คำนำ

จากการที่ครูผู้สอนคอมพิวเตอร์ได้รับการอบรมในหลักสูตรที่ 1 ทักษะการโปรแกรมพื้นฐาน และ หลักสูตรที่ 2 วิทยาการคอมพิวเตอร์เบื้องต้นและการโปรแกรมขั้นสูง มาแล้ว โดยในหลักสูตรทั้งสองนั้นคาดหวังว่าครูผู้ผ่านการอบรมจะมีทักษะการโปรแกรมและมีความมั่นใจในการเขียนโปรแกรมตลอดทั้งสามารถบูรณาการด้วยตัวอย่างที่ประยุกต์ใช้คณิตศาสตร์และโครงสร้างข้อมูลต่างๆ ที่เกี่ยวข้องเป็นอย่างดี แต่จากการศึกษาและรวบรวมข้อมูลของครู พบว่า ในช่วงเวลาที่ครูเข้ารับการอบรม ครูส่วนใหญ่ยังไม่มีประสบการณ์ในการสอนโดยเฉพาะด้านการโปรแกรม เพราะส่วนใหญ่ครูที่มาจากสายวิชาอื่น เมื่อผ่านการอบรมและได้ทำการสอนมาในระยะเวลาหนึ่งแล้ว จะเริ่มมองเห็นปัญหาในการเรียนการสอน ปัญหาบางอย่างครูอาจค้นคว้าหาคำตอบเองได้ แต่อาจมีปัญหาบางปัญหาที่ครูอาจต้องการความช่วยเหลือจากภายนอกเพื่อช่วยทำให้มีความรู้ความเข้าใจที่มากขึ้น ดังนั้น เพื่อเป็นการเติมเต็มในส่วนที่ขาดหายไปของครู และเป็นเพิ่มพูนความรู้ให้ครูสามารถส่งเสริมให้นักเรียนเข้าร่วมแข่งขันการเขียนโปรแกรมในระดับต่าง ๆ ที่มีอยู่ในประเทศได้ สมาคมวิทยาศาสตร์แห่งประเทศไทย ในพระบรมราชูปถัมภ์ ที่ประชุมคณบดีคณะวิทยาศาสตร์แห่งประเทศไทย และ สำนักงานคณะกรรมการการศึกษาขั้นพื้นฐาน กระทรวงศึกษาธิการ เห็นสมควรให้จัดอบรมครูแกนนำด้านการโปรแกรมขึ้น โดยความมุ่งหวังของการอบรมครูในหลักสูตรนี้ คือ การเน้นการฝึกปฏิบัติในการแก้ปัญหาด้วยการโปรแกรมให้ครูคอมพิวเตอร์ โดยจะเน้นให้ครูคอมพิวเตอร์ได้ทดลองกับโจทย์ปัญหาจริงที่ใช้ในการแข่งขัน เช่น ปัญหาที่ใช้ในการแข่งขันการเขียนโปรแกรมในงานสัปดาห์วันวิทยาศาสตร์แห่งชาติ ในระดับมัธยมศึกษาตอนปลาย และปัญหาที่ใช้ในการแข่งขันโอลิมปิกวิชาการระดับชาติ เป็นต้น

คณะทำงานยกร่างหลักสูตรครูแกนนำด้านการโปรแกรม หวังเป็นอย่างยิ่งว่า การอบรมครูครั้งนี้จะสามารถเพิ่มทักษะการโปรแกรม และความมั่นใจในการถ่ายทอดความรู้แก่ลูกศิษย์ และบุคลากรทางการศึกษาท่านอื่นที่ยังไม่มีโอกาสได้เข้าร่วมอบรม

ในโอกาส นี้ ขอขอบคุณ คณะอนุกรรมการยกร่างหลักสูตรครูแกนนำทุกท่าน ซึ่งประกอบด้วยผู้แทนคณาจารย์จาก จุฬาลงกรณ์มหาวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์ มหาวิทยาลัยศิลปากร สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง และ มหาวิทยาลัยบูรพา ที่กรุณาร่วมประชุมพิจารณากรอบเนื้อหา ตลอดทั้งรูปแบบการอบรมครูแกนนำอันจะก่อให้เกิดประโยชน์สูงสุดต่อครูผู้เข้ารับการอบรม โดยเฉพาะอย่างยิ่ง รองศาสตราจารย์ ธีรวัฒน์ ประกอบผล ที่ให้ความกรุณาเรียบเรียงเนื้อหาในคู่มืออบรมฉบับนี้ และ ผู้ช่วยศาสตราจารย์ ดร.กฤษณะ ชินสาร ที่ทำหน้าที่เป็นผู้ประสานงานและรวบรวมข้อมูล

ศาสตราจารย์ ดร.ชิดชนก เหลือสินทรัพย์
ประธานคณะอนุกรรมการอบรมครูสาขาคอมพิวเตอร์

หลักสูตรครูแกนนำด้านการโปรแกรม 60 ชั่วโมง

(Master Teacher in Programming)

จากการที่ครูผู้สอนคอมพิวเตอร์ได้รับการอบรมในหลักสูตรที่ 1 และ หลักสูตรที่ 2 (ทักษะการโปรแกรมพื้นฐาน และ วิทยาการคอมพิวเตอร์เบื้องต้นและการโปรแกรมขั้นสูง) มาแล้ว ซึ่งครูผู้ผ่านการอบรมทั้งสองหลักสูตรนี้จะมีทักษะการโปรแกรมและสร้างความมั่นใจในการเขียนโปรแกรมแก่ครูและบูรณาการด้วยตัวอย่างที่ประยุกต์ใช้คณิตศาสตร์และโครงสร้างข้อมูลต่างๆ ที่เกี่ยวข้องเป็นอย่างดี ดังนั้น ความมุ่งหวังต่อไปของการอบรมครูคอมพิวเตอร์ คือ การเน้นการฝึกปฏิบัติในการแก้ปัญหาด้วยการโปรแกรมให้ครูคอมพิวเตอร์ โดยจะเน้นให้ครูคอมพิวเตอร์ได้ทดลองกับโจทย์ปัญหาจริงที่ใช้ในการแข่งขัน เช่น ปัญหาที่ใช้ในสัปดาห์วันวิทยาศาสตร์แห่งชาติ (ในระดับมัธยมศึกษาตอนปลาย) ปัญหาที่ใช้ในการแข่งขันโอลิมปิกวิชาการทั้งในระดับชาติ

วัตถุประสงค์ของการอบรมในหลักสูตรนี้ คือ

1. ครูผู้เข้ารับการอบรมสามารถวิเคราะห์ความต้องการเกี่ยวกับข้อมูลนำเข้า ข้อมูลส่งออกและ ความต้องการด้านทรัพยากรของปัญหาต่างๆ ที่กำหนดให้ได้ในเวลาที่กำหนด (โจทย์เป็นภาษาไทยและภาษาอังกฤษ)
2. ครูผู้เข้ารับการอบรมสามารถเลือกโครงสร้างข้อมูลและขั้นตอนวิธีที่เหมาะสมสำหรับการแก้ปัญหาต่างๆ ที่กำหนดให้ได้
3. ครูผู้เข้าอบรมสามารถเขียนโปรแกรมสำหรับการแก้ปัญหาตามที่กำหนดให้ได้ในเวลาที่กำหนด โดยโปรแกรมที่เขียนจะต้องใช้เวลาและทรัพยากรน้อยกว่าที่โจทย์กำหนด
4. ครูผู้เข้าอบรมจะสามารถนำความรู้ที่ได้ไปถ่ายทอดกับเพื่อนครูและนักเรียนที่อยู่ในความรับผิดชอบ ให้เข้าใจรูปแบบการแข่งขันการแก้ปัญหาด้วยการโปรแกรมทั้งในระดับชาติ ตลอดทั้งสามารถสอนนักเรียนให้สามารถเข้าร่วมแข่งขันในกิจกรรมดังกล่าวข้างต้นได้

เนื้อหาการอบรม ประกอบด้วย

บทที่	ตัวอย่างโจทย์ปัญหา	จำนวนชั่วโมง
ทบทวนกระบวนการแก้ปัญหาด้วยคอมพิวเตอร์ – Problem-based learning – Problem Solving Steps	– การแบ่งกลุ่มข้อมูล – การค้นหาข้อมูล – การคำนวณค่าทางสถิติต่างๆ	9
ทบทวนหลักการโปรแกรมเบื้องต้น – โครงสร้างโปรแกรม – การประกาศตัวแปร – คำสั่งรับข้อมูล และแสดงค่าข้อมูล – นิพจน์และตัวดำเนินการ – ค่าคงที่ – การใช้ Editor/Compiler – การทำงานแบบทางเลือก (if, switch) – การทำงานแบบวนซ้ำ (for, while, do while) – Function and Procedure – Parameter passing – Structures	– โปรแกรมรับค่าและแสดงค่าอย่างง่าย – โปรแกรมรับค่าตัวเลขเพื่อคำนวณพื้นที่รูปทรงเรขาคณิตอย่างง่าย	9
โครงสร้างข้อมูลและตัวอย่างการประยุกต์ใช้งาน	– 1D Array, 2D Array, Stack, Queue, Tree, Graph	24
ครูคอมพิวเตอร์แกนนำด้านการโปรแกรม	– ตัวอย่างโจทย์ในการแข่งขันสัปดาห์วันวิทยาศาสตร์แห่งชาติ และโอลิมปิกระดับประเทศ	18

ปฏิทินการอบรมหลักสูตร 4 วันที่ 3-12 พฤษภาคม 2553

ปฏิทินการอบรม
โครงการอบรมครูวิทยาศาสตร์ คณิตศาสตร์ คอมพิวเตอร์ โลกและ ดาราศาสตร์
วิชาคอมพิวเตอร์ หลักสูตรครูแกนนำด้านการโปรแกรม (60 ชั่วโมง)

วันที่	เวลา	เนื้อหาอบรม	วิทยากร
3	08.30 - 09.00	ลงทะเบียน	
	09.00 - 09.30	พิธีเปิดการอบรม	
	09.30 - 12.00	เปิดเวทีซักถามปัญหาด้านการโปรแกรม เพื่อรับทราบถึงปัญหาและอุปสรรคในการสอนวิชาการเขียนโปรแกรม ตลอดถึงเทคนิคที่ต้องการให้ศูนย์อบรมเพิ่มเติมให้ โดยคำถามควรอยู่ในประเด็นดังนี้ – predicates as conditions for selections and iterations – subprograms: procedures and functions – pointers – array and pointers – structures as aggregation of data – self-referential structures and pointers พร้อมนี้ให้แจ้งผู้เข้าอบรมทราบว่าจะมีการสอบ Post-Test จำนวน 3 วิชา คือ – เทคโนโลยีสารสนเทศ (ผู้เข้าอบรมต้องศึกษาด้วยตนเอง) – หลักการแก้ปัญหากับภาษาคอมพิวเตอร์ – คณิตศาสตร์กับภาษาคอมพิวเตอร์	
	13.00 - 16.30	**ทบทวนกระบวนการแก้ปัญหาด้วยคอมพิวเตอร์** (โดยอ้างอิงกับปัญหาที่ได้รับจากครูผู้เข้ารับการอบรมในช่วงเช้า) – Problem-based learning – Problem Solving Steps	
4	09.00-12.00	**ทบทวนกระบวนการแก้ปัญหาด้วยคอมพิวเตอร์** (ต่อ) – Problem-based learning – Problem Solving Steps	
	13.00-16.30	**ทบทวนหลักการโปรแกรมเบื้องต้น** – โครงสร้างโปรแกรม – การประกาศตัวแปร – คำสั่งรับข้อมูล และแสดงค่าข้อมูล – นิพจน์และตัวดำเนินการ – ค่าคงที่ – การใช้ Editor/Compiler – การทำงานแบบทางเลือก (if, switch) – การทำงานแบบวนซ้ำ (for, while, do while) – Function and Procedure – Parameter passing (with Pointer) – Structures	
5	09.00-12.00	**ทบทวนหลักการโปรแกรมเบื้องต้น** (ต่อ)	
	13.00-16.00		
6	09.00-12.00	**1D Array**	
	13.00-16.00	**ฝึกปฏิบัติการการโปรแกรมเบื้องต้นด้วยโจทย์ปัญหา เช่น** – การแบ่งประเภทข้อมูล (เน้น if-else-if, switch) – การค้นหาข้อมูล – การคำนวณค่าทางสถิติต่างๆ – โปรแกรมรับค่าและแสดงค่าอย่างง่าย – การเรียงข้อมูล	

