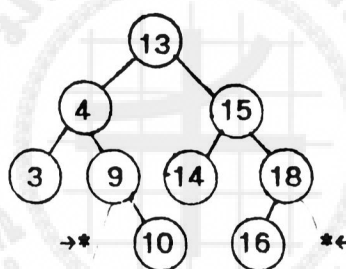
7.7.3 การนำข้อมูลออกจากต้นไม้แบบทวิภาค

การนำข้อมูลออกจากต้นไม้แบบทวิภาค ต้องพิจารณาตำแหน่งของ โหนดดังนี้

1. โหนดที่เป็นใบสามารถนำออกได้ทันที
2. โหนดที่เป็นรากที่มีต้นไม้ย่อยทางซ้ายหรือทางขวาเท่านั้น สามารถนำออกได้โดย ยกรากของต้นไม้ย่อยขึ้นมาแทนได้ทันที
3. โหนดที่เป็นรากที่มีลูกทั้งซ้ายและขวา จะต้องเลือกโหนดที่มีค่ามากที่สุดจากต้นไม้ย่อย ซ้าย หรือโหนดที่มีค่าน้อยที่สุดจากต้นไม้ย่อยขวา มาทดแทนเพื่อรักษาลำดับของค่าเสียไว้

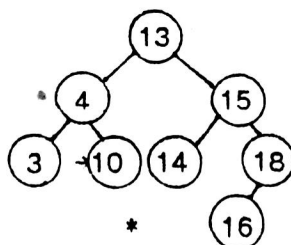
ตัวอย่างการนำโหนดจากต้นไม้รูป 7.17

- (1) โหนดที่นำออกไปคือ 7 และ 20 เป็นใบสามารถนำออกไปได้ทันที



หมายเหตุ ใช้สัญลักษณ์ * แทนตำแหน่งที่นำข้อมูลออกไป

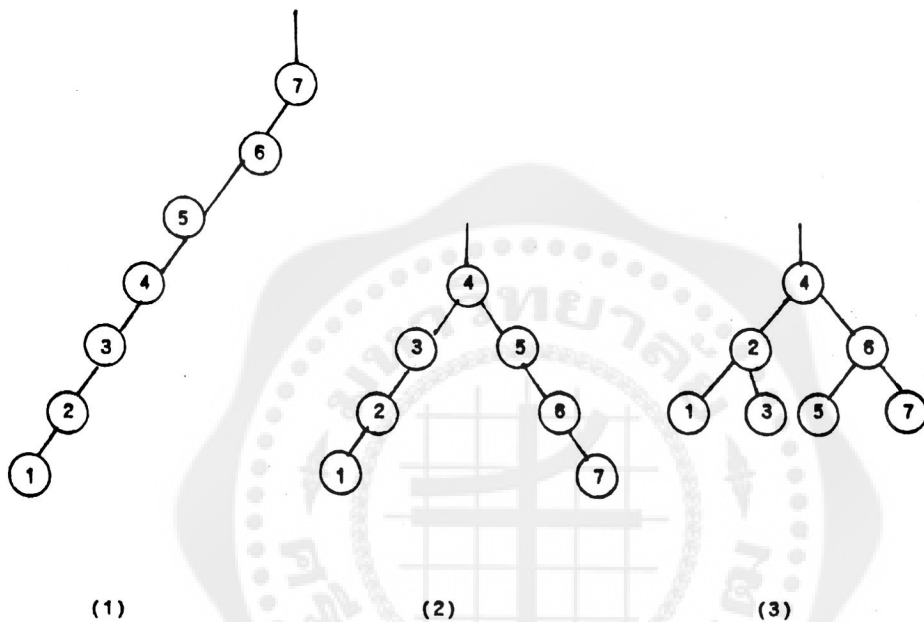
- (2) นำโหนด 9 ออก มีต้นไม้ย่อยทางขวาเท่านั้น ยก 10 มาแทนที่ 9 ได้ทันที



7.8 ต้นไม้ได้ดุล

7.8.1 นิยามของต้นไม้ได้ดุล

พิจารณาต้นไม้แบบค้นหาที่เก็บข้อมูลเหมือนกัน แต่รูปร่างต่างกัน ดังรูป 7.23

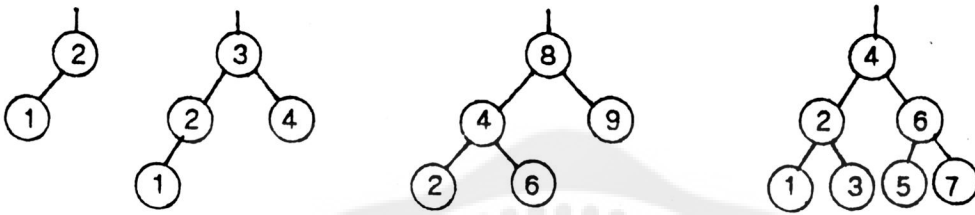


รูป 7.23 ต้นไม้แบบค้นหาที่เก็บข้อมูลเหมือนกันแต่รูปร่างต่างกัน

รูปร่างของต้นไม้แบบค้นหาที่มีความสูงแตกต่างกัน มีผลต่อความเร็วในการเข้าถึงข้อมูลอย่างไร ดังที่ได้กล่าวแล้วว่าต้นไม้แบบค้นหาที่มีโครงสร้างที่สมดุลจะรับประกันได้ว่า จำนวนครั้งของการเปรียบเทียบในการค้นข้อมูลจะไม่เกิน $\log_2 n$ ท่านตอบได้หรือไม่ว่าต้นไม้ทั้ง 3 รูปข้างต้นมีประสิทธิภาพในการค้นข้อมูลแตกต่างกันอย่างไร ตามปกติต้นไม้แบบค้นหาที่มีรูปร่างเบี่ยงเบนไปได้ตลอดเวลาที่มีการใส่ข้อมูลเข้าและนำข้อมูลออกจากต้นไม้ นั้น ทำให้ประสิทธิภาพในการค้นข้อมูลไม่แน่นอน การจัดโครงสร้างของต้นไม้ให้ได้ดีจะทำให้สามารถประกันถึงประสิทธิภาพในการเข้าถึงข้อมูลของต้นไม้ นั้น ต้นไม้แบบค้นหาที่มีโครงสร้างสมดุลเรียกว่า ต้นไม้ได้ดุล (Balanced Tree) หรือเรียกตามชื่อของผู้คิดค้นคือ Adelson-Velskii และ Landis ว่า ต้นไม้แบบเอวีแอล (AVL-Tree) ซึ่งกำหนดนิยามไว้ดังรูป 7.24

ต้นไม้ได้ดุล หมายถึง ต้นไม้ที่แต่ละ โหนดในต้นไม้ นั้น มีความสูงของต้นไม้ย่อยซ้าย และ ความสูงของต้นไม้ย่อยขวาแตกต่างกันไม่เกินหนึ่ง

รูป 7.24 นิยามต้นไม้แบบได้ดุล



รูป 7.25 ตัวอย่างต้นไม้ได้ดุล

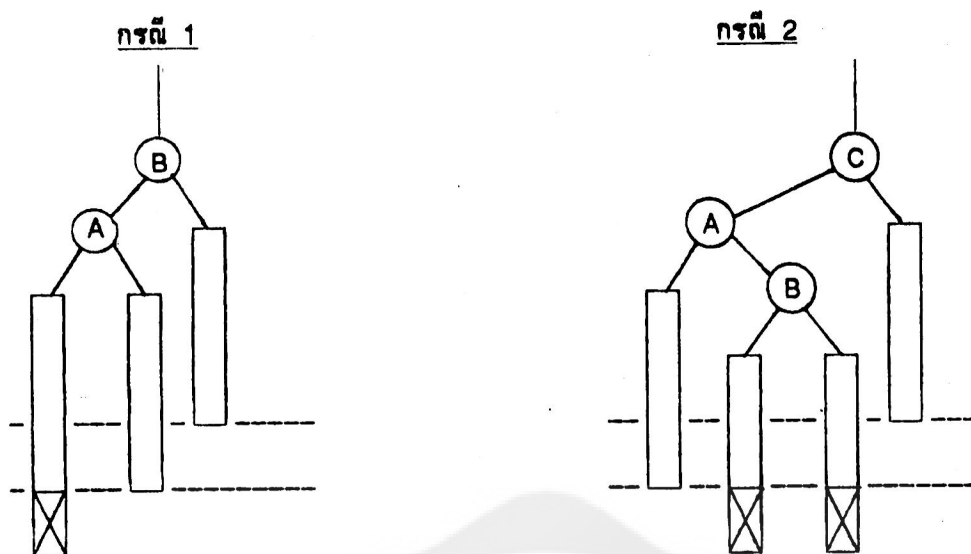
จากนิยามดังกล่าว ทำให้รับประกันได้ว่าประสิทธิภาพในการเข้าถึงข้อมูลในต้นไม้ได้ดุลนั้นจะ ไม่เกิน $O(\log_2 n)$

7.8.2 การปรับสมดุลของต้นไม้

การค้นข้อมูลในต้นไม้ได้ดุลใช้วิธีการเช่นเดียวกับการค้นข้อมูลในต้นไม้แบบค้นหวิภาค แต่กรณีที่เพิ่มข้อมูลและการลบข้อมูลทำให้เกิดการเสียสมดุลของต้นไม้ ก็จำเป็นต้องมีขั้นตอนวิธี สำหรับปรับสมดุลของต้นไม้

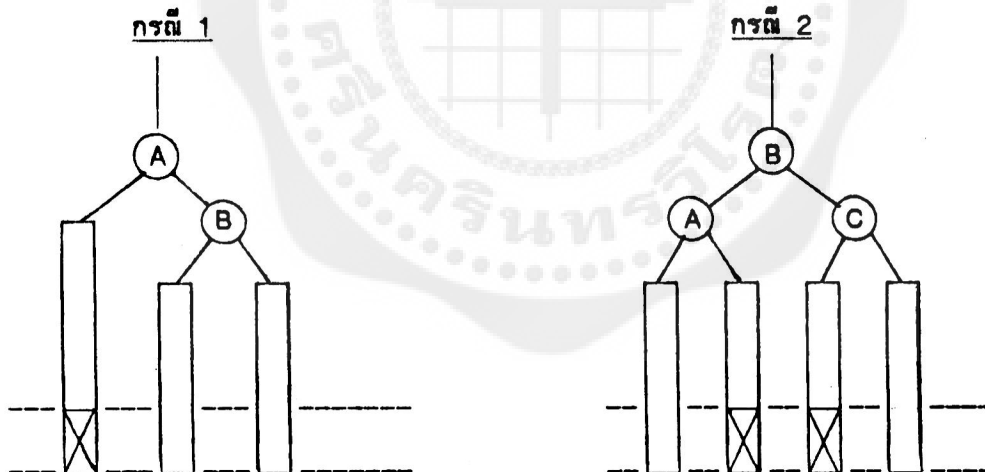
การปรับสมดุลเมื่อเพิ่มข้อมูล

วิธีการปรับสมดุลเมื่อเพิ่มข้อมูลจะขึ้นอยู่กับรูปร่างของต้นไม้ที่เปลี่ยนแปลงไป เมื่อมี การเพิ่มข้อมูล แบ่งเป็น 2 กรณีดังรูป 7.26



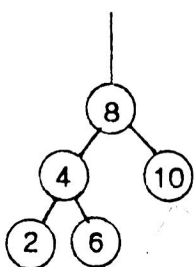
รูป 7.26 ต้นไม้ที่เสียสมดุลจากการใส่ข้อมูลเข้า

การปรับสมดุลของต้นไม้ทำได้โดยการปรับโครงสร้างใหม่ดังรูป 7.27

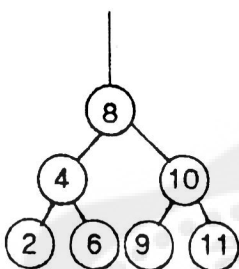


รูป 7.27 การปรับสมดุลของต้นไม้

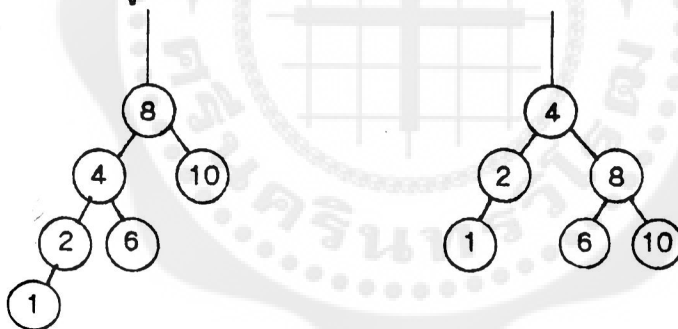
พิจารณาการปรับสมดุลของต้นไม้ในการใส่ข้อมูล จากรูป 7.28



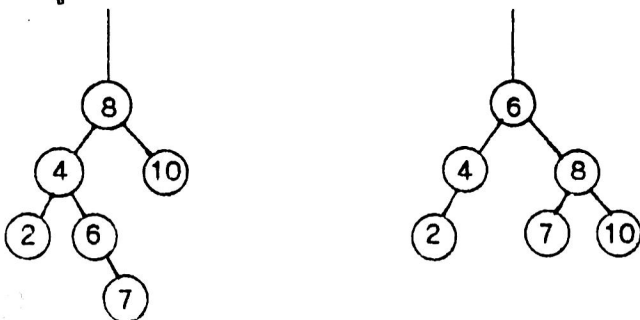
(1) เพิ่มโหนดซึ่งมีคีย์เป็น 9 และ 11 จะไม่ทำให้ต้นไม้เสียสมดุล ดังรูป



(2) เพิ่มโหนดซึ่งมีคีย์เป็น 1 จะทำให้ต้นไม้เสียสมดุลตามกรณีที่ 1 ต้องปรับโครงสร้างต้นไม้ใหม่ ดังรูป



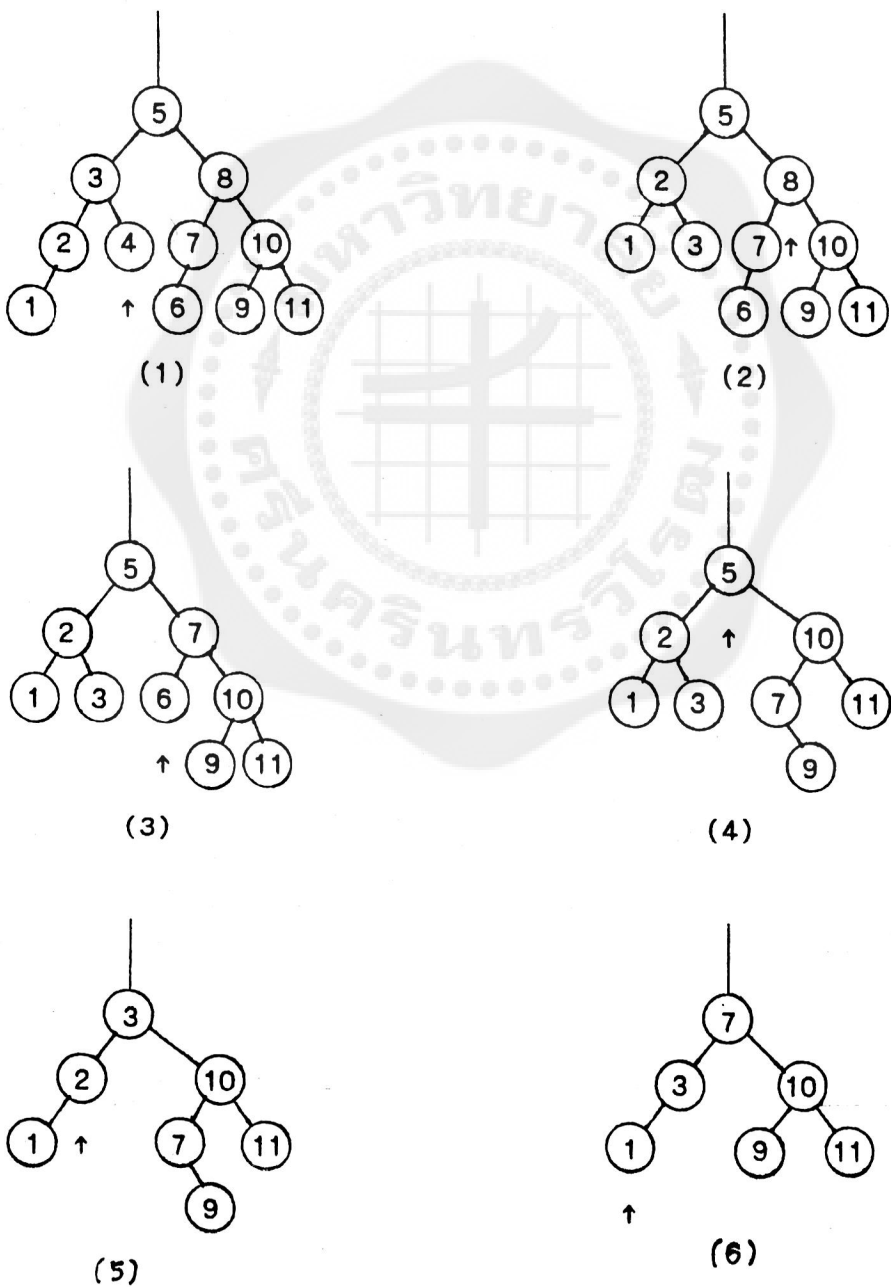
(3) เพิ่มโหนดซึ่งมีคีย์เป็น 7 จะทำให้ต้นไม้เสียสมดุลตามกรณีที่ 2 ต้องปรับโครงสร้างต้นไม้ใหม่ ดังรูป



รูป 7.28 การปรับสมดุลในการเพิ่มข้อมูลในต้นไม้ได้ดุล

การปรับสมดุลเมื่อนำข้อมูลออก

การนำข้อมูลก็ใช้วิธีการเช่นเดียวกับการนำข้อมูลออกในต้นไม้แบบค้นหวิภาค คือถ้า โหนดที่
นำออกเป็นใบก็ทำได้ทันที หรือถ้ามีต้นไม้ย่อยทางซ้ายหรือทางขวาเท่านั้น ก็นำออกแล้วนำโหนดราก
ของต้นไม้ย่อยมาทดแทน แต่ถ้า เป็นโหนดที่มีต้นไม้ย่อยทั้งทางซ้ายและขวาในที่นี้จะ เลือก โหนดขวาสุด
ของต้นไม้ย่อยซ้ายมาทดแทน แล้วทำการปรับสมดุลของต้นไม้ พิจารณาการนำข้อมูลออกจากต้นไม้
จากรูป 7.29 โดยทำการนำโหนดที่มีค่าคือเป็น 4,8,6,5,2,1 และ 7 ออกไปตามลำดับ





รูป 7.29 ตัวอย่างการปรับสมดุลของต้นไม้ได้ดุลเมื่อนำข้อมูลออก

7.9 ตัวอย่างโปรแกรม

ตัวอย่าง โปรแกรมต่อไปนี้ แสดงขั้นตอนวิธีการดำเนินการในต้นไม้แบบค้นหวิภาค

ได้แก่

- การสร้างต้นไม้แบบค้นหวิภาค
- การแสดงรูปของต้นไม้
- การใส่ข้อมูลเข้า

และ - การนำข้อมูลออก

โดยผู้ใช้สามารถกดเลือกการทำงานที่ต้องการได้จากเมนู โปรแกรมนี้เขียนและทดสอบการทำงานในเทอร์โบปาสกาล รุ่น 5.0 และ 6.0 การทดลองดำเนินการตามโปรแกรมนี้จะช่วยให้เข้าใจโครงสร้างข้อมูลแบบต้นไม้และการดำเนินการในต้นไม้มากขึ้น

```
PROGRAM BINARY_TREE_DEMO;
```

```
{ This program demonstrates the operations of the Binary Search Tree, }
{ such as, Create, Display tree on screen, Insert nodes, Delete nodes, }
{ and Search for specified node in tree. This program was writtten }
{ and tested on turbo pascal version 5.0 and 6.0 }
```

```
USES CRT;
TYPE DATAITEM = STRING[10];
      POINTER = ^NODE;
      NODE = RECORD
          LEFT :POINTER;
          DATA :DATAITEM;
          RIGHT :POINTER
      END;
VAR ROOT :POINTER;
    X :DATAITEM;
    CHOICE :CHAR;
```

```
PROCEDURE UNDERLINE;
```

```
BEGIN
```

```
Writeln('-----');
```

```
END;
```

```
PROCEDURE INSERT_NODE(VAR P:POINTER; X:DATAITEM);
```

```
BEGIN {insert new node in binary search tree }
```

```
IF P = NIL THEN
```

```
BEGIN
```

```
NEW(P);
```

```
P^.DATA := X;
```

```
P^.LEFT := NIL;
```

```
P^.RIGHT := NIL;
```

```
END
```

```
ELSE
```

```
IF P^.DATA <= X THEN
```

```
INSERT_NODE(P^.RIGHT,X)
```

```
ELSE
```

```
INSERT_NODE(P^.LEFT,X);
```

```
END;
```

```
PROCEDURE PRN_TREE(P:POINTER;INDENT:INTEGER);
```

```
{ Form of Binary Search Tree Printing }
```

```
{-----}
```

```
{ (Right Subtree) }
```

```
{ *(Root) }
```

```
{ (Left Subtree) }
```

```
{-----}
```

```
VAR I:INTEGER;
```

```
BEGIN
```

```
IF P <> NIL THEN
```

```
BEGIN
```

```
PRN_TREE(P^.RIGHT,INDENT+1);
```

```
IF INDENT = 0 THEN WRITE('*');
```

```
FOR I:=1 TO INDENT DO WRITE(' ':7);
```

```
Writeln('(',P^.DATA,')'); Writeln;
```

```

        PRN_TREE(P^.LEFT, INDENT+1);
    END
END;

PROCEDURE PREORDER(P:POINTER);
BEGIN
    IF P <> NIL THEN
        BEGIN
            WRITE(P^.DATA:6);
            PREORDER(P^.LEFT);
            PREORDER(P^.RIGHT)
        END
    END;
END;

PROCEDURE INORDER(P:POINTER);
BEGIN
    IF P <> NIL THEN
        BEGIN
            INORDER(P^.LEFT);
            WRITE(P^.DATA:6);
            INORDER(P^.RIGHT)
        END
    END;
END;

PROCEDURE POSTORDER(P:POINTER);
BEGIN
    IF P <> NIL THEN
        BEGIN
            POSTORDER(P^.LEFT);
            POSTORDER(P^.RIGHT);
            WRITE(P^.DATA:6);
        END;
    END;
END;

PROCEDURE PRINT_BST(VAR ROOT:POINTER);
VAR N :INTEGER;
    CH :CHAR;
BEGIN
    CLRSCR;
    WRITELN;
    WRITELN('  Form of  Binary Search Tree  Printing  ');
    WRITELN('-----');
    WRITELN('          (Right Subtree) ');
    WRITELN('      *(Root) ');
    WRITELN('          (Left Subtree) ');
    WRITELN('-----');
    WRITELN; WRITE('Press Any Key..');
    CH:=READKEY;WRITELN;
    CLRSCR; WRITELN;
    WRITELN('*** DISPLAY BINARY SEARCH TREE ***');
    UNDERLINE;
    PRN_TREE(ROOT,0);
    WRITELN;
    UNDERLINE;

```

```

WRITE('Press Any Key to See Traversing Order');
CH := READKEY; WRITELN;
CLRSCR;
UNDERLINE;
WRITELN('*** PREORDER TRAVERSAL ***');
PREORDER(ROOT);
WRITELN; UNDERLINE;
WRITELN;
WRITELN('*** INORDER TRAVERSAL ***');
INORDER(ROOT);
WRITELN; UNDERLINE;
WRITELN;
WRITELN('*** POSTORDER TRAVERSAL ***');
POSTORDER(ROOT);
WRITELN; UNDERLINE;
WRITELN;
WRITE('Press Any Key...');
CH := READKEY; WRITELN
END;

PROCEDURE CREATE_BST(VAR ROOT:POINTER);
VAR I:INTEGER;
BEGIN
  CLRSCR;
  ROOT := NIL; {INITIAL ROOT}
  WRITELN;
  WRITELN('*** CREATE BINARY SEARCH TREE ***'); WRITELN;
  WRITELN('ENTER DATA [ "*" FOR END ] ');
  UNDERLINE;
  READLN(X);
  WHILE X <> '*' DO
    BEGIN
      INSERT_NODE(ROOT,X);
      READLN(X);
    END;
  END;

PROCEDURE INSERT(VAR P:POINTER);
{ Insert new node in binary tree }
VAR X :DATAITEM;
    CH :CHAR;
BEGIN
  CLRSCR;
  WRITELN;
  WRITELN('*** INSERT NODE ***'); UNDERLINE;
  WRITE('ENTER DATA:'); READLN(X);
  INSERT_NODE(P,X);
  UNDERLINE; WRITELN;
  WRITE('—>Already Inserted, Press Any Key...');
  CH := READKEY; WRITELN
END;

PROCEDURE DELETE_NODE(VAR P:POINTER; X :DATAITEM);
VAR Q:POINTER;

```

```

PROCEDURE DEL(VAR R:POINTER);
{ Select the greatest data from left subtree to replace }
{ root node (deleted node) which pointed by Q }
BEGIN
  IF R^.RIGHT <> NIL THEN DEL(R^.RIGHT)
  ELSE {Assign value of Rightmost node of the left subtree }
       {to the node to be delete, ie. root, Deallocate the }
       {rightmost node. }
       BEGIN Q^.DATA := R^.DATA;
             Q := R;
             R := R^.LEFT
       END;
END; {DEL}

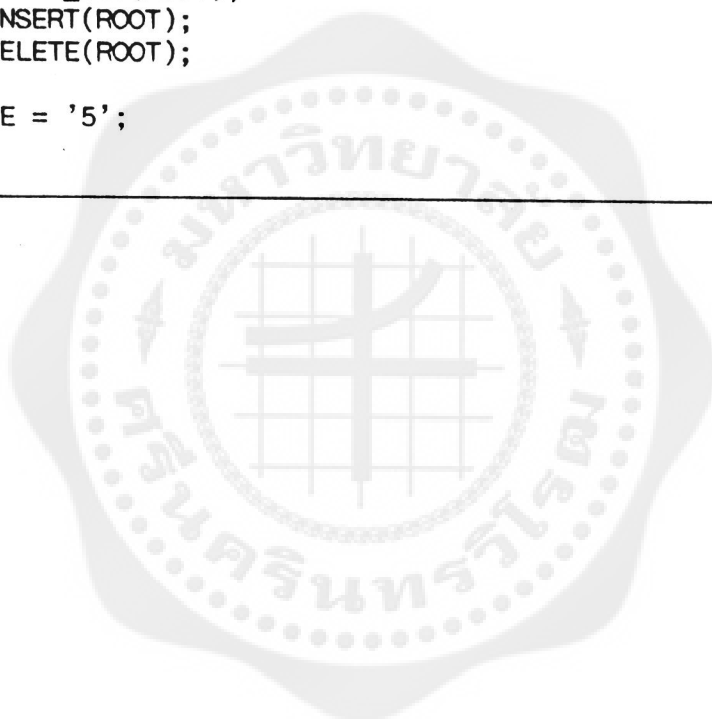
BEGIN
  IF P = NIL THEN {Not Found}
  BEGIN UNDERLINE; WRITELN;
        WRITELN(CHR(7),'--> Not Found,Invalid Key..');
  END
  ELSE
  BEGIN
    IF X < P^.DATA THEN DELETE_NODE(P^.LEFT,X)
    ELSE IF X > P^.DATA THEN DELETE_NODE(P^.RIGHT,X)
    ELSE { DELETE P^ }
    BEGIN Q := P;
          IF Q^.RIGHT = NIL THEN P := Q^.LEFT
          ELSE IF Q^.LEFT = NIL THEN P:= Q^.RIGHT
          ELSE {Delete node which has both subtrees}
               DEL(Q^.LEFT); {Replace by the rightmost node of
                               the left subtree }
          END;
    END;
  END;
END;

PROCEDURE DELETE(VAR ROOT:POINTER);
{ Delete node in binary search tree }
VAR KEY :DATAITEM;
    CH :CHAR;
BEGIN
  CLRSCR;
  WRITELN;
  WRITELN('*** DELETE NODE ***');UNDERLINE;
  WRITE('ENTER KEY OF NODE:');READLN(KEY);
  DELETE_NODE(ROOT,KEY);
  UNDERLINE; WRITELN;
  WRITE('Press Any Key...');
  CH := READKEY; WRITELN
END;

BEGIN {MAIN}
  CLRSCR;
  REPEAT
    CLRSCR;
    GOTOXY(25,5);

```

```
WRITELN('< BINARY SEARCH TREE DEMO >');
GOTOXY(25,7);
WRITELN('<1> CREATE BINARY SEARCH TREE ');
GOTOXY(25,8);
WRITELN('<2> DISPLAY BINARY SEARCH TREE ');
GOTOXY(25,9);
WRITELN('<3> INSERT NODE');
GOTOXY(25,10);
WRITELN('<4> DELETE NODE');
GOTOXY(25,11);
WRITELN('<5> QUIT ');
GOTOXY(25,13);
WRITE ('SELECT <1-5>:');READLN(CHOICE);
CASE CHOICE OF
  '1' : CREATE_BST(ROOT);
  '2' : PRINT_BST(ROOT);
  '3' : INSERT(ROOT);
  '4' : DELETE(ROOT);
END;
UNTIL CHOICE = '5';
END.{MAIN}
```



ตัวอย่างการทำงานของโปรแกรม BINARY_TREE_DEMO

การทำงานของโปรแกรมประกอบด้วย เมนูสำหรับเลือกรายการดังนี้

```
< BINARY SEARCH TREE DEMO >

<1> CREATE BINARY SEARCH TREE
<2> DISPLAY BINARY SEARCH TREE
<3> INSERT NODE
<4> DELETE NODE
<5> QUIT
SELECT <1-5>:_
```

ตัวอย่างการทำงานเมื่อเลือกรายการ <1> CREATE BINARY SEARCH TREE

*** CREATE BINARY SEARCH TREE ***

ENTER DATA ["*" FOR END]

6๔

8๔

9๔

3๔

2๔

*๔ {ใส่ข้อมูลแต่ละโหนดตามลำดับ จบด้วย *}

ตัวอย่างการทำงานเมื่อเลือกรายการ <2> DISPLAY BINARY SEARCH TREE

จะปรากฏคำอธิบายรูปแบบการแสดงผลในต้นไม้แบบทวิภาค ดังนี้

Form of Binary Search Tree Printing

(Right Subtree)

*(Root)

(Left Subtree)

Press Any Key...↵

เมื่อกดปุ่มใดๆ จะแสดงผลข้อมูลในต้นไม้แบบทวิภาคดังนี้

*** DISPLAY BINARY SEARCH TREE ***

(9)

(8)

*(6)

(3)

(2)

Press Any Key to see Traversing Order↵

เมื่อกดปุ่มใดๆ จะแสดงลำดับการแวะผ่านต้นไม้ด้วยวิธีก่อนลำดับ ตามลำดับ และหลังลำดับ ดังนี้

*** PREORDER TRAVERSAL ***

6 3 2 8 9

*** INORDER TRAVERSAL ***

2 3 6 8 9

*** POSTORDER TRAVERSAL ***

2 3 9 8 6

Press Any Key...