วันที่	เวลา	เนื้อหาอบรม	วิทยากร
7	09.00-12.00 13.00-16.00	**2D Array** (Sorting and Searching algorithms) **ฝึกปฏิบัติการการโปรแกรมเบื้องต้นด้วยโจทย์ปัญหา ดังนี้** – การแบ่งประเภทข้อมูล (เน้น if-else-if, switch) – การค้นหาข้อมูล – การคำนวณค่าทางสถิติต่างๆ – โปรแกรมรับค่าและแสดงค่าอย่างง่าย	
8	09.00-12.00 13.00-16.00	**แนวคิดเกี่ยวกับ พร้อมยกตัวอย่างของการโปรแกรม** **Structures and Self-Referential Structures** **Dynamic Memory Allocation and Linked-list**	
9	09.00-12.00 13.00-16.00	**แนวคิดเกี่ยวกับ พร้อมยกตัวอย่างของการโปรแกรม** **Stack, Queue, Tree and Graph**	
10	09.00-10.00 10-00-12.00 13.00-16.00	**แนะนำการแข่งขันและการประกวดโปรแกรมในประเทศ** **แนะนำลักษณะโจทย์การแข่งขันระดับประเทศและแบ่งกลุ่มทำโจทย์** **แนะนำลักษณะโจทย์การแข่งขันระดับประเทศและแบ่งกลุ่มทำโจทย์** (ต่อ)	
11	09.00-12.00 13.00-16.00	**แนะนำลักษณะโจทย์การแข่งขันระดับประเทศและแบ่งกลุ่มทำโจทย์** (ต่อ) **สอบข้อสอบ Post-test**	
12	09.00-12.00 13.00-15.00 15.00-16.00	**สอบวัดคุณสมบัติครูแกนนำด้านการโปรแกรม ข้อสอบตอนที่ 1** (ปฏิบัติ) **สอบวัดคุณสมบัติครูแกนนำด้านการโปรแกรม ข้อสอบตอนที่ 2** (ทฤษฎี) **สรุปและตอบคำถามที่เกี่ยวข้องกับการอบรม มอบเกียรติบัตร และปิดการอบรม**	

การให้คะแนนหลักสูตรครูแกนนำด้านการโปรแกรม

- ผู้เข้าอบรมทุกคนต้องเข้าอบรมอย่างน้อยร้อยละ 80 จึงจะมีสิทธิ์เข้าสอบ และได้รับเกียรติบัตร
- เกณฑ์การให้คะแนนเป็นดังนี้

	สอบ		
	การบ้าน	**ข้อสอบตอนที่ 1** (ปฏิบัติ)	**ข้อสอบตอนที่ 2** (ทฤษฎี)
คะแนน	25	40	35

การตัดสินผล

0-49	ตก
50-79	ผ่าน
80-100	ดีมาก

คู่มืออบรมครูคอมพิวเตอร์

หลักสูตรครูแกนนำด้านการโปรแกรม

บทที่ 2
องค์ประกอบของระบบคอมพิวเตอร์

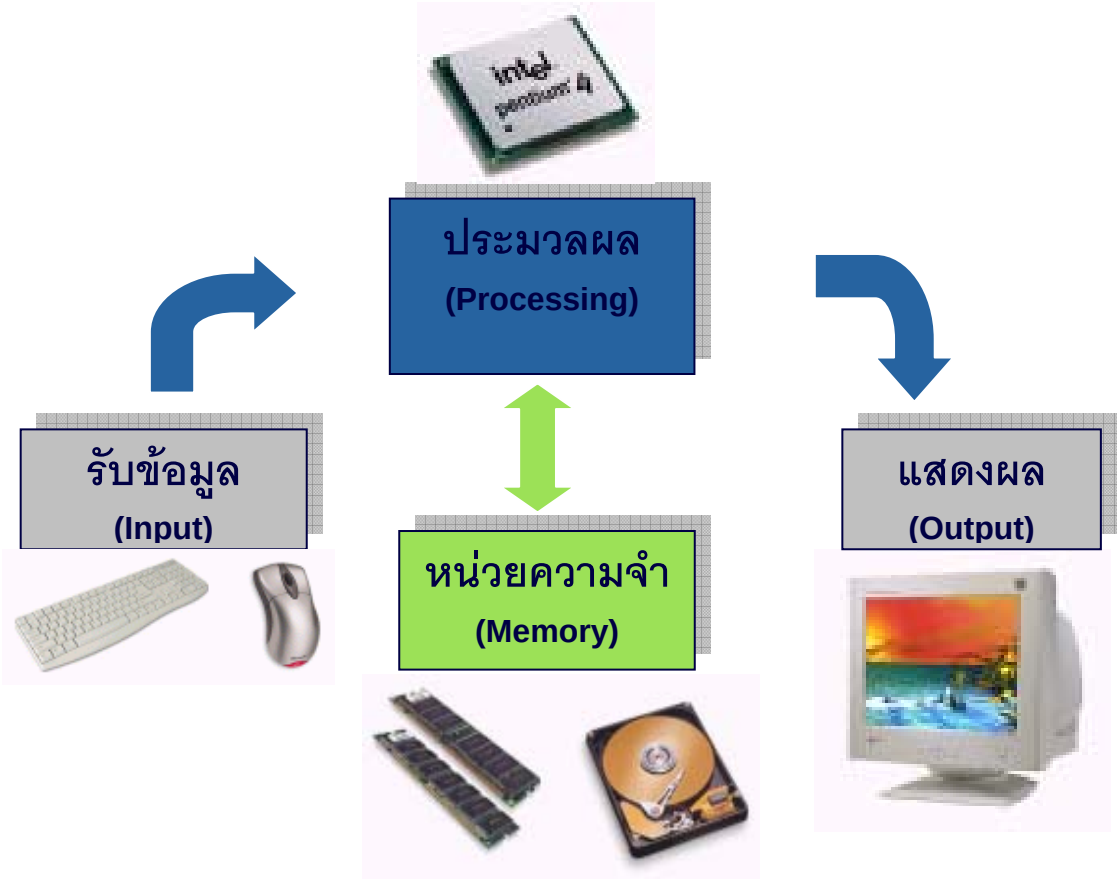
เครื่องคอมพิวเตอร์เป็นเครื่องมือที่อำนวยความสะดวกในการทำงานให้กับมนุษย์หลาย ๆ ด้าน การทำงานของระบบคอมพิวเตอร์นั้นจะต้องมีองค์ประกอบที่สำคัญ 4 ส่วน คือ ตัวเครื่องหรือฮาร์ดแวร์ (Hardware) โปรแกรมสำหรับทำงานหรือซอฟต์แวร์(Software) มนุษย์ และข้อมูลต่าง ๆ ที่จะให้คอมพิวเตอร์ประมวลผล

2.1 ความหมายของคอมพิวเตอร์

คอมพิวเตอร์เป็นอุปกรณ์ทางไฟฟ้าชนิดหนึ่งที่สามารถจำข้อมูลต่าง ๆ ได้ สามารถคิดคำนวณตัวเลข สามารถตอบสนองต่อการกระทำของผู้ใช้ได้ และมีความสามารถในการเชื่อมต่อกับอุปกรณ์ต่อพ่วงบางชนิดเข้ากับเครื่องคอมพิวเตอร์ เพื่อสั่งให้อุปกรณ์เหล่านั้นทำงานตามคำสั่งได้ เมื่อกล่าวถึงฮาร์ดแวร์คอมพิวเตอร์ (Computer Hardware) โดยทั่วไปจะหมายถึงตัวเครื่องคอมพิวเตอร์รวมทั้งอุปกรณ์ต่าง ๆ ที่ต่ออยู่กับเครื่องคอมพิวเตอร์

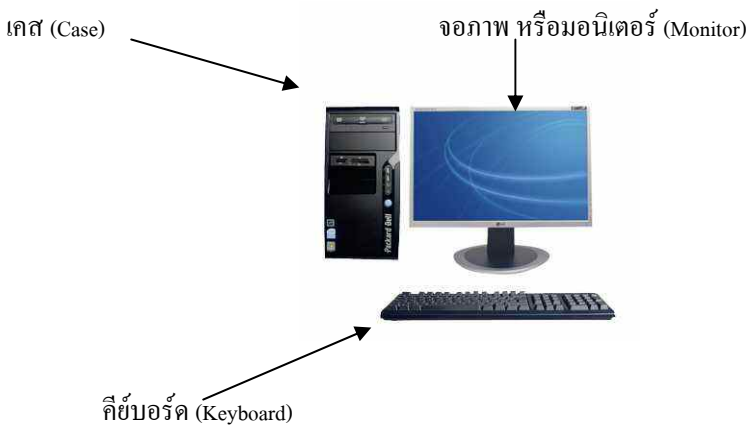
คอมพิวเตอร์ (ตามพจนานุกรมฉบับราชบัณฑิตยสถาน พ.ศ. 2525) หมายถึงเครื่องอิเล็กทรอนิกส์แบบอัตโนมัติ ทำหน้าที่เสมือนสมองกล ใช้สำหรับแก้ปัญหาต่าง ๆ ทั้งที่ง่ายและซับซ้อน โดยวิธีทางคณิตศาสตร์

ขั้นตอนการทำงานของคอมพิวเตอร์นั้นประกอบด้วย 3 ขั้นตอนใหญ่ ๆ คือ การรับข้อมูล(Input Data) การประมวลผล(Data Processing) และการแสดงผลลัพธ์ออกทางเอาต์พุต (Output Result) ดังรูปที่ 2.1 การรับข้อมูลนั้นคอมพิวเตอร์จะรับข้อมูลเข้ามาทางอุปกรณ์ที่ต่ออยู่ เช่น คีย์บอร์ต เมาส์ ส่วนการแสดงผลลัพธ์จะแสดงออกทางจอภาพหรือเครื่องพิมพ์เป็นหลัก สำหรับวิธีการประมวลผลนั้นคอมพิวเตอร์จะอ่านโปรแกรมจากหน่วยความจำที่เก็บโปรแกรมไว้ขึ้นมาไว้ในหน่วยความจำหลักเพื่อประมวลผลต่าง ๆ ต่อไป



2.2 ส่วนประกอบของคอมพิวเตอร์

คอมพิวเตอร์มีผลิตออกมาหลายรุ่นแต่ละรุ่นจะมีความสามารถต่างกัน บางรุ่นเหมาะสำหรับทำงานในบ้าน บางรุ่นเหมาะสำหรับการใช้งานทางด้านกราฟิกระดับสูง แต่ถ้าหากมองถึงส่วนประกอบภายนอกจะประกอบด้วยองค์ประกอบหลัก 5 ส่วน คือ หน่วยประมวลผลกลาง (Central Processing Unit) หรือซีพียู (CPU) หน่วยรับเข้า (Input Unit) หน่วยส่งออก(Output Unit) หน่วยความจำหลัก (Main Memory) และหน่วยความจำรอง (Secondary Memory)

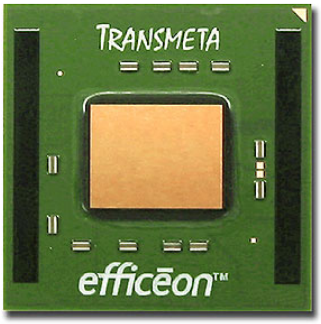


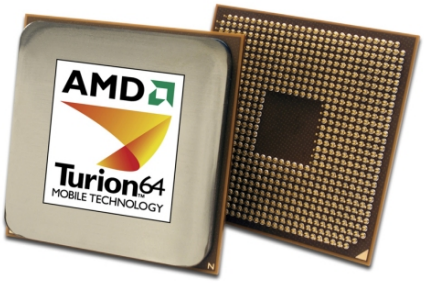
หน่วยประมวลผลกลาง

หากกล่าวถึงหน่วยประมวลผลกลางของคอมพิวเตอร์ หรือ ซีพียู (Central Processing: CPU) บางคนเข้าใจผิดคิดว่าเป็นตัวเครื่องหรือ เคส(Case) ของคอมพิวเตอร์ แต่ตามความจริงแล้ว ซีพียู เป็นอุปกรณ์ทางอิเล็กทรอนิกส์ตัวหนึ่งที่เป็นหัวใจการทำงานของคอมพิวเตอร์ ภายในตัวมันประกอบด้วยส่วนสำคัญสองส่วนคือ หน่วยควบคุม(Control Unit) ทำหน้าที่ควบคุมการทำงานต่าง ๆ และหน่วยคำนวณทางคณิตศาสตร์และตรรกะ ซึ่งจะทำหน้าที่ในการคำนวณข้อมูลต่าง ๆ เช่น การบวก ลบ คูณ หาร และการเปรียบเทียบทางตรรกศาสตร์ การทำงานของซีพียูจะต้องเขียนคำสั่งเพื่อสั่งงานให้ซีพียูทำงานตามที่ต้องการ ตัวซีพียูนี้ถือว่าเป็นศูนย์รวมการทำงานทั้งหมดของระบบคอมพิวเตอร์ หรือเปรียบเหมือนเป็นสมองของคอมพิวเตอร์นั้นเอง

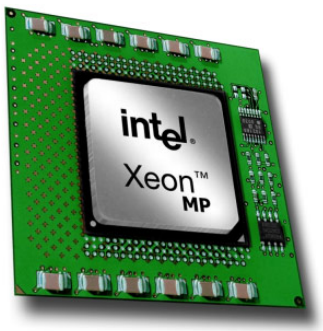
ในอดีตหน่วยตัวซีพียูนี้จะมีขนาดใหญ่ แต่ในปัจจุบันเมื่อนำสารกึ่งตัวนำมาใช้ก็ทำให้ซีพียูถูกพัฒนาให้มีขนาดเล็กลง โดยรวมวงจรต่าง ๆ ไว้ภายในตัวมันเพียงตัวเดียวเรียกว่า ไมโครโปรเซสเซอร์ (Microprocessor) ดังนั้นอาจกล่าวได้ว่าคอมพิวเตอร์ในปัจจุบันมีไมโครโปรเซสเซอร์เป็นหน่วยประมวลผลกลาง

ในปัจจุบันได้มีการผลิตไมโครโปรเซสเซอร์ออกมาหลายรุ่น โดยอินเทล (Intel) เป็นบริษัทใหญ่ที่ผลิตไมโครโปรเซสเซอร์ออกมามากมาย นอกจากนี้ยังมีไมโครโปรเซสเซอร์ของบริษัทอื่น ๆ อีกที่ผลิตซีพียูให้เข้ากันได้กับซีพียูของอินเทล ที่เรียกว่า Intel-Compatible Processor โดยสามารถใช้ชุดคำสั่งและโปรแกรมเหมือนกับของอินเทลได้ เช่นไมโครโปรเซสเซอร์ของเอเอ็มดี (AMD) ของไซริก(Cyrix) เป็นต้น





ตัวอย่างไมโครโปรเซสเซอร์สำหรับคอมพิวเตอร์ และคอมพิวเตอร์แบบพกพา



ตัวอย่างไมโครโปรเซสเซอร์สำหรับเครื่อง workstation

หน่วยรับเข้า

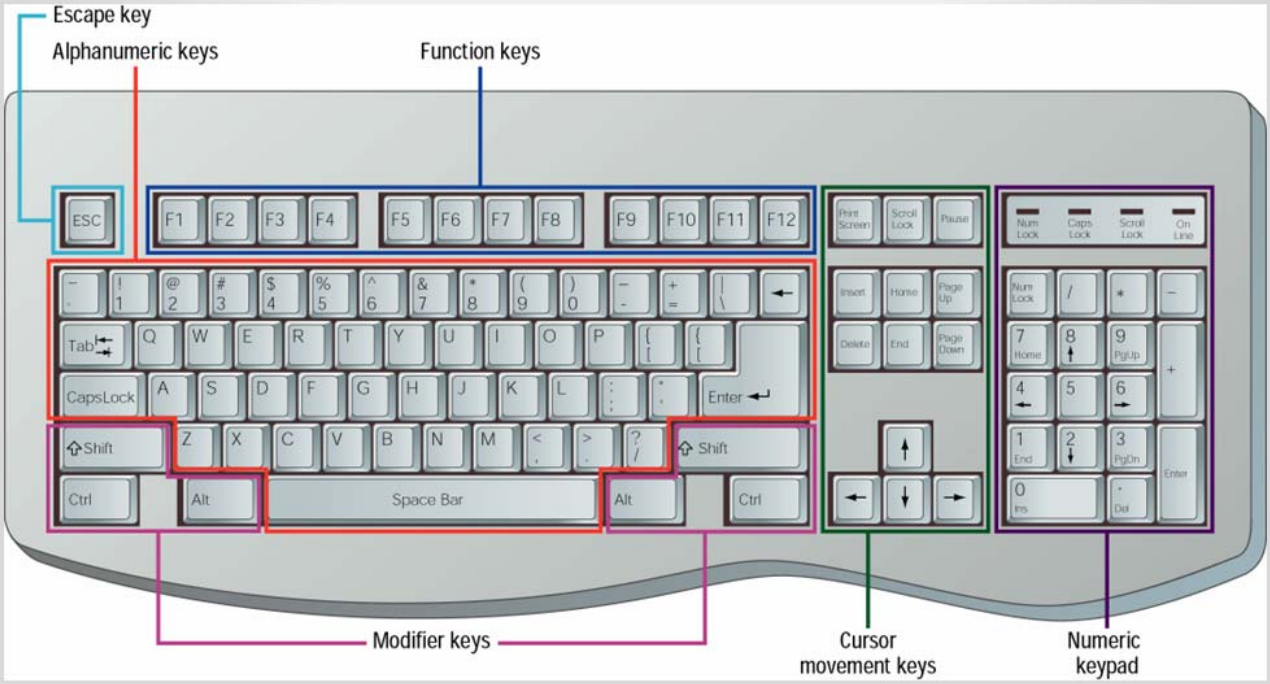
แม้ว่าคอมพิวเตอร์จะเป็นอุปกรณ์อิเล็กทรอนิกส์ที่ประมวลผลได้ แต่ก็สามารถติดต่อกับมนุษย์ได้ทาง **พอร์ต (port)** ซึ่งแบ่งออกเป็นพอร์ตอินพุตและพอร์ตเอาต์พุต หากต้องการรับข้อมูลเข้ามาประมวลผลจะต้องมีอุปกรณ์รับข้อมูล (Input Devices) มาต่อกับพอร์ตอินพุตของคอมพิวเตอร์ โดยอุปกรณ์รับข้อมูลนี้ทำหน้าที่รับข้อมูลและคำสั่งเข้าสู่คอมพิวเตอร์ โดยจะนำลักษณะของข้อมูลที่ถูกป้อนเข้ามาเปลี่ยนไปเป็นสัญญาณที่คอมพิวเตอร์เข้าใจ การใช้อุปกรณ์ต่าง ๆ นั้น โปรแกรมที่ใช้งานอยู่จะต้องสนับสนุนกับอุปกรณ์ประเภทนั้นด้วย ตัวอย่างของอุปกรณ์อินพุตได้แก่ แป้นพิมพ์หรือคีย์บอร์ต(Keyboard) เมาส์(Mouse) เครื่องสแกน (Scanner) กล้องดิจิตอล(Digital Camera) เครื่องอ่านบาร์โค้ด เป็นต้น

คีย์บอร์ด (Keyboard)

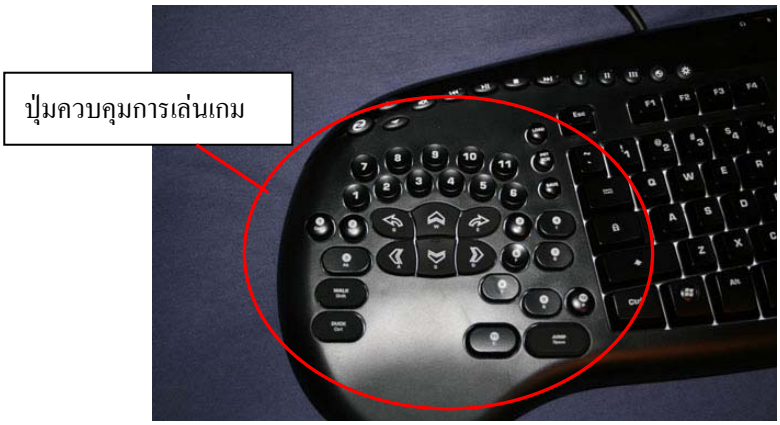
คีย์บอร์ดเป็นอุปกรณ์ป้อนข้อมูลหรือคำสั่งที่นิยมใช้มากที่สุด และเป็นอุปกรณ์อินพุตแบบดั้งเดิมที่ทำงานร่วมกับคอมพิวเตอร์มานาน โดยเฉพาะการพิมพ์ข้อมูล สร้างเอกสารต่าง ๆ ร่วมถึงกลุ่มคีย์พิเศษที่สามารถทำงานร่วมกับโปรแกรมได้ คีย์บอร์ดนี้นับว่าเป็นอุปกรณ์รับข้อมูลเข้าที่คอมพิวเตอร์จะขาดไม่ได้

คีย์บอร์ดของคอมพิวเตอร์โดยทั่วไปแล้วจะมีคีย์ตั้งแต่ 101 ถึง 105 คีย์ ประกอบด้วยกลุ่มคีย์อักขระ (Alphanumeric key)ซึ่งเป็นคีย์ตัวอักษรมาตรฐานที่ใช้งานทั่วไป คีย์ตัวเลข(Numeric key) ใช้สำหรับป้อนตัวเลขให้สะดวกขึ้น คีย์ฟังก์ชัน(Function key) เช่น F1 ถึง F12 ใช้สำหรับงานเฉพาะอย่างที่โปรแกรมกำหนดขึ้น คีย์เคลื่อนย้ายเคอร์เซอร์ใช้ย้ายเคอร์เซอร์ไปยังตำแหน่งต่าง ๆ บนจอภาพ และคีย์พิเศษ เช่น Esc, PrintScreen เป็นต้น