ตัวอย่างการทำงานเมื่อเลือกรายการ <3> INSERT NODE

*** INSERT NODE ***

ENTER DATA:7↵ {ผู้ใช้ใส่ข้อมูลสำหรับโหนดใหม่}

-->Already Inserted, Press Any Key...↵

ตัวอย่างการทำงานเมื่อเลือกรายการ <4> DELETE NODE

*** DELETE NODE ***

ENTER KEY OF NODE:3↵ {ผู้ใช้ใส่คีย์ของโหนดที่ต้องการลบออก}

Press Any Key...

ถ้าไม่พบข้อมูลที่ระบุ จะปรากฏข้อความว่า

-->Not Found,Invalid Key...

จากขั้นตอนดังกล่าวข้างต้นสามารถเรียกดูรูปของต้นไม้ ซึ่งจะปรากฏรูปแบบจอภาพดังนี้

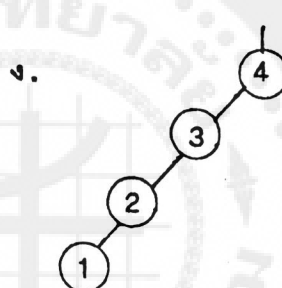
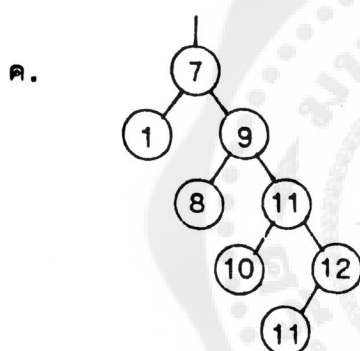
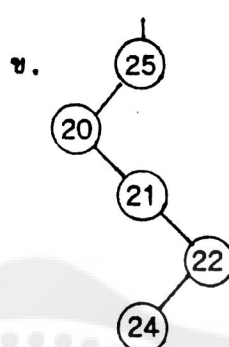
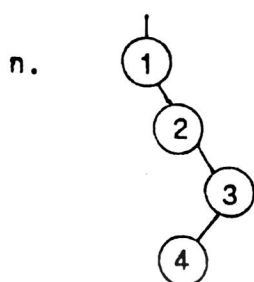
*** DISPLAY BINARY SEARCH TREE ***

(9)
(8)
(7)
*(6)
(2)

Press Any Key to see Traversing Order

แบบฝึกหัด

1. ต้นไม้ในรูปนี้ต้นไม้ใดเป็นต้นไม้แบบค้นหวิภาค และต้นไม้ใดไม่เป็นเพราะเหตุใด

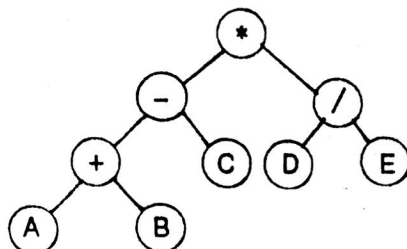


2. ในต้นไม้แบบหวิภาคใด ๆ

- ก. โหนดใดมีค่าน้อยที่สุด
- ข. โหนดใดมีค่ามากที่สุด
- ค. ค่าซ้ายของรากเป็นค่าน้อยที่สุดเมื่อใด
- ง. ค่าซ้ายของรากเป็นค่ามากที่สุดเมื่อใด

3. เขียนฟังก์ชัน Min และ Max เพื่อหาค่าข้อมูลทีน้อยสุดและมากที่สุดในต้นไม้แบบหวิภาค เมื่อค่าข้อมูลในโหนดเป็นค่าจำนวนเต็ม

4. แสดงลำดับของโหนดที่ถูกเข้าเยี่ยมสำหรับการแวะผ่านต้นไม้แบบทวิภาครูปต่อไปนี้ โดยใช้วิธีก่อนลำดับ, วิธีตามลำดับ และ วิธีหลังลำดับ



5. เขียนฟังก์ชันนับจำนวนโหนดในต้นไม้แบบทวิภาค
6. เขียนโปรซีเจอร์ Swap เพื่อทำการสลับโหนดทางซ้ายและขวาของต้นไม้แบบทวิภาค
7. เขียนโปรซีเจอร์เพื่อหาความสูงของต้นไม้แบบทวิภาค
8. เขียนฟังก์ชันเพื่อค้นข้อมูลในต้นไม้แบบทวิภาค เพื่อตรวจสอบว่ามีโหนดในต้นไม้ที่มีค่าคีย์ตามที่ระบุหรือไม่ ถ้าพบคืนค่าเป็นจริง ถ้าไม่พบคืนค่าเป็นเท็จ กำหนดหัวฟังก์ชันดังนี้
- ```
FUNCTION IF_IN_TREE(ROOT):BOOLEAN;
```
9. เปรียบเทียบประสิทธิภาพการค้นข้อมูลในโครงสร้างข้อมูลแบบรายการโยง ต้นไม้แบบทวิภาค ต้นไม้แบบทวิภาค และต้นไม้แบบทวิภาคที่ได้ผล

## บทที่ 8

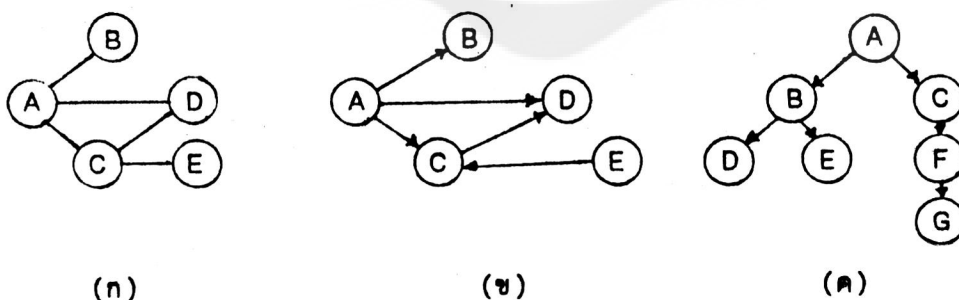
### โครงสร้างข้อมูลแบบกราฟ

- หัวข้อ :
- 8.1 กราฟและเครือข่าย
  - 8.2 การแทนโครงสร้างข้อมูลแบบกราฟ
  - 8.3 การแวะผ่านกราฟ
  - 8.4 การประยุกต์ของกราฟ

โครงสร้างข้อมูลแบบกราฟ (Graph) เป็นโครงสร้างแบบไม่เป็นเชิงเส้นอีกประเภทหนึ่งซึ่งสามารถแสดงความสัมพันธ์ระหว่างข้อมูลที่ซับซ้อนมากขึ้น เช่น เส้นทางคมนาคม ระหว่างเมืองต่าง ๆ เครือข่ายการสื่อสาร ลำดับกิจกรรมในโครงการ เครือข่ายการบิน เป็นต้น

#### 8.1 กราฟและเครือข่าย

กราฟ ประกอบด้วยเซตของโหนดและเส้นเชื่อมระหว่างโหนด (edge) การระบุเส้นเชื่อมระหว่างโหนดทำได้โดยการระบุโหนดที่ปลายทั้งสองด้านของเส้นเชื่อมนั้น ดังตัวอย่างรูป 8.1



รูป 8.1 ตัวอย่างกราฟ

กราฟรูป 8.1 (ก) ประกอบด้วย เซตของโหนดดังนี้  $\{A, B, C, D, E\}$  และ เซตของเส้นเชื่อมได้แก่  $\{(A, B), (A, D), (A, C), (C, D), (C, F)\}$

กราฟที่มีการกำหนดทิศทางของเส้นเชื่อม เรียกว่า direct graph หรือ digraph ดังรูป 8.1 (ข) ซึ่งใช้สัญลักษณ์แทนเส้นเชื่อมแบบมีทิศทางด้วยรูป  $\langle \text{โหนดต้นทาง}, \text{โหนดปลายทาง} \rangle$  เซตของเส้นเชื่อมของกราฟรูป 8.1(ข) ได้แก่  $\{\langle A, B \rangle, \langle A, D \rangle, \langle A, C \rangle, \langle C, D \rangle, \langle F, C \rangle\}$  โครงสร้างข้อมูลแบบต้นไม้ก็จัดอยู่ในโครงสร้างแบบกราฟด้วย ดังรูป 8.1(ค)

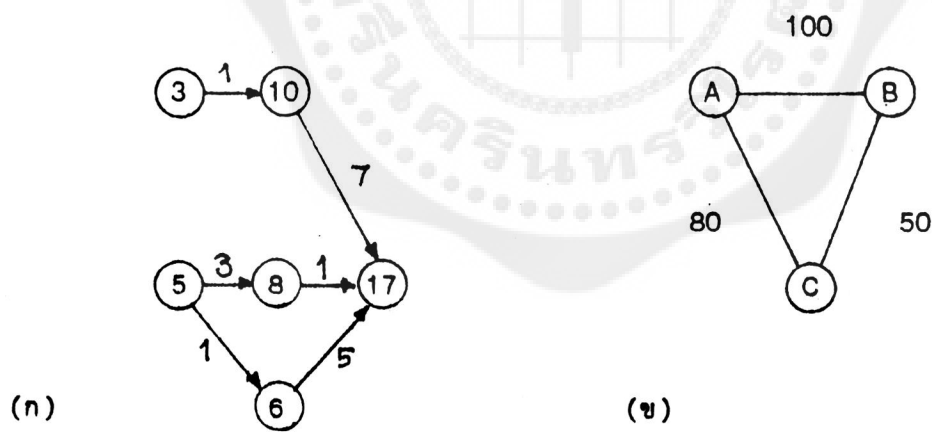
degree ของโหนด หมายถึง จำนวนเส้นเชื่อมที่ติดต่อกับโหนดนั้น

indegree ของโหนด หมายถึง จำนวนเส้นเชื่อมที่มีทิศทางพุ่งเข้าหาโหนดนั้น

outdegree ของโหนด หมายถึง จำนวนเส้นเชื่อมที่มีทิศทางพุ่งออกจากโหนดนั้น

โหนดคู่ที่มีเส้นเชื่อมระหว่างกัน เรียกว่า โหนดที่ adjacent กัน เช่น โหนด n adjacent กับโหนด m เมื่อมีเส้นเชื่อมระหว่างโหนด m และ n

กราฟที่มีตัวเลขระบุน้ำหนักกำหนดอยู่ที่เส้นเชื่อม เรียกว่า กราฟถ่วงน้ำหนัก (Weighted graph) หรือเครือข่าย (network) ซึ่งน้ำหนักเส้นเชื่อมเป็นสิ่งที่ใช้กำหนดความสัมพันธ์ระหว่างโหนดนั่นเอง ตัวอย่างดังรูป 8.2



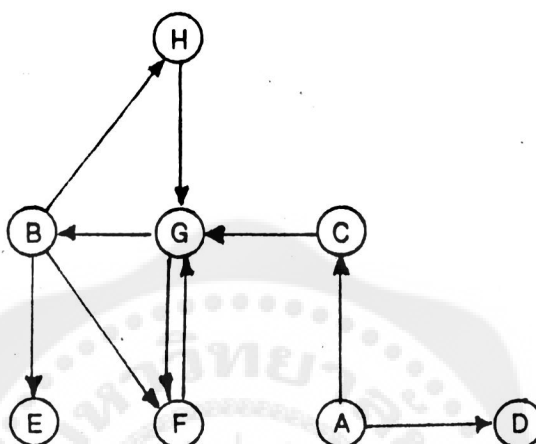
รูป 8.2 ตัวอย่างกราฟแสดงความสัมพันธ์ของข้อมูล

- (ก) น้ำหนักของโหนด = เศษเหลือจากการหารค่าโหนดปลายทางกับโหนดต้นทาง
- (ข) น้ำหนักของโหนด = ระยะทางระหว่างแต่ละโหนด

เส้นทางเชื่อมโหนดที่วนกลับมายังโหนดเดิมได้ เรียกว่า cycle

กราฟที่มี cycle เรียกว่า cyclic (กราฟที่ไม่มี cycle เรียกว่า acyclic)

พิจารณาจากรูป 8.3 มี cycle จากโหนด B ไป B, F ไป F และ H ไป H



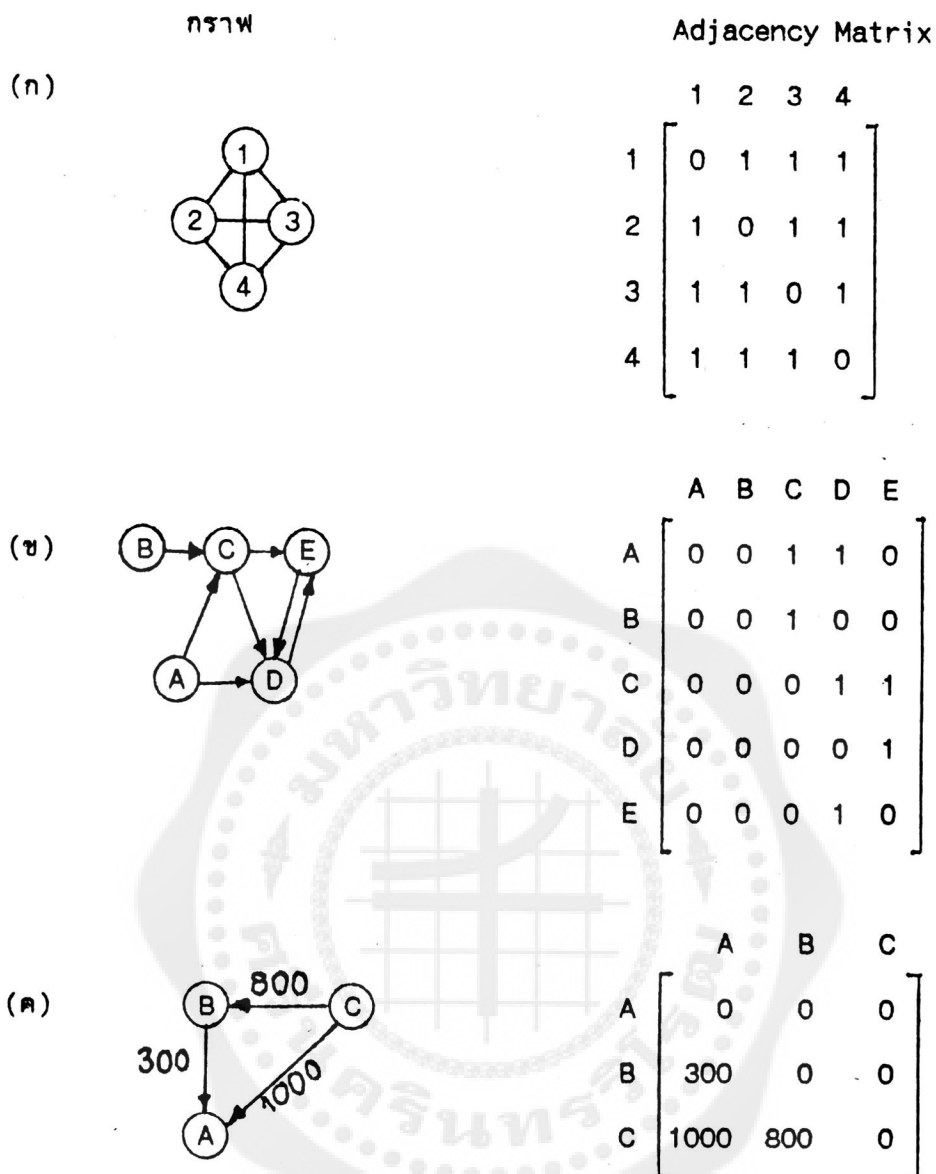
รูป 8.3 ตัวอย่างกราฟ

## 8.2 การแทนโครงสร้างข้อมูลแบบกราฟ

การแทนโครงสร้างแบบกราฟสามารถแทนได้หลายแบบในที่นี้จะกล่าวถึง การแทนด้วย adjacency matrix และ adjacency list

### 8.2.1 Adjacency Matrix

ใช้เมตริกซ์ 2 มิติ ขนาด  $n \times n$  เมื่อ  $n$  = จำนวนโหนด สำหรับการแทนรูปกราฟ เพื่อแสดงว่าโหนดแต่ละคู่ (ตามตำแหน่งแถว-สดมภ์) มีเส้นเชื่อมโยงหรือไม่ ถ้ามีให้แทนค่าข้อมูลในเมตริกซ์ตามตำแหน่งนั้นเป็น 1 ถ้าไม่มีเส้นเชื่อมโยงให้แทนค่าในเมตริกซ์เป็นตามตำแหน่งนั้นเป็น 0 ดังตัวอย่างรูป 8.4



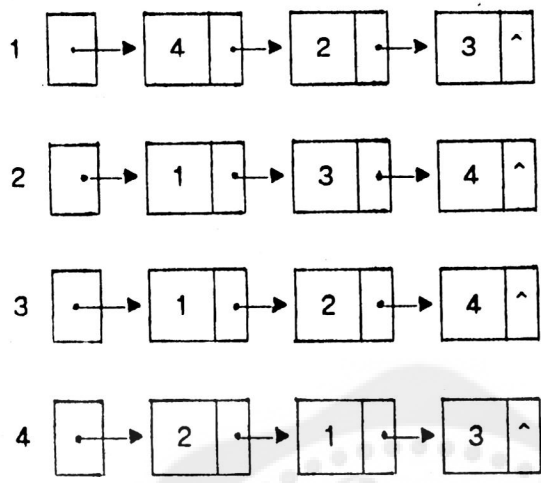
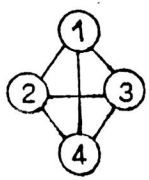
รูป 8.4 การแทนกราฟด้วย Adjacency Matrix

### 8.2.2 Adjacency List

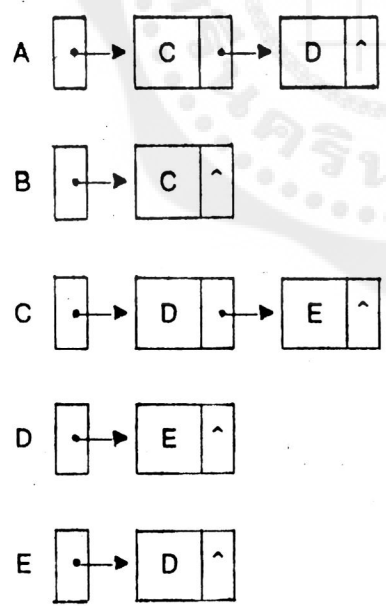
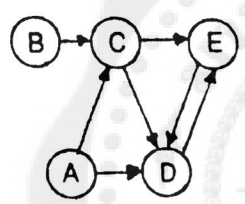
การแทนกราฟด้วย Adjacency List เป็นการใช้อันดับของรายการโยงแทนรูปกราฟ โดยประกอบด้วยรายการโยง  $n$  รายการ เมื่อ  $n$  = จำนวนโหนด สมาชิกในรายการโยงที่  $i$  แสดงถึงโหนดที่มีเส้นเชื่อมกับโหนดที่  $i$  ;  $i = 1..n$  ดังตัวอย่างรูป 8.5



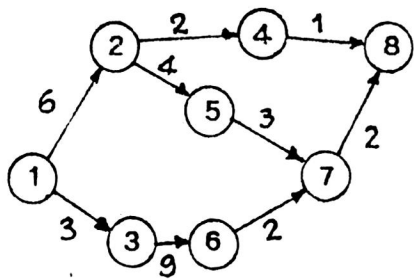
(ก)



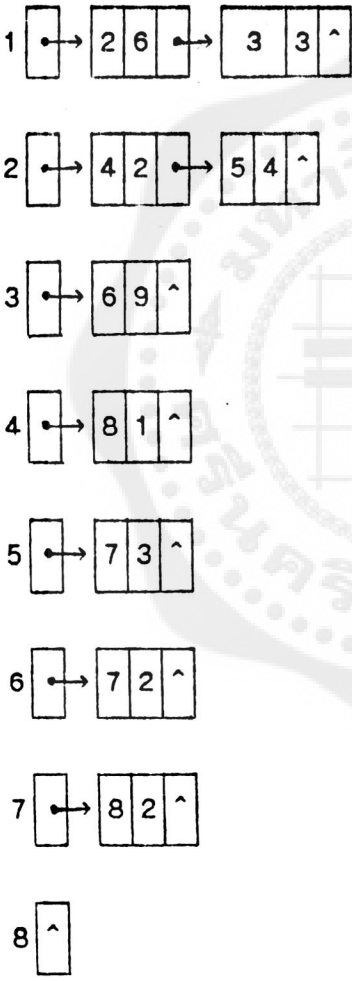
(ข)



รูป 8.5 ตัวอย่างการแทนกราฟแบบไม่ถ่วงน้ำหนักด้วย Adjacency List



แต่ละโหนดประกอบด้วย ชื่อโหนด น้ำหนักของเส้นเชื่อม และตัวชี้โหนดถัดไป



รูป 8.6 ตัวอย่างการแทนกราฟแบบถ่วงน้ำหนักด้วย Adjacency List

### 8.2.3 การกำหนดโครงสร้างข้อมูลแบบกราฟ

การกำหนดโครงสร้างข้อมูลแบบกราฟ จะขึ้นอยู่กับวิธีที่ใช้แทนกราฟว่าใช้

วิธีแบบ Adjacency Matrix หรือ Adjacency List

วิธีที่ 1 การแทนกราฟด้วย Adjacency Matrix

(ก) กราฟแบบไม่ถ่วงน้ำหนัก เขียนประกาศโครงสร้างของกราฟได้ดังรูป 8.7

```
Const Maxnodes = 50;
Type Nodeptr = 1..Maxnodes;
Adjmatrix = Array[nodeptr,nodeptr] of Boolean;
Graph = Adjmatrix;
```

รูป 8.7 การกำหนดโครงสร้างข้อมูลแบบกราฟด้วย Adjacency Matrix

(ข) กราฟแบบถ่วงน้ำหนัก เขียนประกาศโครงสร้างของกราฟได้ดังรูป 8.8

```
Const Maxnodes = 50;
Type Nodeptr = 1..Maxnodes;
Edge = Record
 Adj : Boolean;
 Weight : Integer;
End;
Weighted_graph = Array[nodeptr,nodeptr] of Edge;
Graph = Weighted_graph;
```

รูป 8.8 การกำหนดโครงสร้างข้อมูลแบบกราฟถ่วงน้ำหนักด้วย Adjacency Matrix

## วิธีที่ 2 การแทนกราฟด้วย Adjacency List

(ก) กราฟแบบไม่ถ่วงน้ำหนัก เขียนประกาศโครงสร้างของกราฟได้ดังรูป 8.9

```
Type Nodeid = 0..50;
 Edgeptr = ^ Node;
 Node = Record
 Id : Nodeid;
 Next : Edgeptr
 End;
 Graph = Array[nodeid] of Edgeptr;
```

รูป 8.9 การกำหนดโครงสร้างข้อมูลแบบกราฟด้วย Adjacency List

(ข) กราฟแบบถ่วงน้ำหนัก เมื่อแทนด้วย Adjacency List แต่ละโหนดต้องเพิ่มส่วนสำหรับเก็บน้ำหนัก เขียนประกาศได้ดังรูป 8.10

```
Type Nodeid = 0..50;
 Edgeptr = ^ Node;
 Node = Record
 Id : Nodeid;
 Weight : Integer;
 Next : Edgeptr
 End;
 Graph = Array[nodeid] of Edgeptr;
```

รูป 8.10 การกำหนดโครงสร้างข้อมูลแบบกราฟถ่วงน้ำหนักด้วย Adjacency List

### 8.3 การแหวผ่านกราฟ

การแหวผ่านกราฟ (Traverse Graph) เป็นการกำหนดลำดับในการเข้าถึงข้อมูลแต่ละโหนดของกราฟอย่างมีระเบียบแบบแผน

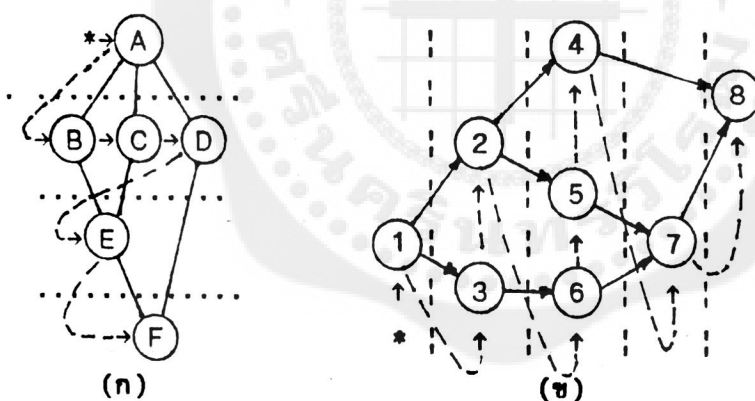
การแหวผ่านกราฟมี 2 วิธี ได้แก่

1. การค้นในแนวกว้าง
2. การค้นในแนวลึก

#### 8.3.1 การค้นในแนวกว้าง

การแหวผ่านต้นไม้ด้วยวิธีการค้นในแนวกว้าง (Breadth First Search) เป็นการเข้าถึงข้อมูลแต่ละโหนดในกราฟโดยกำหนดโหนดเริ่มต้น แล้วแหวเยี่ยมโหนดต่างๆทุกโหนดที่อยู่ระดับถัดมาที่เชื่อมโยงกับโหนดเริ่มต้น แล้วจึงเยี่ยมโหนดต่างๆที่เชื่อมโยงกับโหนดในระดับถัดรองลงมาตามลำดับ ซึ่งเป็นการแหวผ่านโหนดตามแนวกว้างของกราฟนั่นเอง

พิจารณาตัวอย่างการแหวผ่านกราฟด้วยวิธีการค้นในแนวกว้างในรูป 8.11

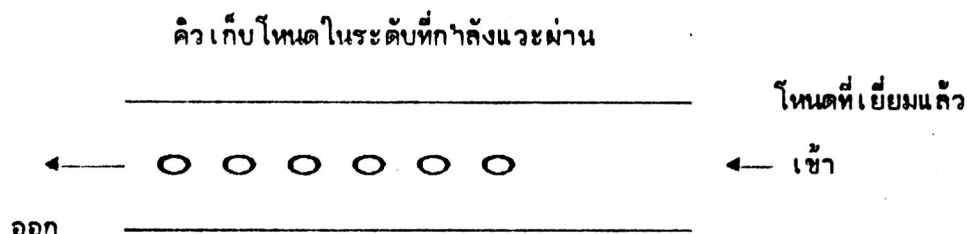


รูป 8.11 ตัวอย่างการแหวผ่านกราฟด้วยวิธีการค้นในแนวกว้าง

จากรูป 8.11 (ก) ลำดับของโหนดที่ถูกเข้าเยี่ยมคือ A,B,C,D,E,F หรือเป็น A,D,C,B,E,F ก็ได้เช่นกัน และจากรูป 8.11 (ข) ลำดับของโหนดที่ถูกเข้าเยี่ยมคือ 1,3,2,6,5,4,7,8 หรืออาจเป็น 1,2,3,4,5,6,7,8 ก็ถูกต้องเช่นกัน

### 8.3.2 ขั้นตอนวิธีการค้นในแนวกว้าง

ขั้นตอนวิธีการแหว่ผ่านกราฟด้วยวิธีการค้นในแนวกว้าง จะนำโครงสร้างแบบคิวมาช่วยเก็บโหนดตามระดับของกราฟที่กำลังแหว่ผ่าน โดยเก็บแต่ละโหนดที่ได้เข้าเยี่ยมแล้วเข้าในคิวตามลำดับ ดังรูป 8.12 ในที่นี้จะแทนกราฟด้วย Adjacency List



ไปหาโหนดที่จะเข้าเยี่ยมในระดับความลึกถัดลงไป

รูป 8.12 คิวที่ช่วยในการแหว่ผ่านกราฟด้วยวิธีค้นตามแนวกว้าง

เมื่อจบการแหว่ผ่านโหนดในแต่ละระดับก็จะนำข้อมูลออกจากคิว เพื่อไปหาโหนดในระดับถัดลงไปที่จะเข้าเยี่ยม แต่ละโหนดจะถูกเข้าเยี่ยมเพียงครั้งเดียวเท่านั้น การแหว่ผ่านกราฟจะสิ้นสุดเมื่อทุกโหนดถูกเข้าเยี่ยมหมดแล้ว และคิวจะต้องเป็นคิวงว้าง

วิธีการแหว่ผ่านกราฟด้วยวิธีค้นตามแนวกว้าง แสดงด้วยผังงานรูป 8.13 และต้องมีการเพิ่มการกำหนดข้อมูลในโครงสร้างของกราฟ เพื่อเป็นเครื่องหมายแสดงว่าโหนดนั้นถูกเข้าเยี่ยมแล้ว เพื่อป้องกันการเยี่ยมซ้ำ โดยกำหนดเป็นตัวบ่งชี้ดังนี้

mark : 0..1;

โดยถ้า mark = 0 แสดงว่าโหนดนั้นยังไม่ถูกเข้าเยี่ยม และถ้า mark = 1 แสดงว่าโหนดนั้นถูกเข้าเยี่ยมแล้ว