1. คีย์ตัวอักขระ ได้แก่ A – Z, a – z, Shift, Ctr, Alt เป็นต้น
2. คีย์ตัวเลข ได้แก่ 0 – 9 และ +, -, *, /
3. คีย์ฟังก์ชัน ได้แก่ F1 – F12
4. คีย์ย้ายเคอร์เซอร์ ได้แก่ ขึ้น, ลง, ซ้าย, ขวา
5. คีย์พิเศษ ได้แก่ Esc, PrintScreen, Scrool Lock, Pause



นอกจากนี้ยังมีคีย์บอร์ดที่ออกแบบมาเป็นพิเศษให้เหมาะกับการใช้งานลักษณะต่าง ๆ เช่น Window key เป็นคีย์บอร์ดที่มีคีย์เสริมเพื่อให้ใช้ระบบปฏิบัติการ Window ได้รวดเร็วขึ้น ซึ่งจะมีคีย์ลัดสำหรับเข้าสู่การทำงานฟังก์ชันต่าง ๆ ของวินโดว์ เช่นควบคุมเมนู เข้าสู่เมนู Start เป็นต้น นอกจากนี้ยังใช้ควบคุมอื่น ๆ เช่นปรับความดังเสียง เปิดเว็บ เป็นต้น คีย์บอร์ดแบบพิเศษอีกแบบหนึ่งได้แก่ คีย์บอร์ดสำหรับเล่นเกม (gaming keyboard) ซึ่งออกแบบมาให้เล่นเกมบนคอมพิวเตอร์ได้สนุกสนานมากขึ้น



คีย์บอร์ดในปัจจุบันจะเชื่อมต่อกับคอมพิวเตอร์ได้หลายวิธี มีขั้วต่อหลายแบบ เช่น แบบพอร์ตอนุกรม (Serial Port) ซึ่งเป็นพอร์ตมาตรฐานที่ใช้กับคอมพิวเตอร์ในยุคแรก ๆ พอร์ต PS/2 มาตรฐานที่ใช้กันอยู่ในปัจจุบัน และแบบพอร์ต USB ร่วมถึงคีย์บอร์ตแบบไร้สาย (Wireless keyboard)

เมาส์ (Mouse)

เมาส์เป็นอุปกรณ์ที่สามารถใช้ชี้ตำแหน่งต่าง ๆ บนจอภาพได้ ในสมัยก่อนการป้อนคำสั่งและข้อมูลให้กับคอมพิวเตอร์จะใช้เพียงคีย์บอร์ดเท่านั้น แต่พอมีการพัฒนาการเชื่อมต่อกับผู้ใช้แบบกราฟิกหรือ GUI และมีโปรแกรมด้านกราฟิกเช่นโปรแกรมวาดภาพเกิดขึ้น เมาส์จึงเป็นสิ่งจำเป็นสำหรับคอมพิวเตอร์นับแต่เวลานั้นเป็นต้นมา โดยเมาส์จะเป็นอุปกรณ์รับข้อมูลที่ใช้ชี้ตำแหน่งบนจอภาพ ใช้เลือกข้อมูลบนจอภาพ เมื่อมีการใช้เมาส์จะมีตัวชี้ขึ้นมาบนหน้าจอคอมพิวเตอร์เรียกว่า ตัวชี้เมาส์ (mouse pointer)

การใช้งานเมาส์หรือการกดปุ่มเมาส์เรียกว่าการ คลิก(Click) การคลิกปุ่มด้านซ้ายมือหนึ่งครั้งจะแสดงถึงการเลือก แต่ถ้าหากเป็นการคลิกสองครั้งติด ๆ กันหมายถึงการสั่งให้ประมวลผล ถ้าหากมีการกดปุ่มเมาส์ทางซ้ายค้างไว้แล้วมีการเลื่อนเมาส์เรียกว่า แดรกกิ้ง(Dragging) การทำงานแบบนี้ทำให้สามารถเลื่อนตำแหน่งของวัตถุบนจอภาพได้ แต่ถ้าหากเป็นการคลิกปุ่มทางขวามักจะเป็นการให้แสดงเมนูพิเศษของโปรแกรมที่กำลังใช้งานอยู่

ปัจจุบันเมาส์เป็นอุปกรณ์หลักสำหรับใช้รับข้อมูลเข้ามาในเครื่องคอมพิวเตอร์ เมาส์ที่ใช้กันอยู่นั้นมีปุ่มใช้งานแบบ 2 ปุ่ม หรือแบบ 3 ปุ่ม และมีพอร์ตเชื่อมต่อให้เลือกหลาย ๆ ประเภท เช่น พอร์ตแบบ PS/2, พอร์ต USB และพอร์ตแบบไร้สาย

หน่วยส่งออก

หลังจากที่คอมพิวเตอร์ประมวลผลแล้วต้องการแสดงสารสนเทศหรือข้อมูลต่าง ๆ ออกมาจะต้องแสดงออกทางหน่วยส่งออก (Output Unit) ของระบบคอมพิวเตอร์ โดยจะนำรหัสที่คอมพิวเตอร์ประมวลผลได้มาแสดงในรูปแบบที่เอาไปใช้งานได้ ซึ่งเป็นรูปแบบที่มนุษย์เข้าใจ การแสดงผลของคอมพิวเตอร์ทำได้หลายรูปแบบ ได้แก่

- แสดงผลเป็นข้อความ (Text) จะให้เอาต์พุตเป็นตัวอักษร ในรูปแบบของรายงาน ข่าวสารต่าง ๆ
- แสดงผลเป็นภาพ (Graphics) โดยให้ข้อมูลเป็นภาพ กราฟ ผังงานต่าง ๆ เป็นต้น
- แสดงผลเป็นเสียง (Audio) อาจเป็นเสียงเพลง เสียงประกอบเกม เป็นต้น
- แสดงผลเป็นวิดีโอ (Video) โ ดยแสดงเป็นไฟล์วิดีโอออกมาทางหน้าจอ

ไม่ว่าการแสดงผลจะออกมาในรูปแบบใดจะต้องมีอุปกรณ์แสดงผล (Output Devices) ต่ออยู่ทางพอร์ตเอาต์พุตของคอมพิวเตอร์ อุปกรณ์แสดงผลที่ได้รับความนิยมมากที่สุดได้แก่ จอภาพ และเครื่องพิมพ์ นอกจากนี้ยังมีเอาต์พุตแบบอื่น ๆ อีกเช่น ลำโพง แฟล็ก โมเด็ม เป็นต้น

จอภาพ

จอภาพหรือมอนิเตอร์(Monitor)เป็นอุปกรณ์เอาต์พุตที่นิยมใช้กันมากที่สุด ผลลัพธ์ที่ได้จากจอภาพจะเรียกว่า ซอฟต์ก๊อปปี้ (soft copy) เนื่องจากเป็นการแสดงผลเพียงชั่วคราว ไม่สามารถเก็บไว้ใช้งานได้ ในอดีสจอภาพจะแสดงได้เพียงสีเดียวที่เรียกว่าจอแบบโมโนโครม (Monochrome) แต่จอภาพในปัจจุบันสามารถแสดงเป็นสีได้ ซึ่งเกิดจากการผสมสีระหว่างสีแดง สีเขียน และสีน้ำเงิน

จอภาพที่พบจะมีอยู่สองประเภทใหญ่ ๆ คือจอ CRT (Cathode Ray Tube) ซึ่งเป็นจอภาพแบบหนา ภาพที่ได้เกิดจากการสร้างภาพของปืนอิเล็กตรอน อีกประเภทหนึ่งคือจอภาพแบบบาง (flat panel display) โดยจอที่นิยมใช้กันมากที่สุดคือจอแอลซีดี (LCD) ซึ่งในปัจจุบันได้มีราคาถูกลงมากแล้ว และในปัจจุบันยังมีการพัฒนาให้สามารถแสดงภาพสองจอพร้อมกันได้อีกด้วย ซึ่งจะทำให้การทำงานที่ซับซ้อนทำได้สะดวกขึ้น

จอภาพแบบ LCD

จอภาพแบบ LCD สองจอ

ในการเลือกจอภาพควรจะคำนึงถึงสิ่งต่อไปนี้คือ ความละเอียดของภาพ(resolution), ขนาด(size), ดอตพิช(dot pitch)

- **ความละเอียดของภาพ** หมายถึงจำนวนจุดหรือพิกเซลบนจอภาพ ถ้าหากมีความละเอียดสูงจะทำให้ภาพคมชัดมากขึ้น ตัวอย่างเช่นจอภาพที่มีความละเอียด 1600 x 1200 เป็นจอภาพที่มีจุดภาพในแนวนอน 1600 จุด มีจุดภาพในแนวตั้ง 1200 จุด โดยทั่วไปแล้วจอภาพสามารถเลือกโหมดการแสดงผลที่มีความละเอียดค่าต่าง ๆ ได้ และได้มีการกำหนดโหมดมาตรฐานดังนี้

มาตรฐาน	ชื่อเต็ม	ความละเอียด	Aspect Ratio
SVGA	Super Video Graphics Array	800 x 600	4 : 3
XGA	Extended Graphics Array	1024 x 768	4 : 3
SXGA	Super XGA	1280 x 1024	5 : 4
WXGA	Wide XGA	1280 x 800 or 1366 x 768	16 : 10 or 16 : 9
UXGA	Ultra XGA	1600 x 1200	4 : 3
WSXGA	Wide Super XGA	1680 x 1050	16 : 10
WUXGA	Wide Ultra XGA	1920 x 1200	16 : 10
WQXGA	Wide Quad XGA	2560 x 1600	16 : 10

ตัวอย่างเช่น ถ้าหากมีคอมพิวเตอร์โน๊ตบุ๊คบอกว่าสามารถแสดงผลในระดับ WXGA ได้ ก็หมายความว่าคอมพิวเตอร์เครื่องนั้นแสดงภาพความละเอียด 1280 x 1024 ได้เป็นต้น

- **ขนาด** ขนาดของจอภาพจะวัดเป็นแนวทะแยงมุม เช่นจอแบบ 19 นิ้ว แบบ 21 นิ้วเป็นต้น
- **ดอตพิช (dit pitch)** บางครั้งจะเรียกว่า pixel pitch หมายถึงระยะห่างระหว่างจุดภาพ จอภาพในปัจจุบันควรเลือกที่มีขนาดน้อยกว่า 0.28 mm