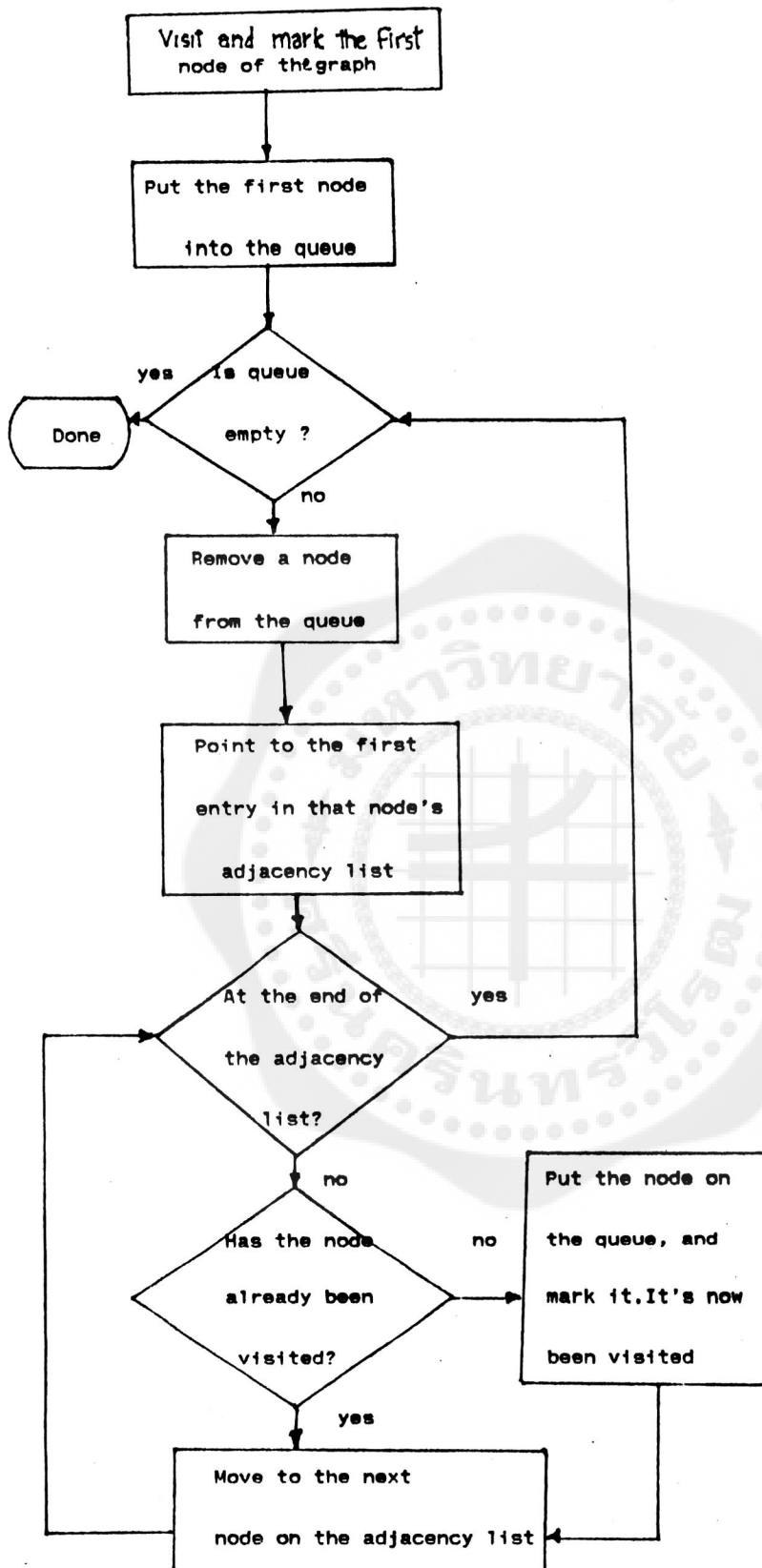
รูป 8.14 แสดงขั้นตอนวิธีแหว่ผ่านกราฟด้วยวิธีค้นตามแนวกว้างด้วยโปรซีเจอร์ BFS กำหนดโปรซีเจอร์ Insert\_Q สำหรับใส่ข้อมูลเข้าในคิว และฟังก์ชัน Delete\_Q สำหรับนำข้อมูลออกจากคิว และฟังก์ชัน Empty ตรวจสอบคิวงว้าง และกำหนดโครงสร้างของคิวไว้ดังนี้

```
Type QueueStruct = Record Queue :Array[1..100] of integer;
```

```
Front,Rear :Integer
```

```
End;
```

```
Var Queue:QueueStruct;
```



รูป 8.13 มังงานแสดงขั้นตอนวิธีการแหว่ม่านกราฟด้วยวิธีค้นตามแนวกว้าง

กำหนดโครงสร้างของกราฟ ซึ่งแทนด้วย Adjacency List ดังนี้

```

Const Maxnode = 50;

Type Nodeid = 0..Maxnode;

Edgeptr = ^ Edgeinfo;

Node = Record
 Mark :0..1;
 Adjlist :Edgeptr
End;

Edgeinfo = Record
 Id :Nodeid;
 Weight :Integer;
 Next :Edgeptr
End;

Graphtype = Array[1..Maxnode] of Node;

Var Graph :Graphtype;
 firstnode :Nodeid;

```

### 8.3.3 การค้นหาตามแนวลึก

การแวะผ่านกราฟแบบการค้นหาตามแนวลึก (Depth First Search) เป็นการเข้าถึงข้อมูลแต่ละโหนดในกราฟ โดยกำหนดโหนดเริ่มต้นแล้วแวะเยี่ยมโหนดทีละโหนดในระดับถัดลงไปเรื่อยๆตามเส้นทางในแนวลึกจนจบเส้นทาง แล้วจึงย้อนกลับมาเยี่ยมโหนดในเส้นทางอื่นๆถัดมาที่ยังไม่ได้แวะเยี่ยม พิจารณาการแวะผ่านกราฟในรูป 8.15 ตามแนวลึก

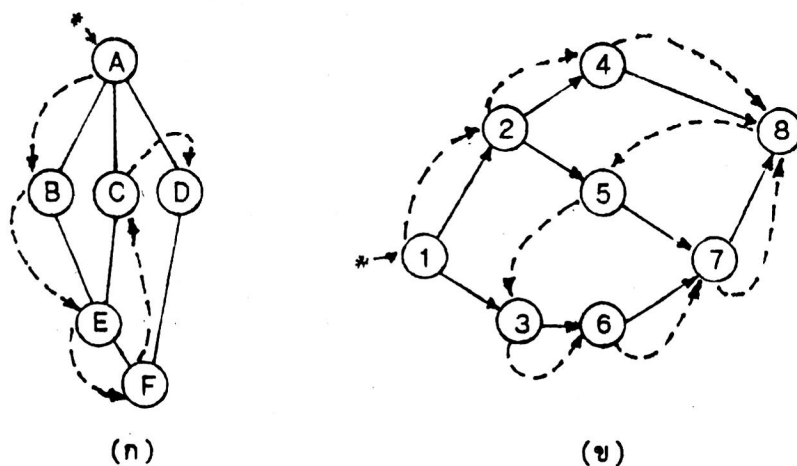


```

Procedure BFS(Firstnode);
Const visited = 1; unvisited = 0;
Var Savenode :Nodeid;
 Adjptr :Edgeptr;
Begin {init first node}
 Graph[Firstnode].mark := visited;
 Insert_Q(Q,Firstnode);
 While Not Empty(Q) Do
 Begin
 Savenode := Delete_Q(Q);
 Adjptr :=Graph[savenode].Adjlist;
 {visit node adjacent to savenode}
 While Adjptr <> Nil Do
 Begin
 Savenode := Adjptr^.Id;
 If (Graph[savenode].mark = unvisited) Then
 Begin Insert_Q(Q,Savenode);
 Graph[savenode].mark = visited
 End;
 Adjptr := Adjptr^.Next;
 End;
 End;
End;

```

รูป 8.14 ขั้นตอนวิธีค้นหาผ่านกราฟด้วยวิธีค้นหาตามแนวกว้าง

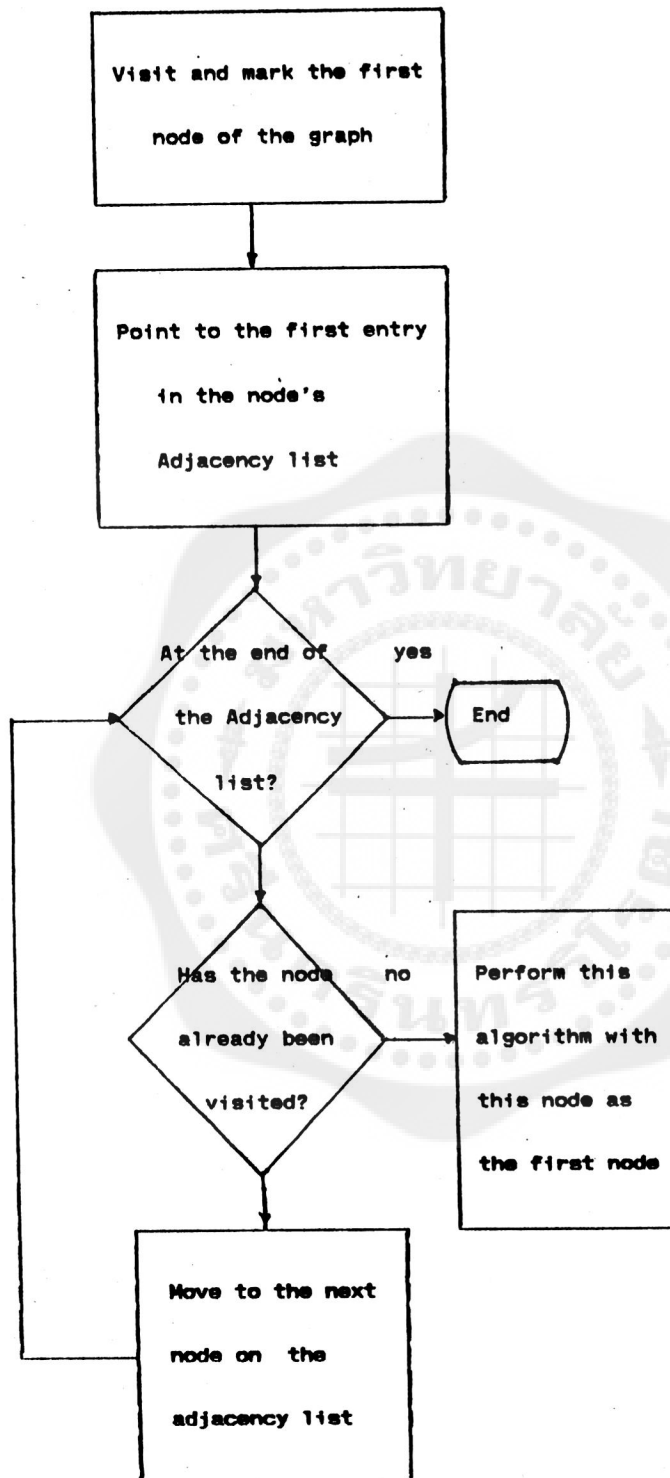


รูป 8.15 ตัวอย่างการแหว่ม่านกราฟด้วยวิธีการค้นตามแนวลึก

จากรูป 8.15 (ก) ลำดับของโหนดที่ถูกเข้าเยี่ยมคือ A,B,E,F,C,D หรือเป็น A,D,F,C,E,B ก็ได้ และจากรูป 8.15 (ข) ลำดับของโหนดที่ถูกเข้าเยี่ยมคือ 1,2,4,8,5,7,3,6 หรือเป็น 1,3,6,7,8,2,5,4 ก็ถูกต้องเช่นกัน

#### 8.3.4 ขั้นตอนวิธีการค้นตามแนวลึก

ขั้นตอนวิธีการแหว่ม่านกราฟด้วยวิธีค้นตามแนวลึก เป็นการแหว่ม่านกราฟที่สามารถเขียนเป็นขั้นตอนวิธีแบบเรียกซ้ำได้ แสดงด้วยม้งงานดังรูป 8.16 และเขียนแสดงด้วยโปรซิเจอร์ DFS ดังรูป 8.17 ทั้งนี้ใช้การแทนกราฟด้วย Adjacency List



รูป 8.16 ฟังก์ชันแสดงขั้นตอนวิธีผ่านกราฟด้วยวิธีค้นหาแบบลึก

```

Procedure DFS(Var Thisnode:Nodeid);
Const visited = 1;
 unvisited = 0;
Var Savenode : Nodeid;
 Adjptr : Edgeptr;
Begin {visit this node}
 Graph[Thisnode].mark := visited;
 Adjtr := Graph[Thisnode].Adjlist;
 While Adjptr <> Nil Do
 Begin Savenode := Graph[Thisnode].Id;
 If Graph[Savenode].mark = unvisited Then
 DFS(Savenode);
 Adjptr := Adjptr^.Next
 End;
 End;
End;

```

รูป 8.17 ขั้นตอนวิธีแหวะผ่านกราฟด้วยวิธีค้นตามแนวลึก

พิจารณาจำนวนรอบการทำงานของวิธีแหวะผ่านกราฟที่แทนด้วย Adjacency List ด้วยวิธีค้นตามแนวกว้าง การวนซ้ำวงนอกทำทั้งหมด  $N$  รอบ เมื่อ  $N$  เท่ากับ จำนวนโหนดในกราฟ และวนซ้ำรอบในเท่ากับ  $E$  ซึ่ง  $E$  เป็นจำนวนเส้นเชื่อมทั้งหมด ดังนั้น ประสิทธิภาพการทำงานจะอยู่ในรูป  $O(N \times E)$  และประสิทธิภาพการทำงานของวิธีค้นตามแนวลึกก็อยู่ในรูป  $O(N \times E)$  เช่นเดียวกัน ถ้าแทนกราฟด้วย Adjacency Matrix จำนวนรอบการทำงานทั้งหมดจะเท่ากับ  $N^2$  เนื่องจากต้องตรวจสอบเส้นเชื่อมโหนดทุกคู่ ดังนั้นประสิทธิภาพการทำงานจะอยู่ในรูป  $O(N^2)$  เสมอ เมื่อเปรียบเทียบแล้วจะเห็นว่า การแทนกราฟด้วย Adjacency List จะดีกว่าการแทนด้วย Adjacency Matrix เมื่อ  $E < N$  นั่นคือจำนวนเส้นเชื่อมระหว่างโหนดน้อยกว่าจำนวนโหนดของ

กราฟนั่นเอง

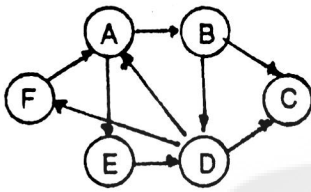
### 8.5 การประยุกต์ของกราฟ

การประยุกต์ของกราฟและเครือข่ายถูกนำไปใช้อย่างกว้างขวางทั้งในงานประยุกต์ต่าง ๆ เช่น ปัญหาการขนส่ง ปัญหาการบริหารโครงการด้วยวิธี PERT และ CPM ตลอดจนงานของคอมพิวเตอร์ เช่น โปรแกรมระบบจัดการฐานข้อมูล (DBMS) ก็มีการนำโครงสร้างแบบกราฟไปใช้ในการจัดโครงสร้างฐานข้อมูล



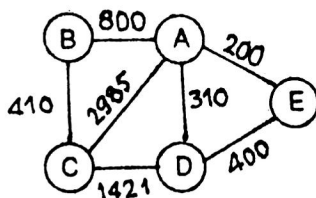
### แบบฝึกหัด

1. โครงสร้างข้อมูลแบบกราฟ แตกต่างจากโครงสร้างข้อมูลแบบต้นไม้อย่างไร
2. จากระบบกราฟต่อไปนี้



วาดรูปแสดงการแทนกราฟด้วยวิธี

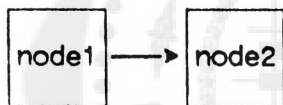
- (ก) Adjacency Matrix
  - (ข) Adjacency List
3. เขียนลำดับการแฉะผ่านกราฟในข้อ 2 โดยใช้วิธีการแฉะผ่านกราฟแบบ
    - (ก) การค้นหาตามแนวกว้าง
    - (ข) การค้นหาตามแนวลึก
  4. เขียน Adjacency Matrix ของกราฟแบบถ่วงน้ำหนัก ต่อไปนี้



กำหนดให้แทนกราฟด้วย Adjacency Matrix ต่อไปนี้ จงเขียนขั้นตอนวิธีสำหรับการ  
ในข้อ 5 ถึง 7

```
Const Maxnodes = 50;
Type Nodeptr = 1..Maxnodes;
Adjmatrix = Array[nodeptr,nodeptr] of Boolean;
Graph = Adjmatrix;
```

5. เขียนโปรซีเจอร์ join เพื่อทำการกำหนดเส้นเชื่อมระหว่างโหนด 1 และ 2 ดังรูป



กำหนดหัวโปรซีเจอร์ไว้ดังนี้

```
Procedure join (var adj : adjmatrix; node1,node2: nodeptr);
```

6. เขียนฟังก์ชัน adjacent เพื่อทดสอบว่ามีเส้นเชื่อมระหว่างโหนด 1 และ 2 หรือไม่  
กำหนดหัวฟังก์ชันไว้ดังนี้

```
Function adjacent (adj:adjmatrix;node1,node2:nodeptr):boolean;
```

7. เขียนโปรซีเจอร์ remove เพื่อลบเส้นเชื่อมระหว่างโหนด 1 และ 2 กำหนดหัวฟังก์ชันไว้  
ดังนี้

```
Procedure remove (var adj:adjmatrix; node1,node2:nodeptr);
```

8. กำหนดให้แทนกราฟแบบถ่วงน้ำหนัก ด้วย Adjacency Matrix ซึ่งกำหนดโครงสร้างไว้ดังนี้

```
Type edge = record
 adj:boolean;
 weight : integer
end;
edges = array [nodeptr,nodeptr] of edge;
Var graph : edges;
```

เขียนโปรซีเจอร์ joinwt เพื่อกำหนดเส้นเชื่อมพร้อมน้ำหนัก สำหรับ node1 และ node2  
กำหนดหัวโปรซีเจอร์ได้ดังนี้

```
Procedure joinwt (Var g :edges; node1,node2 : nodeptr;
 w : integer);
```



## บรรณานุกรม

น้สาชล โตอดิเทพย์, โครงสร้างข้อมูลเบื้องต้น, มหาวิทยาลัยศิลปากร.

Aho, Alfred V., DATA STRUCTURES AND ALGORITHMS, Addison-Wesley, 1983.

Dale, Nell, PASCAL PLUS DATA STRUCTURES, ALGORITHMS, AND ADVANCED PROGRAMMING, D.C. Health and Company, 1988.

Horowitz, Ellis, FUNDAMENTALS OF DATA STRUCTURES, computer Science Press, 1976.

Knapp, Judith, DATA STRUCTURES FOR BUSINESS PROGRAMMING, McGraw-Hills, 1989.

Loomis, Mary, DATA MANAGEMENT AND FILE STRUCTURES, Prentice-Hall, 1989.

Singh, Bhagat, INTRODUCTION TO DATA STRUCTURES, West Publishing, 1985

Tenenbaum, Aaron M., DATA STRUCTURES USING PASCAL, Prentice-Hall, 1981.

Tremblay, Jean-Paul, AN INTRODUCTION TO DATA STRUCTURES WITH APPLICATIONS, International Student Edition, 1984.

Wirth, Niklaus , ALGORITHMS & DATA STRUCTURES, Prentice-Hall, 1986

## ภาคผนวก ก : ภาษาปาสกาล

ในที่นี้จะกล่าวถึงเฉพาะคำสั่งที่ควรทราบสำหรับการแสดงขั้นตอนวิธีที่ใช้ในตำราเล่มนี้เท่านั้น

- หัวข้อ :
- ก.1 โครงสร้างโปรแกรมปาสกาล
  - ก.2 กฎเกณฑ์การตั้งชื่อข้อมูล
  - ก.3 ประเภทข้อมูล
  - ก.4 ขอบเขตและเครื่องหมาย
  - ก.5 ประเภทข้อมูลที่ผู้ใช้กำหนดขึ้น
  - ก.6 การประกาศ
  - ก.7 โปรแกรมย่อย
  - ก.8 คำสั่งประมวลผล
  - ก.9 คำสั่งการใช้เนื้อที่แบบไดนามิก
  - ก.10 คำสั่งเกี่ยวกับแฟ้มข้อมูล

### ก.1 โครงสร้างของโปรแกรมปาสกาล

ภาษาปาสกาลกำหนดโครงสร้างของโปรแกรมเป็นรูปแบบดังนี้

```

PROGRAM ชื่อโปรแกรม; {หัวโปรแกรม}
{-----}

LABEL
CONST
TYPE
VAR {ส่วนการประกาศ}
PROCEDURE
FUNCTION
{-----}

```

BEGIN

คำสั่ง; {ส่วนคำสั่งประมวลผล}

.....

คำสั่ง

END.

{จบโปรแกรมด้วยจุด '.' หลัง END}

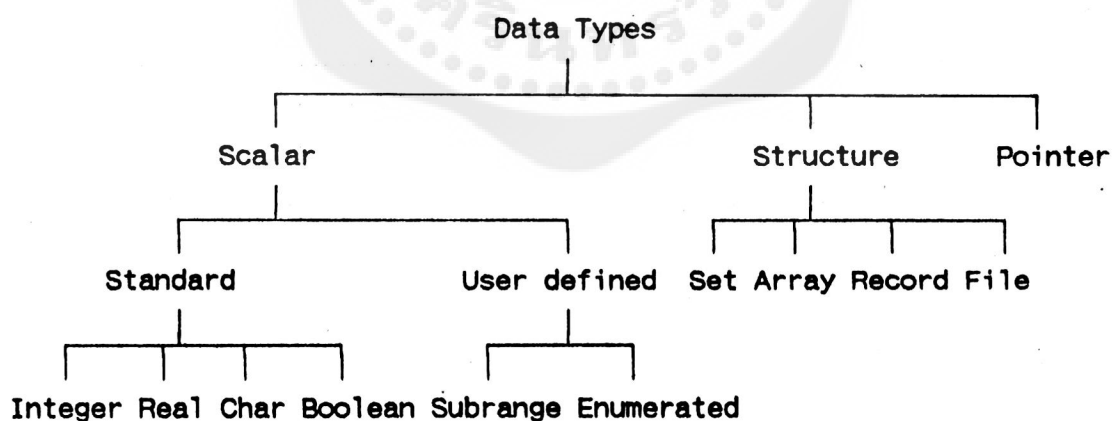
## ก.2 กฎเกณฑ์การตั้งชื่อข้อมูล

ชื่อข้อมูล ได้แก่ ชื่อโปรแกรม, ชื่อโปรซีเจอร์, ชื่อฟังก์ชัน, ชื่อไฟล์, ชื่อประเภทข้อมูล, ชื่อตัวแปร เป็นต้น

1. ขึ้นต้นด้วยตัวอักษร (A-Z หรือ a-z)
2. อาจตามด้วยตัวเลขหรือตัวอักษร
3. ใช้เครื่องหมายขีดเส้นใต้ (underbar) คั่นระหว่างชื่อได้
4. ห้ามซ้ำกับกับคำสงวน

## ก.3 ประเภทข้อมูล

ประเภทข้อมูล (Data Types) ในภาษาปาสคาล ได้แก่



Scalar เป็นข้อมูลซึ่งเป็นหน่วยย่อยที่สุด ได้แก่ประเภทข้อมูลมาตรฐานที่ใช้งานอยู่ทั่วไป (Integer, Real, Char, Boolean) และประเภทข้อมูลที่ถูกกำหนดขึ้น (Subrange, Enumerated)

Structure เป็นประเภทข้อมูลแบบโครงสร้าง โดยนำเอาข้อมูลแบบ scalar มาประกอบกัน ได้แก่ Array, Record, Set และ File

(ข้อมูลที่เก็บข้อความในเทอร์โบปาสกาลกำหนดเป็น String)

Pointer เป็นข้อมูลแบบตัวชี้

- \* ข้อมูลทุกตัวที่ใช้ในโปรแกรม ต้องถูกประกาศไว้ในส่วนการประกาศตัวแปร (VAR) ซึ่งต้องระบุประเภทข้อมูลกำกับไว้เสมอ
- \* ประเภทข้อมูลแต่ละแบบจะใช้ได้ภายในช่วงค่าหนึ่งซึ่งคอมพิวเตอร์ได้กำหนดไว้ และใช้งานได้เฉพาะกับเครื่องหมายหรือ Operation ที่ได้กำหนดไว้สำหรับประเภทข้อมูลนั้นเท่านั้น

ก.4 ขอบเขตของข้อมูลและเครื่องหมาย (ตามการกำหนดของเทอร์โบปาสกาล 5.0)

ก.4.1 Integer

ขนาดที่เก็บ : 2 ไบต์  
ขอบเขตข้อมูล: -32768 ถึง 32767

| เครื่องหมาย | ความหมาย                               |
|-------------|----------------------------------------|
| +           | บวก                                    |
| -           | ลบ                                     |
| *           | คูณ                                    |
| DIV         | หาร ตัดเศษทิ้งผลลัพธ์เป็นจำนวนเต็ม     |
| MOD         | เศษเหลือ (Modulus) (เศษเหลือจากการหาร) |

(ถ้าใช้เครื่องหมาย / หารค่าของ integer จะได้ผลลัพธ์เป็น Real)  
นอกจากนี้ค่าที่เป็นจำนวนเต็มในเทอร์โบปาสกาลยังมีแบบอื่น ๆ ได้แก่

| <u>ประเภท</u> | <u>ขนาดที่เก็บ(ไบต์)</u> | <u>ขอบเขตข้อมูล</u>              |
|---------------|--------------------------|----------------------------------|
| Byte          | 1                        | 0 ถึง 255                        |
| ShortInt      | 1                        | -128 ถึง 127                     |
| LongInt       | 4                        | -2,147,483,648 ถึง 2,147,483,647 |
| Word          | 2                        | 0 ถึง 65535                      |

#### ก.4.2 Real

ขนาดที่เก็บ : 4 ไบต์

ขอบเขตข้อมูล:  $-10^{-38}$  ถึง  $10^{38}$

| <u>เครื่องหมาย</u> | <u>ความหมาย</u> |
|--------------------|-----------------|
| +                  | บวก             |
| -                  | ลบ              |
| *                  | คูณ             |
| /                  | หาร             |

#### ก.4.3 Boolean

ขนาดที่เก็บ : 1 ไบต์

ขอบเขตข้อมูล: False, True (False = 0, True = 1)

| <u>เครื่องหมาย</u> | <u>ความหมาย</u> |
|--------------------|-----------------|
| NOT                | นิเสธ           |
| AND                | และ             |
| OR                 | หรือ            |
| XOR                | exclusive OR    |

#### ก.4.4 CHAR

ขนาดที่เก็บ : 1 ไบต์

ขอบเขตข้อมูล: ตัวอักษรใน ASCII Set

| เครื่องหมาย | ความหมาย             |
|-------------|----------------------|
| ORD (X)     | Ordinal Function     |
| CHR (I)     | Character Function   |
| PRED (X)    | Predecessor Function |
| SUCC (X)    | Successor Function   |

X : ตัวอักษร

I : จำนวนเต็ม

#### ก.4.5 ลำดับการทำงานของเครื่องหมาย

1. - (Unary minus)
- ก. NOT
3. \*, /, DIV, MOD, AND, SHL, SHR
4. +, -, OR, XOR
5. =, <>, <, >, <=, >=, IN

#### ก.5 ประเภทข้อมูลที่ผู้ใช้กำหนดขึ้น

##### ก.5.1 ขอบเขตย่อย

ประเภทข้อมูลแบบขอบเขตย่อย (subrange) เป็นการกำหนดประเภทข้อมูลแบบใหม่ให้มีค่าข้อมูลอยู่ภายในขอบเขตย่อยของประเภทข้อมูลแบบอื่นที่ได้กำหนดไว้แล้ว เช่น

VAR alpha : A..Z;

Score : 0..100;

### ก.5.2 เชิงลำดับ

ประเภทข้อมูลแบบเชิงลำดับ (Enumerated) เป็นกำหนดประเภทข้อมูลแบบใหม่ โดยแจกแจงทุกค่าของข้อมูลในประเภทนี้ตามลำดับจากน้อยไปมาก

ตัวอย่าง TYPE DAY = (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY);

GRADE = (F,D,C,B,A);

### ก.6 การประกาศ

ส่วนการประกาศประกอบด้วยส่วนการประกาศหลายประเภท ให้เลือกเขียนเฉพาะส่วนที่ต้องการใช้ในโปรแกรม

#### ก.6.1 ประกาศ LABEL

LABEL - เป็นชื่อของจุดทางเข้าของคำสั่งให้มาทำงานโดยคำสั่ง GOTO

LABEL ชื่อlabel;

ตัวอย่าง

LABEL Loop\_1, Finish;

.....

```

>Loop_1 : Readln (A);

 Goto Loop_1;

```

{\* ควรพยายามหลีกเลี่ยงการใช้คำสั่ง Goto}

### ก.6.2 ประกาศ CONST ประกาศชื่อสำหรับค่าคงที่

CONST ชื่อ = ค่าคงที่

ตัวอย่าง

CONST MAX = 1000;

RATE = 12.45;

### ก.6.3 ประกาศ TYPE

ประกาศประเภทข้อมูลแบบใหม่ขึ้นใช้ในโปรแกรม

TYPE ชื่อประเภทข้อมูล = ประเภทข้อมูล

ตัวอย่าง

TYPE Itemtype = Integer;

Message = String [20];

Range = 0..100;

Traffic = (Red, Yellow, Green);

#### (1) การประกาศขอบเขตย่อย

ค่าเริ่มต้น .. ค่าสิ้นสุด

Type Score = 0..100;

#### (2) การกำหนดข้อมูลแบบเชิงลำดับ



ค่าที่ 1, ค่าที่ 2; ค่าที่ 3,...,ค่าสุดท้าย

มีผลให้ ค่าที่ 1 < ค่าที่ 2 < ค่าที่ 3 <...< ค่าสุดท้าย

ประโยชน์

- สามารถใช้กับฟังก์ชัน PRED และ SUCC ได้
- ใช้ควบคุมการวนซ้ำได้
- ใช้กำหนดขอบเขตย่อยได้

Type classes = (Freshman, Sophomore, Junior, Senior)

### (3) การประกาศ Array

Array [ขอบเขตดัชนี] OF ประเภทข้อมูล

ตัวอย่าง

Type Table1 = Array [-10..10] of Integer;

Table2 = Array [A..D,1..40] of Real;

### (4) การประกาศ Record

Record

ชื่อเขตข้อมูล: ประเภทข้อมูล;

.....

ชื่อเขตข้อมูล: ประเภทข้อมูล

End

ตัวอย่าง    Type person = Record

                 name : string [20];

                 age : integer

         End;

(5) การประกาศ Set

Set of ประเภทข้อมูล

Type letter = Set of char;

(6) การประกาศ File

File of ประเภทข้อมูล

ประเภทข้อมูลเป็นได้ทั้ง scalar, structure และแบบที่ผู้ใช้กำหนดขึ้น

ตัวอย่าง

Type Data\_file = File of Integer;

In\_File = File of Person-Record;

กรณีที่เป็น Text File กำหนดได้ดังนี้

Type txt\_file = Text;

(7) ประกาศ pointer

Type pointer = ^ node;

node = Record

         info : integer;

         link : pointer

         End;

Var t, head : pointer;

#### ก.6.4 ประกาศตัวแปร

VAR    ตัวแปร : ประเภทข้อมูล

ตัวอย่าง

```

VAR a,
 b,c : Real;
 x,y : Integer;
 data : Person_Record;
 M : Data_Array;
 T : Table;
 Message: String[255];{max length}

```

#### ก.7 โปรแกรมย่อย

##### ก.7.1 การประกาศ Procedure

มีโครงสร้างเหมือนโปรแกรม ดังนี้

PROCEDURE      ชื่อโปรซีเจอร์ (พารามิเตอร์)

LABEL

CONST

TYPE

VAR

PROCEDURE

FUNCTION

BEGIN

คำสั่ง;

.....