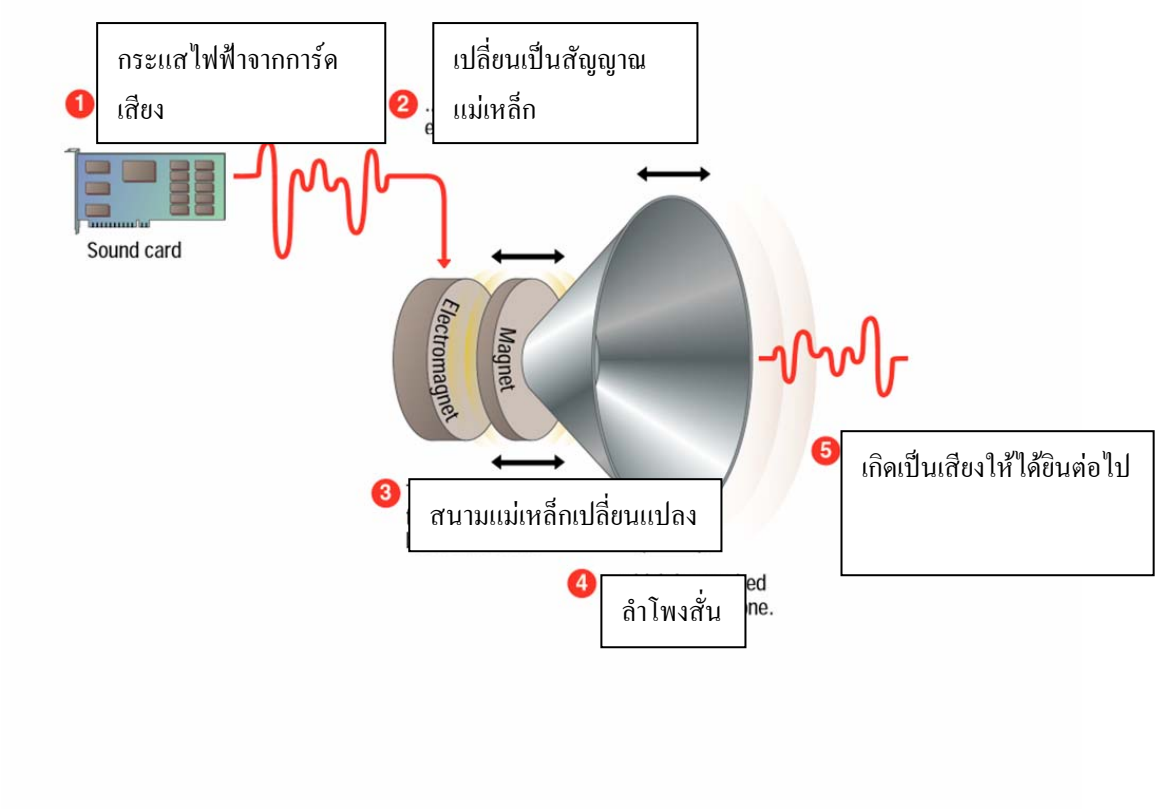
เครื่องพิมพ์

การแสดงผลทางเครื่องพิมพ์เป็นการแสดงผลแบบถาวร สามารถเก็บไว้ใช้ต่อไปได้ จึงเรียกว่าเป็นการแสดงผล ฮาร์ดก๊อปปี้ (hard copy) ซึ่งเป็นการแสดงผลแบบภาวร ในปัจจุบันเครื่องพิมพ์ถูกใช้สำหรับการพิมพ์เอกสารต่าง ๆ รวมทั้งภาพกราฟิกสวย ๆ ภาพที่ถ่ายจากกล้องดิจิตอล จนกลายเป็นอุปกรณ์ที่คอมพิวเตอร์แทบทุกเครื่องจะต้องมี

เครื่องพิมพ์ที่พบมากในปัจจุบันได้แก่ เครื่องพิมพ์แบบกระทบ หรือ Dot-Matrix โดยจะใช้หัวเข็มกระทบผ้าหมึกลงไปบนกระดาษที่ต้องการพิมพ์ เครื่องพิมพ์แบบเลเซอร์ (Laser Printer) และเครื่องพิมพ์แบบฉีดหมึก (Inkjet Printer)

อุปกรณ์ส่งเสียง (Speaker)

ใช้สำหรับแสดงเสียงที่ได้จากโปรแกรมคอมพิวเตอร์ โดยทั่วไปแล้วจะใช้ลำโพง ซึ่งมักจะอยู่ในชุดมัลติมีเดียของคอมพิวเตอร์ เสียงที่ได้ออกมานั้นจะมีการ์ดเสียง (Sound Card) เป็นตัวควบคุม โดยคอมพิวเตอร์จะนำไฟล์เสียงที่ได้มาประมวลผลแล้วให้การ์ดเสียงนี้เปลี่ยนข้อมูลดิจิตอลให้เป็นสัญญาณอนาลอกแล้วขยายสัญญาณที่ได้ให้ไปขับลำโพงให้ดังเป็นเสียงต่อไป



การ์ดเสียงที่ใช้กับคอมพิวเตอร์นั้นยังมีขั้วต่อสำหรับไมโครโฟน ขั้วต่อสำหรับรับสัญญาณเสียงภายนอก และส่งเสียงให้กับลำโพงแบบพิเศษได้อีกด้วย

หน่วยความจำหลัก

หน่วยความจำ (Main Memory) ทำหน้าที่เป็นตัวพักข้อมูลที่ได้รับจากผู้ใช้ เพื่อส่งต่อไปยังหน่วยประมวลผลกลางอีกทีหนึ่ง ในระบบคอมพิวเตอร์มีหน่วยความอยู่หลายส่วน โดยหน่วยความจำหลักเป็นหน่วยความจำที่คอมพิวเตอร์ใช้เก็บข้อมูลและคำสั่งที่ต้องการประมวลผลในงานนั้น ๆ ถ้าหากคอมพิวเตอร์ต้องการทำงานโปรแกรมใด ๆ จะต้องนำโปรแกรมมาเก็บในหน่วยความจำหลักเสมอ อย่างเช่นในฮาร์ดดิสก์ของคอมพิวเตอร์มีข้อมูลและโปรแกรมอยู่มากมาย หากคอมพิวเตอร์ต้องทำโปรแกรมใดก็จะต้องนำโปรแกรมจากฮาร์ดดิสก์ขึ้นมาเก็บในหน่วยความจำหลักก่อนจึงจะทำงานได้

หน่วยความจำหลักของระบบคอมพิวเตอร์จะสร้างจากวงจรอิเล็กทรอนิกส์อยู่ในรูปของวงจรรวม (Integrated circuit) หรือไอซี แบ่งเป็นสองประเภทใหญ่ ๆ คือ แรม(RAM) และ รอม(ROM)

หน่วยความจำแรม

หน่วยความจำแรม (RAM) มาจากคำว่า Random Access Memory) บางครั้งจะเรียกว่าหน่วยความจำแบบชั่วคราว เป็นหน่วยความจำที่ทำงานร่วมกับซีพียู ใช้พักข้อมูลและโปรแกรมชั่วคราว แต่ข้อมูลต่าง ๆ จะหายไปเมื่อปิดเครื่อง หน่วยความจำประเภทนี้คือหน่วยความจำที่เราเห็นบนเครื่องคอมพิวเตอร์ และมักนึกถึงเมื่อพูดถึงหน่วยความจำของเครื่องคอมพิวเตอร์นั่นเอง

รูปที่ 2. ตัวอย่างหน่วยความจำแรม

หน่วยความจำรอม

หน่วยความจำรอม (ROM) มาจากคำว่า (Read Only Memory) เป็นหน่วยความจำที่บันทึกข้อมูลแบบถาวร (เปลี่ยนแปลงไม่ได้) ข้อมูลจะคงอยู่ตลอดแม้ว่าจะปิดเครื่องคอมพิวเตอร์ไปแล้ว หน่วยความจำประเภทนี้จะถูกโปรแกรมมาจากโรงงานที่ผลิต ในเครื่องคอมพิวเตอร์จะใช้หน่วยความจำรอมสำหรับเก็บโปรแกรมที่ใช้เริ่มต้นการทำงานของระบบ หรือโปรแกรมที่ใช้เปิดเครื่อง ควบคุมการทำงานของเครื่องที่เรียกว่า ROM BIOS นั่นเอง

การเก็บข้อมูลในหน่วยความจำนั้นจะใช้หน่วยในการเก็บเป็นไบต์ (Byte) กิโลไบต์(kilobyte) เมกะไบต์ (Megabyte) กิกะไบต์(Gigabyte) และเทราไบต์(Terabyte) โดยขนาดความจุต่าง ๆ มีความสัมพันธ์กันดังนี้

ความจุ	ตัวย่อ	ความจุโดยประมาณ	ความจุเป็นไบต์
กิโลไบต์	kB	หนึ่งพันไบต์	1,024
เมกะไบต์	MB	หนึ่งล้านไบต์	1,048,576
กิกะไบต์	GB	หนึ่งพันล้านไบต์	1,073,741,825
เทราไบต์	TB	หนึ่งล้านล้านไบต์	1,099,511,627,776

หน่วยความจำรอง

หน่วยความจำรอง (Secondary Memory) ใช้สำหรับเก็บคำสั่งและข้อมูลที่คอมพิวเตอร์ยังไม่ใช้ในทันทีทันใด แต่ต้องการใช้ในอนาคต และใช้เก็บข้อมูลที่ได้จากการประมวลผลเพื่อนำไปใช้งานต่อไป หน่วยความจำนี้จะเก็บข้อมูลไว้ได้แม้ว่าจะปิดเครื่องไปแล้ว หน่วยความจำรองมีอยู่หลายชนิดขึ้นกับเทคโนโลยีที่ใช้ในการบันทึกข้อมูล บางประเภทเก็บข้อมูลโดยใช้เทคโนโลยีแบบแม่เหล็ก บางประเภทเก็บข้อมูลโดยใช้เทคโนโลยีทางแสง ตัวอย่างของหน่วยความจำรองได้แก่ ฮาร์ดดิสก์(Harddisk), CD, DVD, หน่วยความจำแบบแฟช (Flash Memory) เป็นต้น

หน่วยความจำรองจะเก็บข้อมูลได้มาก โดยทั่วไปแล้วขนาดความจุของหน่วยความจำรองจะมากกว่าหน่วยความจำหลัก แผ่นดิสก์ซึ่งเป็นหน่วยความจำรองประเภทหนึ่งมีความจุเพียง 1.44 MB แต่หน่วยความจำยุคใหม่ ๆ จะเก็บข้อมูลได้มากขึ้น หน่วยที่ใช้วัดความจุข้อมูลแสดงได้ดังตารางต่อไปนี้

หน่วย	จำนวนไบต์โดยประมาณ	หาได้จาก
Kilobyte (kB)	1 พันไบต์	2^{10} หรือ 1,024
Megabyte(MB)	1 ล้านไบต์	2^{20} หรือ 1,048,576
Gigabyte (GB)	1 พันล้านไบต์	2^{30}
Terabyte(TB)	1 ล้านล้านไบต์	2^{40}
Petabyte(PB)	1000 ล้านล้านไบต์	2^{50}
Exabyte(EB)	1 ล้านล้านล้านไบต์	2^{60}
Zettabyte(ZB)	1000 ล้านล้านล้านไบต์	2^{70}
Yottabyte(YB)	1 ล้านล้านล้านล้านไบต์	2^{80}

2.3 ประเภทของคอมพิวเตอร์

เครื่องคอมพิวเตอร์นั้นสามารถจำแนกได้หลายประเภท ขึ้นกับขนาด ประสิทธิภาพ และลักษณะการใช้งาน โดยทั่วไปแล้วสามารถแบ่งประเภทของคอมพิวเตอร์ได้ดังนี้

คอมพิวเตอร์ส่วนบุคคล หรือพีซี(Personal Computer)

คอมพิวเตอร์ประเภทนี้เป็นคอมพิวเตอร์ที่มีใช้งานกันทั่วไป เป็นคอมพิวเตอร์แบบตั้งโต๊ะที่เหมาะสำหรับใช้งานในบ้าน ในสำนักงาน ราคาไม่แพง คอมพิวเตอร์ประเภทนี้ที่นิยมใช้กันมีอยู่สองตระกูลคือ PC-Compatible ที่มีต้นแบบเป็นคอมพิวเตอร์ของบริษัท IBM และคอมพิวเตอร์ตระกูล Apple คอมพิวเตอร์แบบ PC มีการผลิตออกมาหลายรุ่นหลายแบบ โดยส่วนใหญ่แล้วจะใช้โปรแกรมระบบปฏิบัติการ Windows ส่วนคอมพิวเตอร์ Apple จะใช้โปรแกรมระบบปฏิบัติการของ Macintosh ที่เรียกว่า Mac OS

คอมพิวเตอร์ PC ใช้ระบบปฏิบัติการ Windows

คอมพิวเตอร์ Apple ใช้ระบบปฏิบัติการ Mac OS

คอมพิวเตอร์โน้ตบุ๊ก (Notebook Computer)

เป็นคอมพิวเตอร์ส่วนบุคคลขนาดเล็กที่มีน้ำหนักเบา สะดวกกับการเคลื่อนย้ายไปยังที่ต่าง ๆ คอมพิวเตอร์แบบนี้อาจเรียกได้ว่าเป็น Mobile Computer สามารถใช้พลังงานไฟฟ้าทั่วไปหรือพลังงานจากแบตเตอรี่ได้ ในปัจจุบันคอมพิวเตอร์ประเภทนี้จะมีประสิทธิภาพสูงไม่แพ้คอมพิวเตอร์แบบ PC แต่หากเทียบกับ PC ที่มีประสิทธิภาพเท่ากันแล้ว คอมพิวเตอร์แบบโน้ตบุ๊กจะมีราคาสูงกว่า

คอมพิวเตอร์แบบ Notebook

คอมพิวเตอร์แบบพกพา (Handheld Computer)

เป็นคอมพิวเตอร์ขนาดเล็กที่เหมาะสำหรับพกพาไปที่ต่าง ๆ เนื่องจากเครื่องมีขนาดเล็กจึงไม่เหมาะที่จะออกแบบคีย์บอร์ดไว้บนตัวเครื่อง การใช้คอมพิวเตอร์ประเภทนี้จะใช้ปากกาที่เรียกว่า สไตลัส (Stylus) เป็นอุปกรณ์สำหรับป้อนข้อมูล คอมพิวเตอร์ประเภทนี้สามารถใช้งานพื้นฐานทั่วไปได้ รับส่ง mail และใช้ในการสื่อสารได้ เครื่องคอมพิวเตอร์ประเภทนี้จะรวมถึงคอมพิวเตอร์แบบ PDA (Personal Digital Assistant) หรือ Pocket PC ที่ใช้กันทั่วไปด้วย ปัจจุบันคอมพิวเตอร์ประเภทนี้ยังมีกล้องถ่ายภาพติดมาบนตัวเครื่องด้วย

ตัวอย่าง Hand-held คอมพิวเตอร์ โดยจะใช้ Stylus เป็นอุปกรณ์อินพุต

เครื่องคอมพิวเตอร์เซิร์ฟเวอร์

เครื่องคอมพิวเตอร์ Server เป็นเครื่องคอมพิวเตอร์ที่มีขนาดใกล้เคียงกับคอมพิวเตอร์ทั่วไป แต่จะมีความสามารถสูงกว่ามาก คอมพิวเตอร์ประเภทนี้จะใช้เป็นเครื่องให้บริการกับคอมพิวเตอร์ PC ต่าง ๆที่ต่ออยู่ในเครือข่าย และยังใช้เป็นเครื่องให้บริการบนเว็บอีกด้วย ขนาดและประสิทธิภาพของคอมพิวเตอร์เซิร์ฟเวอร์นี้มีหลายรุ่น ขึ้นกับการใช้งานว่าจะให้บริการกับเครื่องคอมพิวเตอร์อื่น ๆ หลายเครื่องหรือไม่

เครื่องเซิร์ฟเวอร์ขนาดเล็ก และขนาด

ใหญ่ที่ใช้กับระบบเครือข่ายขนาดใหญ่

คอมพิวเตอร์เมนเฟรม (Mainframe Computer)

เป็นคอมพิวเตอร์ขนาดใหญ่ ที่ต้องการประมวลผลข้อมูลจำนวนมากด้วยความเร็วสูง มีหน่วยความจำขนาดใหญ่ คอมพิวเตอร์ประเภทนี้จะนิยมใช้ในองค์กรที่มีผู้ใช้จำนวนมาก เช่นระบบธนาคารขนาดใหญ่ ระบบธุรกิจขนาดใหญ่ ระบบการจองตั๋วเครื่องบิน เป็นต้น ในการใช้งานคอมพิวเตอร์แบบเมนเฟรมมักจะไม่ใช้เครื่องเมนเฟรมนี้เพียงเครื่องเดียว แต่จะมีการต่อทำงานร่วมกับคอมพิวเตอร์อื่น ๆ ด้วย โดยให้เครื่องเมนเฟรมเป็นเครื่องหลักในการประมวลผล ส่วนคอมพิวเตอร์อื่น ๆ จะใช้เป็นตัวป้อนและแสดงข้อมูลทั่วไป

คอมพิวเตอร์แบบเมนเฟรมที่มีขนาดใหญ่ ใช้

สำหรับประมวลผลข้อมูลจำนวนมาก

ซูเปอร์คอมพิวเตอร์ (Supercomputer)

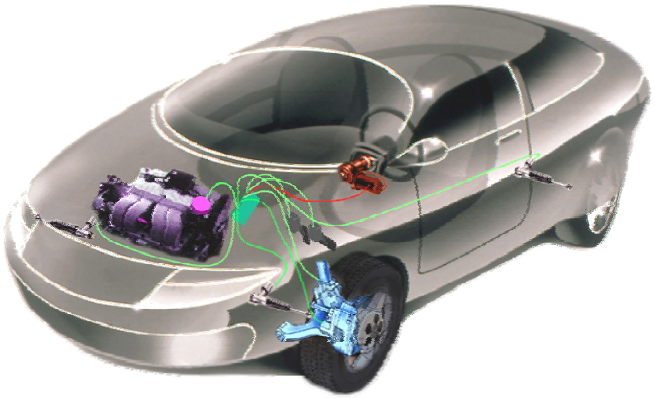
เครื่องคอมพิวเตอร์ประเภทนี้จัดว่าเป็นคอมพิวเตอร์ที่มีประสิทธิภาพการทำงานสูงที่สุด สามารถทำงานได้มากกว่าพันล้านคำสั่งในหนึ่งวินาที มีน้ำหนักหลายตัน สามารถเชื่อมต่อกับคอมพิวเตอร์ PC ทั่วไปได้จำนวนมาก คอมพิวเตอร์ประเภทนี้จะใช้ในงานที่ต้องการประมวลผลกับข้อมูลจำนวนมากด้วยความเร็วสูง ๆ เช่น งาน

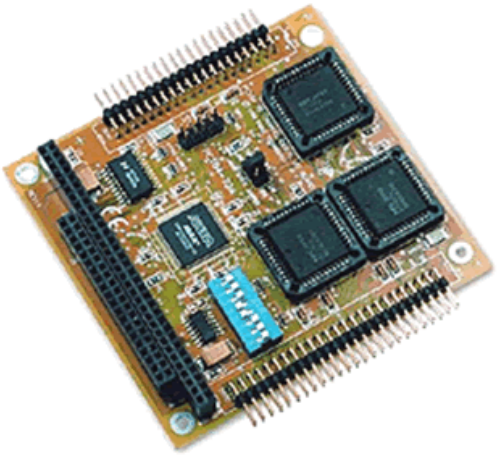
ทางด้านการพยากรณ์อากาศ การคำนวณต่าง ๆ ที่มีความซับซ้อน และงานคอมพิวเตอร์สำหรับการออกแบบในอุตสาหกรรมเป็นต้น

เครื่องซูเปอร์คอมพิวเตอร์

คอมพิวเตอร์ฝังตัว (Embedded Computer)

เป็นคอมพิวเตอร์ที่ได้รับความนิยมอย่างมากในปัจจุบัน โดยจะเป็นระบบประมวลผลที่ฝังตัวอยู่ในอุปกรณ์ต่าง ๆ เพื่อควบคุมการทำงานของอุปกรณ์นั้น ๆ ซึ่งถ้าหากมองภายนอกแล้วจะไม่พบว่ามีคอมพิวเตอร์เชื่อมต่ออยู่ ระบบประเภทนี้จะพบในเครื่องมืออิเล็คทรอนิกส์สมัยใหม่ ตู้เย็น เครื่องปรับอากาศ หรือในรถยนต์ที่ควบคุมการทำงาน การจุดระเบิด ระบบเบรค ด้วยระบบคอมพิวเตอร์





ตัวอย่างคอมพิวเตอร์ฝังตัวที่ควบคุมการทำงานของรถ

แบบฝึกหัดท้ายบท

ตอนที่ 1 จงกาเครื่องหมาย ถูก หรือข้อที่ถูกและกาเครื่องหมาผิดหน้าข้อที่ผิด

…ถูก…. 1. หน่วยประมวลผลกลางเป็นอุปกรณ์อิเล็กทรอนิกส์ที่ใช้ในการประมวลผลคำสั่งและข้อมูล

…ผิด…..2. ฮาร์ดดิสก์จัดว่าเป็นหน่วยความจำหลัก ส่วนแรมจัดว่าเป็นหน่วยความจำสำรอง

…ถูก…..3. คอมพิวเตอร์ที่มีขนาดใหญ่ มักมีความเร็วสูงกว่าคอมพิวเตอร์ที่มีขนาดเล็ก

…ถูก…. 4. หน่วยความจำรอมของคอมพิวเตอร์เป็นหน่วยความจำที่เก็บโปรแกรม BIOS และอ่านข้อมูลได้อย่างเดียว

…ผิด…. 5. ในอนาคตไมโครโปรเซสเซอร์แบบสารกึ่งตัวนำจะทำงานด้วยเร็วสูงขึ้นเรื่อย ๆ

…ถูก…. 6. การแบ่งประเภทของคอมพิวเตอร์จะแบ่งตามขนาด ประสิทธิภาพ และลักษณะการใช้งาน

…ผิด…. 7. หากไม่มีหน่วยความจำหลักคอมพิวเตอร์ก็ยังคงทำงานได้

…ถูก…. 8. ขนาดความจุของหน่วยความจำจะบอกเป็นจำนวนไบต์

…ถูก…. 9. เทคโนโลยีสารกึ่งตัวนำเป็นเทคโนโลยีที่ทำให้คอมพิวเตอร์มีขนาดเล็กลง

…ผิด…. 10. หน่วยส่งออกทำหน้าที่เปลี่ยนรหัสที่มนุษย์เข้าใจเป็นรหัสที่คอมพิวเตอร์เข้าใจ

ตอนที่ 2 จงเลือกคำตอบที่ถูกที่สุดเพียงคำตอบเดียว

1. ความสามารถของคอมพิวเตอร์ขึ้นอยู่กับอะไร
   ก. การทำงานภายในของซีพียู ข. ประสิทธิภาพของโปรแกรมที่ใช้งาน
   ค. ความเร็วของซีพียู **ง. ถูกทุกข้อ**
2. หน่วยความจำแบบใดที่จัดว่าเป็นหน่วยความจำหลัก
   ก. หน่วยความจำประเภทแรม ข. ฮาร์ดดิสก์
   ค. Flash Memory ง. ถูกทุกข้อ
3. ไมโครโปรเซสเซอร์ถูกเรียกว่าอะไรในคอมพิวเตอร์
   ก. ชิพ (chip) **ข. หน่วยประมวลผลกลางหรือซีพียู**
   ค. เมนบอร์ต ง. การ์ดแสดงผล
4. คอมพิวเตอร์ขนาดเล็กที่ซ่อนอยู่ในอุปกรณ์อิเล็กทรอนิกส์คือข้อใด
   ก. MiniComputer ข. MicroComputer
   ค. Embedded Computer ง. Microprocessor
5. การบอกขนาดของหน่วยเก็บข้อมูลในหน่วยความจำจะบอกในหน่วยใด
   ก. รีจีสเตอร์ **ข. กิโลไบต์**
   ค. แอดเดรส ง. บิต
6. ข้อใดเป็นโปรแกรมระบบปฏิบัติการของเครื่อง Apple
   ก. Mac OS ข. Windows
   ค. DOS ง. ถูกทุกข้อ
7. หน่วยความจำประเภทใดเมื่อปิดเครื่องแล้วข้อมูลที่เก็บอยู่จะหายไป
   ก. แรม ข. รอม
   ง. ฮาร์ดดิสก์ ง. หน่วยความจำรอง