คำสั่ง

END;

### ก.7.2 การประกาศ Function

มีโครงสร้างเหมือนโปรแกรม กำหนดหัวฟังก์ชันได้ดังนี้

Function ชื่อฟังก์ชัน (พารามิเตอร์): ประเภทข้อมูล

### ก.7.3 ประเภทของพารามิเตอร์

1. ส่งผ่านด้วยค่า (Passed by Value)
2. ส่งผ่านด้วยตำแหน่ง (passed by Address)

(1) พารามิเตอร์แบบส่งผ่านด้วยค่า จะถูกคัดลอกค่าส่ง ไปให้โปรแกรมย่อย การประกาศพารามิเตอร์ ไม่ต้องเขียนคำว่า VAR หน้าชื่อพารามิเตอร์

\* ใช้สำหรับการส่งค่าข้อมูลไปใช้ในการทำงานของโปรแกรมย่อย และ ไม่ต้องการผลลัพธ์ของข้อมูลตัวนั้นกลับมายัง โปรแกรมหลัก

(2) พารามิเตอร์แบบส่งผ่านด้วยตำแหน่ง จะส่งตำแหน่งของข้อมูลนั้นในหน่วยความจำไปยังโปรแกรมย่อย จึงเป็นการอ้างอิงถึงข้อมูลในตำแหน่งเดียวกัน การเปลี่ยนแปลงค่าข้อมูลนั้นในโปรแกรมย่อย ทำให้เปลี่ยนค่าข้อมูลที่ใช้ในโปรแกรมหลักอัตโนมัติ การประกาศพารามิเตอร์ต้องเขียนคำว่า VAR หน้าชื่อพารามิเตอร์

\* ใช้สำหรับข้อมูลที่ต้องการรับผลลัพธ์กลับมาใช้ในโปรแกรมหลัก

ตัวอย่าง

Procedure proc1 (A : Integer; B: Real;

VAR x : Real);

A และ B เป็นพารามิเตอร์แบบส่งผ่านด้วยค่า

X เป็นพารามิเตอร์แบบส่งผ่านด้วยตำแหน่ง

#### ก.7.4 การเรียกใช้โปรซีเจอร์

ชื่อโปรซีเจอร์ (พารามิเตอร์)

ตัวอย่าง

```
proc1 (i,j,y);
```

#### ก.7.5 การเรียกใช้ฟังก์ชัน

ฟังก์ชันมักใช้สำหรับการคำนวณที่ให้คำตอบเป็นผลลัพธ์เพียงค่าเดียว (ค่าของผลลัพธ์จะถูกกำหนดให้กับชื่อของฟังก์ชัน)

ตัวอย่าง

```
Function Power (x,y :Real): Real;
```

```
.....
```

```
Begin
```

```
Power := Exp(y * Ln(x)); <-- กำหนดผลลัพธ์ให้กับชื่อฟังก์ชัน
```

```
End;
```

คำสั่งเรียกใช้ฟังก์ชัน เขียนเรียกใช้ฟังก์ชันในรูปของนิพจน์ภายในคำสั่งอื่น เช่น คำสั่งกำหนดค่า, คำสั่งแสดงผล เช่น

```
Y := Power(a,b);
```

```
Z := Power(a,b)/ 2;
```

```
WriteLn (Power(a,b):10:2);
```

## ก.8 คำสั่งประมวลผล

คำสั่งประมวลผลเขียนอยู่ระหว่าง Begin และ End ได้แก่คำสั่งต่อไปนี้

|                           |                                                            |
|---------------------------|------------------------------------------------------------|
| อ่านข้อมูล                | Read, Readln                                               |
| แสดงผล                    | Write, Writeln                                             |
| ทางเลือก                  | IF.. Then..Else (2 ทางเลือก)<br>Case (หลายทางเลือก)        |
| กำหนดค่า                  | ตัวแปร := นิพจน์                                           |
| วนซ้ำ                     | While..Do<br>Repeat .. Until..<br>For..{To / Down to }..Do |
| เปลี่ยนลำดับไม่มีเงื่อนไข | Goto                                                       |

Compound Statement ได้แก่การกำหนดกลุ่มคำสั่งที่ต้องการให้ทำงานต่อเนื่องกัน เขียนอยู่ในช่วงของ Begin และ End

### ก.8.1 คำสั่งอ่านข้อมูล (Read / Readln)

Read (ตัวแปร, ตัวแปร,...)

Readln (ตัวแปร, ตัวแปร,...)

### ก.8.2 คำสั่งแสดงผล (Write/Writeln)

Write (รายการข้อมูล)

Writeln (รายการข้อมูล)

รายการข้อมูลอาจเป็นค่าคงที่หรือนิพจน์ เช่น

Writeln ('A = ' , x\*y :10:2);

### การกำหนดขอบเขตสำหรับแสดงผล

#### ค่าจำนวนเต็ม

นิพจน์ : ความกว้าง

เช่น Write ('I '=':7, I:5);

#### ค่าจำนวนจริง

นิพจน์ : ความกว้าง : จำนวนตัวเลขหลังจุดทศนิยม

เช่น Writeln (x/y : 7 : 3);

### ก.8.3 คำสั่งกำหนดค่า

ตัวแปร := นิพจน์

#### ตัวอย่าง

```
Var i,j,k : integer;
 rate, principle, interest : real;
 Ok : boolean;

j := 0;
i := i+1;
interest := principle * rate /100;
Ok := TRUE;
k := i div j;
```

#### ก.8.4 คำสั่งทางเลือก

##### ก.8.4.1 IF-THEN-ELSE

```
IF condition THEN Statement1
[ELSE Statement 2]
```

ใส่ ; เมื่อจบส่วนของ ELSE (ถ้ามี ELSE)

##### ก.8.4.2 CASE

```
CASE expression OF
 label1 : statement1;
 label2 : statement2;
 .
 .
 .
 labeln : statementn;
 [ELSE Statement;]
END;
```

ส่วนของ Else ใช้สำหรับกรณีค่าจากนิพจน์ไม่ตรงกับค่า label ข้างต้น

#### ก.8.5 คำสั่งวนซ้ำ

ก.8.5.1 WHILE..DO วนซ้ำแบบมีเงื่อนไขโดยทดสอบเงื่อนไข  
ก่อนเข้าส่วนวนซ้ำ จะเข้าทำงานเมื่อเงื่อนไขเป็นจริง



```
WHILE condition DO
 statement;
```

ก.8.5.2 REPEAT..UNTIL วนซ้ำแบบมีเงื่อนไขโดยทดสอบเงื่อนไขตอนท้ายของส่วนวนซ้ำ จะวนซ้ำจนกว่าเงื่อนไขเป็นจริง

```
REPEAT
 statement;
 ...
 statement
UNTIL Condition;
```

ก.8.5.3 FOR..TO/DOWNTO

คำสั่งควบคุมการวนซ้ำอัตโนมัติ

```
FOR ตัวแปร := ค่าเริ่มต้น { To } ค่าสิ้นสุด Do
 { Downto }
 Statement;
```

TO ใช้สำหรับการเปลี่ยนค่าตัวแปรควบคุมจากน้อยไปมาก  
DOWNTO ใช้สำหรับการเปลี่ยนค่าตัวแปรควบคุมจากมากไปน้อย  
ประเภทของตัวแปรอาจเป็น integer, char หรือค่าแบบเชิงลำดับ

ก.8.6 คำสั่ง GOTO

```
GOTO ชื่อ label
```

ชื่อ label ต้องประกาศไว้ในส่วนประกาศ LABEL และในโปรแกรมต้องมีคำสั่งที่มีชื่อ label ก้ำก้อย เพื่อเป็นจุดเข้าทำงานตามคำสั่ง GOTO ในลักษณะ

ชื่อ label: คำสั่ง;

### ก.9 คำสั่งการใช้เนื้อที่แบบไดนามิก

คำสั่งการใช้เนื้อที่แบบไดนามิกในเทอร์โบปาสกาล ประกอบด้วย New, Dispose, Mark, Release และ Freeman ในที่นี้จะกล่าวถึงเฉพาะคำสั่ง New และ Dispose เท่านั้น

#### ก.9.1 การประกาศโครงสร้างข้อมูลแบบไดนามิก

```
Type pointer = ^node; { pointer เป็นตัวชี้ node }
node
 = Record
 info : integer;
 link : pointer
 End;
Var t, head : pointer;
```

#### ก.9.2 การขอเนื้อที่

การขอเนื้อที่สำหรับข้อมูลหรือโหนดตามที่ประกาศไว้ ใช้คำสั่ง NEW

NEW (พารามิเตอร์แบบตัวชี้)

เช่น New(t);

#### ก.9.3 การคืนเนื้อที่

เมื่อไม่ต้องการใช้เนื้อที่ของข้อมูลดังกล่าวแล้ว ควรคืนเนื้อที่ให้กับระบบ โดยใช้คำสั่ง Dispose

DISPOSE (พารามิเตอร์แบบตัวชี้)

เช่น Dispose(t);

#### ก.9.4 การอ้างอิงถึงข้อมูล

ต้องใช้ตัวชี้เสมอ เช่น

- การอ้างอิงถึงระเบียบที่ t ช้อยู่ใช้  $t^{\wedge}$
- การอ้างอิงถึงเขตข้อมูล info ของระเบียบที่ t ช้อยู่ใช้  $t^{\wedge}.info$
- การอ้างอิงถึงเขตข้อมูล link ของระเบียบที่ t ช้อยู่ใช้  $t^{\wedge}.link$

#### ก.9.5 การกำหนดค่า

ใช้คำสั่งกำหนดค่าตามปกติ แต่ต้องอ้างอิงข้อมูลให้ถูกต้อง เช่น

$t^{\wedge}.info := 10;$

$t^{\wedge}.link := nil;$

ซึ่งอาจใช้คำสั่ง With เข้ามาช่วยก็ได้เช่น

with  $t^{\wedge}$  do

begin info := 10;

link := nil

end;

หมายเหตุ Nil เป็นตัวชี้ซึ่งแสดงถึงไม่ได้ชี้ไปที่ใด ในเทอร์โบกำหนดค่า Nil เป็น \$0000:\$0000 (\$ แสดงว่าเป็นค่าฐาน 16)

#### ก.10 คำสั่งเกี่ยวกับแฟ้มข้อมูล

##### ก.10.1 การประกาศชื่อแฟ้มข้อมูล

VAR ชื่อแฟ้มข้อมูล : File Of ประเภทข้อมูล

ตัวอย่าง Var Infi : File of Data\_Record;

#### ก.10.2 คำสั่ง ASSIGN

คำสั่ง Assign ใช้เพื่อกำหนดชื่อแฟ้มข้อมูลในจานบันทึก

ASSIGN (ชื่อแฟ้มข้อมูล, ชื่อแฟ้มข้อมูลในจานบันทึก)

ตัวอย่าง Assign(Infi, 'b:test1.dat');

#### ก.10.3 คำสั่งเปิดแฟ้มข้อมูล

REWRITE (ชื่อแฟ้มข้อมูล)

RESET (ชื่อแฟ้มข้อมูล)

REWRITE : เปิดแฟ้มโดยลบข้อมูลเดิมในแฟ้ม ใช้สำหรับการสร้างไฟล์ใหม่

RESET : เปิดแฟ้มแฟ้มข้อมูลโดยไม่ลบข้อมูลเดิมในแฟ้ม

ตัวอย่าง Reset(Infi);

#### ก.10.4 คำสั่งปิดแฟ้มข้อมูล

CLOSE (ชื่อแฟ้มข้อมูล)

ตัวอย่าง Close(Infi)

#### ก.10.5 คำสั่งอ่านข้อมูล

READ (ชื่อแฟ้มข้อมูล, ชื่อข้อมูล)

ตัวอย่าง Read(Infi, X)

ก.10.6 คำสั่งเขียนข้อมูล

WRITE (ชื่อแฟ้มข้อมูล, ชื่อข้อมูล)

ตัวอย่าง Write(Infi, X)

ก.10.7 คำสั่งเลื่อนตัวชี้แฟ้มข้อมูล

SEEK (ชื่อแฟ้มข้อมูล, ตำแหน่งข้อมูล)

ตัวอย่าง SEEK(Infi, I);

ก.10.8 ฟังก์ชันเกี่ยวกับแฟ้มข้อมูล

| ฟังก์ชัน                 | ผลที่ได้                   | ตัวอย่าง               |
|--------------------------|----------------------------|------------------------|
| EOF(ชื่อแฟ้มข้อมูล)      | แสดงการจบแฟ้ม (TRUE/FALSE) | While Not EOF(Infi) Do |
| FILESIZE(ชื่อแฟ้มข้อมูล) | จำนวนข้อมูลในแฟ้ม          | i := Filesize(infi)    |
| FILEPOS(ชื่อแฟ้มข้อมูล)  | ตำแหน่งของตัวชี้แฟ้มข้อมูล | i := Filepos(Infi)     |

## ดัชนีรูป

หน้า

### บทที่ 1

|         |                                       |   |
|---------|---------------------------------------|---|
| รูป 1.1 | การกำหนดตำแหน่งของข้อมูลในหน่วยความจำ | 3 |
| รูป 1.2 | การหาจำนวนเต็มแบบเครื่องหมายและขนาด   | 4 |
| รูป 1.3 | การหาจำนวนเต็มแบบฐานสิบ               | 7 |
| รูป 1.4 | การหาจำนวนเต็มแบบทศนิยมอัดแน่น        | 7 |
| รูป 1.5 | การหาเลขจำนวนจริงแบบจุดลอยตัว         | 8 |

### บทที่ 2

|          |                                                    |    |
|----------|----------------------------------------------------|----|
| รูป 2.1  | เปรียบเทียบเวลาที่เพิ่มขึ้นเมื่อจำนวนข้อมูลมากขึ้น | 17 |
| รูป 2.2  | ขั้นตอนการเรียงลำดับแบบฟอง                         | 18 |
| รูป 2.3  | ตัวอย่างการเรียงลำดับแบบฟอง                        | 19 |
| รูป 2.4  | ขั้นตอนการเรียงลำดับแบบคัดเลือก                    | 20 |
| รูป 2.5  | ตัวอย่างการเรียงลำดับแบบคัดเลือก                   | 21 |
| รูป 2.6  | ขั้นตอนการเรียงลำดับแบบแทรก                        | 22 |
| รูป 2.7  | ตัวอย่างการเรียงลำดับแบบแทรก                       | 23 |
| รูป 2.8  | วิธีการจัดกลุ่มข้อมูลในการเรียงลำดับแบบควิก        | 24 |
| รูป 2.9  | ขั้นตอนเรียงลำดับแบบควิก                           | 25 |
| รูป 2.10 | ขั้นตอนวิธีค้นหาข้อมูลแบบตามลำดับ                  | 28 |
| รูป 2.11 | ขั้นตอนวิธีค้นหาข้อมูลแบบตามลำดับที่ใช้เซนทิเนล    | 29 |
| รูป 2.12 | ลำดับข้อมูลเมื่อเทียบกับค่าตัวกลาง                 | 30 |
| รูป 2.13 | ขั้นตอนการค้นหาข้อมูลแบบทวิภาค                     | 32 |

### บทที่ 3

|          |                                                        |    |
|----------|--------------------------------------------------------|----|
| รูป 3.1  | ตัวอย่างตัวแปรแถวลำดับ                                 | 45 |
| รูป 3.2  | แถวลำดับแบบ 1 มิติ 2 มิติ และ 3 มิติ                   | 46 |
| รูป 3.3  | การเก็บแถวลำดับในหน่วยความจำ                           | 50 |
| รูป 3.4  | การเก็บแถวลำดับแบบสตมภ์เป็นหลัก                        | 52 |
| รูป 3.5  | ตัวอย่างการเก็บแถวลำดับสองมิติแบบลำดับแถวเป็นหลัก      | 53 |
| รูป 3.6  | ตัวอย่างการเก็บแถวลำดับแบบสองมิติแบบลำดับสตมภ์เป็นหลัก | 54 |
| รูป 3.7  | ตัวอย่างการเก็บแถวลำดับสามมิติแบบลำดับแถวเป็นหลัก      | 56 |
| รูป 3.8  | เมตริกซ์สามเหลี่ยม                                     | 57 |
| รูป 3.9  | การเก็บเมตริกซ์สามเหลี่ยมในแถวลำดับหนึ่งมิติ           | 58 |
| รูป 3.10 | ตัวอย่างเมตริกซ์แบบสพาซ                                | 58 |
| รูป 3.11 | การประยุกต์ใช้แถวลำดับสำหรับเก็บเมตริกซ์แบบสพาซ        | 59 |

### บทที่ 4

|          |                                                                                 |    |
|----------|---------------------------------------------------------------------------------|----|
| รูป 4.1  | โครงสร้างข้อมูลแบบคิว                                                           | 62 |
| รูป 4.2  | แสดงคิวว่าง                                                                     | 63 |
| รูป 4.3  | การใส่ข้อมูลเข้าในคิว                                                           | 63 |
| รูป 4.4  | การใส่ข้อมูลเข้าและนำข้อมูลออกจากคิว                                            | 64 |
| รูป 4.5  | การกำหนดโครงสร้างแบบคิวด้วยแถวลำดับ                                             | 65 |
| รูป 4.6  | ขั้นตอนวิธีใส่ข้อมูลเข้าในคิว                                                   | 66 |
| รูป 4.7  | ขั้นตอนวิธีนำข้อมูลออกจากคิว                                                    | 67 |
| รูป 4.8  | แสดงข้อเสียของคิว เมื่อยังมีที่ว่างส่วนหน้าคิวแต่ใส่ข้อมูลไม่ได้                | 67 |
| รูป 4.9  | คิววงกลม                                                                        | 68 |
| รูป 4.10 | ตัวอย่างการปรับตัวชี้ของคิววงกลมเมื่อใส่ข้อมูลเข้า<br>และนำข้อมูลออกจากคิววงกลม | 69 |
| รูป 4.11 | ขั้นตอนวิธีการใส่ข้อมูลเข้าในคิววงกลม                                           | 70 |
| รูป 4.12 | ขั้นตอนวิธีนำข้อมูลออกจากคิววงกลม                                               | 71 |

## บทที่ 5

|          |                                                                 |    |
|----------|-----------------------------------------------------------------|----|
| รูป 5.1  | โครงสร้างข้อมูลแบบสแตก                                          | 75 |
| รูป 5.2  | แสดงสแตกว่าง และสแตกเต็ม                                        | 75 |
| รูป 5.3  | การแทนสแตกด้วยตัวแปรแถวลำดับ                                    | 77 |
| รูป 5.4  | ขั้นตอนวิธีทดสอบสแตกว่าว่างหรือไม่                              | 78 |
| รูป 5.5  | ขั้นตอนวิธีนำข้อมูลออกจากสแตก                                   | 78 |
| รูป 5.6  | ขั้นตอนวิธีใส่ข้อมูลเข้าสแตก                                    | 79 |
| รูป 5.7  | การคำนวณค่าแฟกตอเรียลแบบเรียกซ้ำ                                | 80 |
| รูป 5.8  | ขั้นตอนวิธีคำนวณค่าแฟกตอเรียลแบบเรียกซ้ำ                        | 81 |
| รูป 5.9  | ขั้นตอนการประมวลผลโปรแกรมย่อยแบบเรียกซ้ำ                        | 83 |
| รูป 5.10 | นิพจน์คณิตศาสตร์ในรูปสัญกรณ์เติมกลาง                            | 84 |
| รูป 5.11 | ลำดับการทำงานของเครื่องหมายคำนวณ                                | 85 |
| รูป 5.12 | ลำดับการดำเนินการนิพจน์คำนวณ                                    | 85 |
| รูป 5.13 | นิพจน์คณิตศาสตร์ในรูปสัญกรณ์เติมหลัง                            | 86 |
| รูป 5.14 | ตัวอย่างนิพจน์แบบเติมกลางและแบบเติมหลัง                         | 86 |
| รูป 5.15 | ตัวอย่างการแปลงนิพจน์แบบเติมกลางเป็นนิพจน์เติมหลังโดยใช้สแตก    | 88 |
| รูป 5.16 | ขั้นตอนวิธีการแปลงนิพจน์แบบเติมกลางเป็นนิพจน์เติมหลัง           | 90 |
| รูป 5.17 | เกมหอคอยแห่งกรุงฮานอย                                           | 91 |
| รูป 5.18 | ตัวอย่างโปรแกรมและผลการทำงานสำหรับการแก้ปัญหาหอคอยแห่งกรุงฮานอย | 91 |

## บทที่ 6

|         |                                                |     |
|---------|------------------------------------------------|-----|
| รูป 6.1 | การเก็บข้อมูลในแถวลำดับ                        | 97  |
| รูป 6.2 | รูปแบบการเก็บข้อมูลโดยจัดสรรเนื้อที่แบบไดนามิก | 98  |
| รูป 6.3 | โหนดของรายการโยง                               | 98  |
| รูป 6.4 | ตัวอย่างรายการโยง                              | 99  |
| รูป 6.5 | การกำหนดโครงสร้างของโหนด                       | 99  |
| รูป 6.6 | ตัวอย่างรายการโยงเดี่ยว                        | 100 |



|          |                                                   |     |
|----------|---------------------------------------------------|-----|
| รูป 6.7  | แสดงการสร้างรายการโยง                             | 100 |
| รูป 6.8  | ขั้นตอนวิธีแฉะผ่านรายการโยง                       |     |
|          | และพิมพ์ข้อมูลของแต่ละโหนดออกมาตามลำดับ           | 101 |
| รูป 6.9  | ขั้นตอนวิธีการค้นข้อมูลในรายการโยง                | 102 |
| รูป 6.10 | ตัวอย่างการใส่ข้อมูลเข้าต้นรายการโยง              | 103 |
| รูป 6.11 | ขั้นตอนวิธีการใส่ข้อมูลเข้าต้นรายการโยง           | 103 |
| รูป 6.12 | ตัวอย่างการใส่ข้อมูลเข้ากลางรายการโยง             | 104 |
| รูป 6.13 | ขั้นตอนวิธีการใส่ข้อมูลเข้ากลางรายการโยง          | 104 |
| รูป 6.14 | ตัวอย่างการใส่ข้อมูลต่อท้ายรายการโยง              | 105 |
| รูป 6.15 | ขั้นตอนวิธีการใส่ข้อมูลต่อท้ายรายการโยง           | 106 |
| รูป 6.16 | ตัวอย่างการนำข้อมูลออกจากรายการโยง                | 106 |
| รูป 6.17 | ขั้นตอนวิธีการนำข้อมูลออกจากรายการโยง             | 107 |
| รูป 6.18 | ตัวอย่างการแทนรายการโยงด้วยแถวลำดับ               | 108 |
| รูป 6.19 | โหนดของรายการโยงคู่                               | 109 |
| รูป 6.20 | การกำหนดโครงสร้างของโหนด                          | 109 |
| รูป 6.21 | ตัวอย่างรายการโยงคู่                              | 109 |
| รูป 6.22 | ตัวอย่างการใส่ข้อมูลในรายการโยงคู่หลังโหนดที่ระบุ | 111 |
| รูป 6.23 | ตัวอย่างการใส่ข้อมูลในรายการโยงคู่หน้าโหนดที่ระบุ | 111 |
| รูป 6.24 | ตัวอย่างการลบโหนด                                 | 112 |

## บทที่ 7

|         |                                         |     |
|---------|-----------------------------------------|-----|
| รูป 7.1 | นิยามของต้นไม้                          | 129 |
| รูป 7.2 | โครงสร้างข้อมูลแบบต้นไม้                | 130 |
| รูป 7.3 | ระดับของต้นไม้                          | 131 |
| รูป 7.4 | การแทนโครงสร้างของต้นไม้แบบต่าง ๆ       | 132 |
| รูป 7.5 | นิยามของต้นไม้แบบทวิภาค                 | 133 |
| รูป 7.6 | การแทนพจน์คณิตศาสตร์ด้วยต้นไม้แบบทวิภาค | 133 |

|          |                                                                                      |     |
|----------|--------------------------------------------------------------------------------------|-----|
| รูป 7.7  | การกำหนดโหนดในต้นไม้แบบทวิภาค                                                        | 134 |
| รูป 7.8  | ตัวอย่างการแทนต้นไม้แบบทวิภาค                                                        | 135 |
| รูป 7.9  | ตัวอย่างต้นไม้ได้ดูล                                                                 | 136 |
| รูป 7.10 | ขั้นตอนวิธีสร้างต้นไม้ได้ดูล                                                         | 137 |
| รูป 7.11 | ตัวอย่างต้นไม้แบบทวิภาค                                                              | 138 |
| รูป 7.12 | ขั้นตอนวิธีการแฉะผ่านต้นไม้วิธีก่อนลำดับ                                             | 139 |
| รูป 7.13 | ขั้นตอนวิธีการแฉะผ่านต้นไม้วิธีตามลำดับ                                              | 139 |
| รูป 7.14 | ขั้นตอนวิธีการแฉะผ่านต้นไม้วิธีหลังลำดับ                                             | 140 |
| รูป 7.15 | การพิมพ์ข้อมูลในต้นไม้แบบทวิภาคด้วยวิธีตามลำดับ                                      | 140 |
| รูป 7.16 | มังงานแสดงขั้นตอนวิธีการแฉะผ่านต้นไม้ทวิภาคด้วยวิธีตามลำดับ<br>แบบไม่ใช้วิธีเรียกซ้ำ | 142 |
| รูป 7.17 | ขั้นตอนวิธีการแฉะผ่านต้นไม้ทวิภาคด้วยวิธีตามลำดับแบบไม่ใช้วิธีเรียกซ้ำ               | 143 |
| รูป 7.18 | นิยามของต้นไม้แบบค้นทวิภาค                                                           | 144 |
| รูป 7.19 | ตัวอย่างต้นไม้แบบค้นทวิภาค                                                           | 147 |
| รูป 7.20 | ขั้นตอนวิธีการค้นข้อมูลในต้นไม้แบบทวิภาค                                             | 148 |
| รูป 7.21 | ขั้นตอนการใส่ข้อมูลเข้าในต้นไม้แบบทวิภาค                                             | 150 |
| รูป 7.22 | ขั้นตอนการลบโหนดในต้นไม้แบบค้นทวิภาค                                                 | 151 |
| รูป 7.23 | ต้นไม้แบบทวิภาคที่เก็บข้อมูลเหมือนกันแต่รูปร่างต่างกัน                               | 151 |
| รูป 7.24 | นิยามต้นไม้แบบได้ดูล                                                                 | 152 |
| รูป 7.25 | ตัวอย่างต้นไม้ได้ดูล                                                                 | 152 |
| รูป 7.26 | ต้นไม้ที่เสียสมดุลจากการใส่ข้อมูลเข้า                                                | 153 |
| รูป 7.27 | การปรับสมดุลของต้นไม้                                                                | 153 |
| รูป 7.28 | การปรับสมดุลในการเพิ่มข้อมูลในต้นไม้ได้ดูล                                           | 154 |
| รูป 7.29 | การปรับสมดุลของต้นไม้ได้ดูลในการลบข้อมูล                                             | 156 |

## บทที่ 8

|          |                                                                   |     |
|----------|-------------------------------------------------------------------|-----|
| รูป 8.1  | ตัวอย่างกราฟ                                                      | 168 |
| รูป 8.2  | กราฟแสดงความสัมพันธ์ของข้อมูล                                     | 169 |
|          | (ก) น้ำหนักของโหนด = เศษเหลือจากการหารค่าโหนดปลายทางกับโหนดต้นทาง |     |
|          | (ข) น้ำหนักของโหนด = ระยะทางระหว่างแต่ละโหนด                      |     |
| รูป 8.3  | ตัวอย่างกราฟ                                                      | 170 |
| รูป 8.4  | การแทนกราฟด้วย Adjacency Matrix                                   | 171 |
| รูป 8.5  | ตัวอย่างการแทนกราฟแบบไม่ถ่วงน้ำหนักด้วย Adjacency List            | 172 |
| รูป 8.6  | ตัวอย่างการแทนกราฟแบบถ่วงน้ำหนักด้วย Adjacency List               | 173 |
| รูป 8.7  | การกำหนดโครงสร้างข้อมูลแบบกราฟด้วย Adjacency Matrix               | 174 |
| รูป 8.8  | การกำหนดโครงสร้างข้อมูลแบบกราฟถ่วงน้ำหนักด้วย Adjacency Matrix    | 174 |
| รูป 8.9  | การกำหนดโครงสร้างข้อมูลแบบกราฟด้วย Adjacency List                 | 175 |
| รูป 8.10 | การกำหนดโครงสร้างข้อมูลแบบกราฟถ่วงน้ำหนักด้วย Adjacency List      | 175 |
| รูป 8.11 | ตัวอย่างการแฉะผ่านกราฟด้วยวิธีการค้นหาแบบกว้าง                    | 176 |
| รูป 8.12 | สิ่งที่ช่วยในการแฉะผ่านกราฟด้วยวิธีค้นหาแบบกว้าง                  | 177 |
| รูป 8.13 | ผังงานแสดงขั้นตอนวิธีการแฉะผ่านกราฟด้วยวิธีค้นหาแบบกว้าง          | 178 |
| รูป 8.14 | ขั้นตอนวิธีแฉะผ่านกราฟด้วยวิธีค้นหาแบบกว้าง                       | 180 |
| รูป 8.15 | ตัวอย่างการแฉะผ่านกราฟด้วยวิธีการค้นหาแบบลึก                      | 181 |
| รูป 8.16 | ผังงานแสดงขั้นตอนวิธีแฉะผ่านกราฟด้วยวิธีค้นหาแบบลึก               | 182 |
| รูป 8.17 | ขั้นตอนวิธีแฉะผ่านกราฟด้วยวิธีค้นหาแบบลึก                         | 183 |

## ดัชนีตาราง

|           |                                                            |    |
|-----------|------------------------------------------------------------|----|
| ตาราง 1-1 | การแทนจำนวนเต็มแบบส่วนเติมเต็มสอง เมื่อใช้เนื้อที่ 4 บิต   | 5  |
| ตาราง 1-2 | การแทนจำนวนเต็มแบบส่วนเติมเต็มหนึ่ง เมื่อใช้เนื้อที่ 4 บิต | 6  |
| ตาราง 2-1 | เปรียบเทียบเวลาการเรียงลำดับข้อมูลแบบต่างๆ                 | 26 |
| ตาราง 4-1 | สรุปเงื่อนไขการใส่ข้อมูลเข้าและนำข้อมูลออกจากคิววงกลม      | 70 |