8. จำนวนจุดในแนวนอนและแนวตั้งของจอภาพจะบอกลักษณะใดของภาพ

ก. ความละเอียดของภาพ ข. สีที่จะแสดงได้

ค. ขนาดของภาพ ง. ถูกทุกข้อ

9. ข้อใดคือสิ่งที่ควบคุมการทำงานของซีพียู

ก. ข้อมูลที่รับทางอุปกรณ์รับเข้า **ข. คำสั่ง**

ค. ตัวเลขที่ป้อนทางคีย์บอร์ต ง. การใช้เมาส์

10. หน่วยแสดงผลจะอยู่บนคอมพิวเตอร์ประเภทใด

ก. mobile computer
ข. notebook computer
ค. desktop personal computer
ง. ถูกทุกข้อ

บทที่ 2
ภาษาคอมพิวเตอร์และการพัฒนาโปรแกรม

เครื่องคอมพิวเตอร์เป็นอุปกรณ์อิเล็กทรอนิกส์ประเภทหนึ่ง การให้เครื่องคอมพิวเตอร์ทำงานได้นั้น จะต้องป้อนคำสั่งให้กับมันและต้องเป็นคำสั่งที่เครื่องคอมพิวเตอร์เข้าใจ การนำคำสั่งมาเรียงต่อกันให้ทำงานอย่างใดอย่างหนึ่งเรียกว่าโปรแกรม เมื่อโปรแกรมถูกป้อนเข้าไปในเครื่องคอมพิวเตอร์แล้วมันจะทำงานทีละคำสั่ง สำหรับการใช้คำสั่งสั่งงานให้คอมพิวเตอร์ทำงานนั้นจะต้องใช้ภาษาที่คอมพิวเตอร์สามารถเข้าใจได้ ภาษาที่คอมพิวเตอร์เข้าใจเรียกว่าภาษาเครื่อง (Machine Language) ซึ่งเป็นรหัสเลขฐานสองเมื่อมีการป้อนภาษานี้เข้าไปในเครื่องคอมพิวเตอร์ รหัสเลขฐานสองจะถูกเปลี่ยนเป็นสัญญาณทางไฟฟ้าที่คอมพิวเตอร์เข้าใจ

แต่ถ้ามนุษย์ต้องการป้อนโปรแกรมให้กับคอมพิวเตอร์เป็นเลขฐานสองนั้นจะทำได้ยากมาก เพราะเป็นภาษาที่มนุษย์เข้าใจได้ยากจึงได้มีการออกแบบตัวอักษรภาษาอังกฤษให้แทนคำสั่งรหัสเลขฐานสองเหล่านั้น เรียกว่ารหัสนีโมนิก (mnemonic) ภาษาคอมพิวเตอร์ที่ใช้รหัสนีโมนิกในการเขียนโปรแกรมเรียกว่าภาษาแอสเซมบลี (Assembly Language) ต่อมาได้มีการพัฒนาชุดคำสั่งภาษาต่าง ๆ ให้มีความใกล้เคียงกับภาษาที่มนุษย์เข้าใจเรียกว่าภาษาระดับสูง (High-level Language) ซึ่งมีอยู่หลายภาษาได้แก่ ภาษาซี ภาษาเบสิก ภาษาปาสคาล สำหรับภาษาแอสเซมบลีเป็นภาษาที่ทำงานได้เร็วเพราะเข้าถึงหน่วยประมวลผลได้เร็วที่สุดเราเรียกภาษานี้ว่าภาษาระดับต่ำ (Low-level Language)

ดังนั้นในการเขียนโปรแกรมคอมพิวเตอร์ ผู้เขียนโปรแกรมจะต้องมีตัวแปลภาษาที่จะแปลภาษาโปรแกรมให้ได้เป็นรหัสทางดิจิตอลที่คอมพิวเตอร์สามารถรู้จักและทำงานได้

2.1 ชนิดของภาษาคอมพิวเตอร์

เนื่องจากคอมพิวเตอร์ประมวลผลด้วยระบบดิจิตอล การทำงานภายในคอมพิวเตอร์จะควบคุมด้วยสัญญาณไฟฟ้าที่มีลักษณะเปิดปิด หรือแทนด้วย “0” และ “1” เป็นระบบเลขฐานสอง ซึ่งภาษาที่คอมพิวเตอร์สื่อสารกันภายในเครื่องด้วยระบบเลขฐานสองนั้น เรียกว่า ภาษาเครื่อง (Machine Language) แต่ถ้าหากมนุษย์ต้องจำเลขฐานสองเป็นจำนวนมากเพื่อควบคุมเครื่องคอมพิวเตอร์นั้นจะเป็นไปได้ยาก ดังนั้นจึงมีการพัฒนาภาษาที่เป็นกลาง คือ เป็นภาษาคำสั่งที่ใกล้เคียงกับคำที่มนุษย์รู้จักกันดี แล้วแปลงกลับไปเป็นเลขฐานสองให้คอมพิวเตอร์อีกครั้งหนึ่ง วิธีนี้จะทำให้โปรแกรมคอมพิวเตอร์ทำได้ง่ายขึ้น ดังนั้นจึงมีการคิดค้นภาษาคอมพิวเตอร์ออกมามากมาย โดยสามารถแบ่งได้เป็น 5 ระดับ ดังนี้

- **ภาษาเครื่อง (Machine Language)**

ก่อนปี ค.ศ. 1952 คอมพิวเตอร์จะใช้ภาษาระดับต่ำสุด คือ การใช้เลขฐานสองในการแทนคำสั่งและข้อมูลต่าง ๆ จึงทำให้นักเขียนและพัฒนาโปรแกรมในยุคนั้นต้องกำหนดชุดตัวเลขขึ้นมาในการใช้แทนคำสั่ง จากนั้นจึงนำชุดตัวเลขมาใช้ในการเขียนโปรแกรม ซึ่งเป็นวิธีที่ยุ่งยากมาก

ตัวอย่างเช่น การเขียนคำสั่งเพื่อทำการบันทึกข้อมูล 61H (61 ฐานสิบหก) เก็บไว้ในหน่วยความจำภายในซีพียูที่เป็นรีจิสเตอร์ AL โดยใช้ภาษาเครื่องของคอมพิวเตอร์รุ่น x86/IA-32 จะเขียนดังนี้

10110000 01100001

- **ภาษาแอสเซมบลี (Assembly Language)**

ต่อจากนั้นในปี ค.ศ. 1952 ได้มีการพัฒนาโปรแกรมภาษาตัวใหม่ที่ช่วยให้จดจำคำสั่งได้ง่ายขึ้น ชื่อว่า ภาษาแอสเซมบลี (Assembly Language) โดยการใช้ตัวอักษรภาษาอังกฤษมาแทนคำสั่งที่เป็นเลขฐานสอง ซึ่งเราเรียกอักษรสัญลักษณ์ที่ใช้แทนคำสั่งนี้ว่า นีโมนิกโค้ด (Mnemonic code) แต่ถึงอย่างไรก็ยังจัดให้ภาษาแอสเซมบลีอยู่ในภาษาระดับต่ำ

ตัวอย่าง การใช้ภาษาเครื่องของคอมพิวเตอร์รุ่น x86/IA-32 เขียนคำสั่งให้บันทึกข้อมูล 61H เก็บไว้ในรีจีสเตอร์ AL ภายในซีพียู สามารถทำได้ดังนี้

1011000 01100001

ถ้าหากเป็นการนำภาษาแอสเซมบลีมาเขียนเป็นโปรแกรม จะทำให้คำสั่งโปรแกรมดูและจำได้ง่ายขึ้น โดยมีรูปแบบการเขียนดังนี้

```
MOV     AL , 61H
```

เหตุผลที่ทำให้คอมพิวเตอร์สามารถรับคำสั่งที่เป็นรหัสนีโมนิกได้ เพราะมีการใช้ แอสเซมเบอร์ (Assembler) ซึ่งเป็นตัวแปลภาษานีโมนิกให้เป็นรหัสภาษาเครื่อง จึงทำให้คอมพิวเตอร์ทำงานตามคำสั่งที่ป้อนเข้าไปได้ สำหรับตัวอย่างนี้ตัวแอสเซมเบอร์จะแปลคำสั่ง MOV AL , 61H ให้ไปเป็นรหัสเลขฐานสองที่คอมพิวเตอร์เข้าใจนั่นเอง

- **ภาษาระดับสูง (High-level Language)**

ในปี ค.ศ. 1960 มีการพัฒนาภาษาโปรแกรมเป็น ภาษาระดับสูงโดยใช้คำภาษาอังกฤษมาสั่งงานและควบคุมคอมพิวเตอร์ เรียกภาษาสูงในยุคนี้ว่า ภาษายุคที่สาม (Third-generation Language) ซึ่งทำให้ยุคนั้นมีการหันมาใช้คอมพิวเตอร์กันมากยิ่งขึ้น

ในยุคนี้มีภาษาระดับสูงเกิดขึ้นมากกมาย เช่น ภาษา BASIC, COBOL, FORTRAN และภาษา C ซึ่งแต่ละภาษาก็มีความแตกต่างกันไป เพราะภาษานั้น ๆ จะต้องมีตัวแปลภาษา หรือคอมไพล์เลอร์ ที่จะแปลเป็นภาษาที่ทำให้เครื่องคอมพิวเตอร์ทำงานได้ตรงตามโปรแกรมที่เขียนเอาไว้

ตัวอย่าง ถ้าหากต้องการเขียนโปแกรมภาษา C โดยสั่งให้พิมพ์คำว่า “Test” ซึ่งใช้คำสั่งที่เป็นคำภาษาอังกฤษเข้าใจง่าย ๆ ดังนี้

```
#include <stdio.h>
main() {
        printf(“  Test “);
}
```

เมื่อต้องการให้คอมพิวเตอร์ทำงานจะต้องนำตัวแปลภาษาซี หรือ ซีคอมไพล์เลอร์มาแปลงคำภาษาอังกฤษด้านบนให้เป็นเลขฐานสองที่คอมพิวเตอร์เข้าใจอีกทีหนึ่ง

- **ภาษาระดับสูงมาก (Very high-level Language)**

เป็นยุคที่พัฒนาโปรแกรมขึ้นมาเป็นระดับที่ 4 หรือ ภาษายุคที่สี่ (Fourth-generation Language) หรือ 4GLs เป็นการพัฒนาคุณสมบัติของภาษาได้แตกต่างจากยุคก่อนได้อย่างชัดเจน โดยใช้หลักการเขียนโปรแกรมแบบไม่ใช้โพรซีเยอร์ (Procedural Language) ซึ่งแบบเก่าเราต้องเขียนโปรแกรมสร้างโพรซีเยอร์แต่ละตัวเพื่อไว้ให้ทำงานในด้านต่าง ๆ แต่ยุคนี้โปรแกรมการทำงานในส่วนต่าง ๆ จะถูกสร้างไว้พร้อมแล้ว ดังนั้นนักพัฒนาโปรแกรมไม่จำเป็นต้องเขียนโปรแกรมขึ้นใหม่ทั้งหมด ซึ่งสามารถเรียกชุดการทำงานที่มีอยู่แล้วมาสร้างเป็นโปรแกรมใหม่ที่ต้องการได้ทันที

ตัวอย่างเช่น ถ้าหากต้องการให้แสดงรายชื่อพนักงานที่มีอยู่ในกรุงเทพ จะสามารถเขียนโปรแกรมได้สั้นมาก เมื่อเทียบกับโปรแกรมแบบเก่า โดยใช้คำสั่งสั้น ๆ ดังนี้

```
SELECT  First  Name, Last Name
FROM  Employees
WHERE  City  =  “Bangkok”
```

จากตัวอย่างเป็นการเขียนโปรแกรมที่สั้นมาก เนื่องจากโปรแกรมภาษาที่ใช้มีความใกล้เคียงกับภาษาของมนุษย์มากยิ่งขึ้น หากเทียบกับการเขียนโปรแกรมด้วยภาษา COBOL จะพบว่าต้องเขียนโปรกรมหลายร้อยบรรทัดจึงจะได้ผลลัพธ์เหมือนกัน

ข้อดีของภาษาในยุคที่สี่

- การเขียนโปรแกรมจะมุ่งหวังที่จะได้ผลลัพธ์ของงานเป็นหลักว่าอยากได้อะไร ไม่สนใจวิธีการทำมากนัก
- ส่งเสริมต่อการพัฒนาเนื้องาน เนื่องจากสามารถเขียนและแก้ไขโปรแกรมได้ง่าย
- ลดเวลาในการอบรมและพัฒนาผู้เขียนโปรแกรม เพราะไม่เชี่ยวชาญในการเขียนโปรแกรมก็สามารถทำได้
- ผู้เขียนโปรแกรมไม่จำเป็นต้องรู้และศึกษาถึงโครงสร้างของโปรแกรม และระบบฮาร์ดแวร์ของเครื่อง

ในยุคที่สี่ได้มีการพัฒนาภาษาเพื่อใช้เรียกดูข้อมูลจากฐานข้อมูล ซึ่งรู้จักกันในชื่อ ภาษาเรียกค้นข้อมูล (Query Language) เพื่อช่วยให้เราสามารถเรียกแสดงข้อมูลและติดต่อฐานข้อมูลได้สะดวกยิ่งขึ้น เพราะก่อนหน้านี้ถ้าจะทำการจัดเก็บและเรียกแสดงข้อมูลจะต้องมีการวางแผนไว้ล่วงหน้า หากมีการเรียกแสดงข้อมูลนอกเหนือจากแผนที่กำหนดไว้ ก็จะต้องใช้เวลาในการรอนาน แต่ถ้าเป็นการใช้ภาษาเรียกค้นข้อมูลก็จะช่วยให้เราเรียกดูได้เร็วขึ้น และภาษาเรียกค้นข้อมูลที่เป็นมาตรฐานซึ่งรู้จักกันดี คือ SQL (Structured Query Language)

- ภาษาธรรมชาติ (Natural Language)

เป็นภาษาโปรแกรมที่พัฒนามาถึงระดับที่ 5 เรียกว่า ภาษายุคที่ห้า (Fifth generation Language) หรือ 5GLs เหตุผลที่มาของภาษาธรรมชาติเนื่องจากว่าเป็นภาษาที่ใกล้เคียงกับภาษาธรรมชาติของมนุษย์ ไม่ต้องสนใจ

เรื่องไวยากรณ์ของภาษา เพียงแค่ผู้ใช้พิมพ์คำสั่งลงไป โดยผู้ใช่แต่ละคนอาจจะใช้คำศัพท์ รูปประโยคที่ต่างกัน จากนั้นคอมพิวเตอร์จะแปลความหมายและทำงานตามคำสั่งเหล่านั้นเอง แต่ถ้ามีคำสั่งที่ไม่เข้าใจหรือไม่แน่ใจ ก็จะมีการแจ้งกลับมาให้เรายืนยันความถูกต้อง ซึ่งการแปลความหมายของคำสั่งในภาษาธรรมชาติจะใช้ ระบบฐานความรู้ (Knowledge base system) ที่มีความใกล้เคียงกับการใช้ภาษาของมนุษย์

ตัวอย่างเช่น การใช้ภาษา SQL เรียกค้นข้อมูลโดยให้แสดงรายชื่อพนักงานที่มีอยู่ในกรุงเทพสามารถเขียนได้ดังนี้

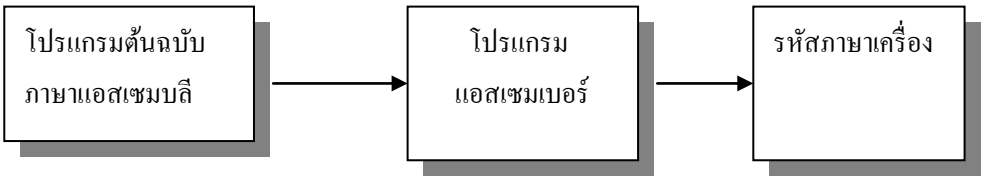
```
SELECT  First Name , Last Name
FROM  Employees
WHERE  City = “BANGKON”
```

แต่ถ้าเป็นการเขียนโปรแกรมด้วยภาษาธรรมชาติ เราจะเขียนโปรแกรมได้ใกล้เคียงกับภาษามนุษย์มากขึ้น ดังนี้

```
TELL  ME  THE  NAMES  OF  EMPLOYEES  IN  BANGKOK
```

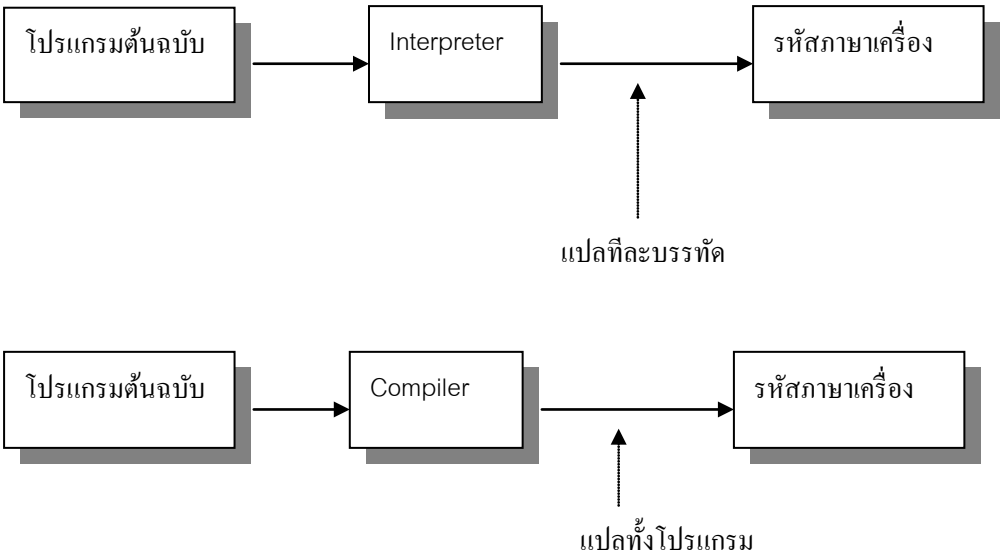
ในการเขียนโปรแกรมคอมพิวเตอร์ไม่ว่าจะเขียนด้วยภาษาระดับสูงหรือภาษาระดับต่ำ เราจะต้องแปลงภาษาเหล่านั้นให้เป็นรหัสภาษาเครื่องที่คอมพิวเตอร์เข้าใจเสียก่อน คอมพิวเตอร์จึงจะทำงานได้ ตามที่ได้กล่าวมาแล้วว่าภาษาคอมพิวเตอร์เป็นการนำชุดคำสั่งแต่ละคำสั่งมาต่อกันให้คอมพิวเตอร์ทำงาน การเขียนชุดคำสั่งนี้ไม่ว่าจะเขียนด้วยภาษาอะไรจะเรียกว่าโปรแกรมต้นฉบับ (Source Program) หรือรหัสต้นฉบับ (Source Code) จากนั้นเราจะต้องแปลงให้เป็นภาษาเครื่องที่คอมพิวเตอร์ทำงานได้เรียกว่าเอ็กซิคิ้วโปรแกรม (Executable Program)

การเขียนโปรแกรมด้วยภาษาแอสเซมบลีจะต้องใช้ตัวแปลภาษาให้เป็นภาษาเครื่อง ตัวแปลนี้เรียกว่า แอสเซมเบอร์ (Assembler) ขั้นตอนการแปลสามารถเขียนได้ดังรูปที่ 2.1



รูปที่ 2.1 ขั้นตอนการแปลงภาษาแอสเซมบลีเป็นภาษาเครื่อง

สำหรับการเขียนโปรแกรมด้วยภาษาระดับสูงจะมีวิธีในการแปลสองประเภทคือ การแปลคำสั่งทีละคำสั่งให้เครื่องทำงานทีละคำสั่ง จากนั้นจึงแปลคำสั่งบรรทัดต่อไปเช่นการเขียนโปรแกรมด้วยภาษาเบสิก ตัวที่แปลภาษาประเภทนี้เรียกว่าอินเตอร์พรีเตอร์ (Interpreter) การทำงานของตัวอินเตอร์พรีเตอร์นี้จะแปลความหมายของคำสั่งทีละคำสั่ง ถ้าไม่พบข้อผิดพลาดเครื่องจะทำคำสั่งที่แปลได้ แต่ถ้าพบข้อผิดพลาดจะหยุดทำงานและแจ้งข้อผิดพลาดออกมา ส่วนการแปลคำสั่งอีกแบบหนึ่งเรียกว่า คอมไพเลอร์ (Compiler) โดยมันจะมองโปรแกรมต้นฉบับทั้งหมด และแปลให้เป็นรหัสภาษาเครื่องถ้าพบข้อผิดพลาดก็จะแจ้งออกมา ทำให้โปรแกรมทำงานได้เร็วเพราะเครื่องไม่ต้องแปลอีกเมื่อจะทำคำสั่งถัดไป ขั้นตอนการแปลภาษาทั้งสองประเภทนี้แสดงได้ดังรูปที่ 2.2



รูปที่ 2.2 ขั้นตอนการแปลภาษาโปรแกรม

2.2 ภาษาคอมพิวเตอร์สำหรับพัฒนาโปรแกรม

ในปัจจุบันมีภาษาคอมพิวเตอร์ที่ใช้สำหรับพัฒนาโปรแกรมมากมาย บางภาษาแม้ว่าจะมีมานานแล้วแต่ก็ยังได้รับความนิยมอยู่เนื่องจากมีการพัฒนามาอย่างยาวนาน จึงมีเครื่องมือช่วยให้เขียนโปรแกรมได้ง่ายขึ้นมากมาย ภาษาแต่ละภาษาจะมีโครงสร้างของภาษาต่างกัน มีความสามารถเด่น ๆ ต่างกัน และแต่ละภาษาก็ใช้สภาพแวดล้อมของเครื่องคอมพิวเตอร์ต่างกันด้วย ภาษาคอมพิวเตอร์ที่นิยมใช้ในการเขียนโปรแกรมได้แก่

ภาษาเบสิก (BASIC)

ภาษาเบสิกเป็นภาษาคอมพิวเตอร์ระดับสูง เกิดขึ้นเมื่อปี ค.ศ. 1963 ที่มหาวิทยาลัย Dartmouth College ต่อมาได้ถูกนำมาใช้ในคอมพิวเตอร์ทั่วไปในปี ค.ศ. 1980 คำว่า Basic ย่อมาจากคำว่า Beginner's Allpurpose Symbolic Instruction Code ภาษานี้เหมาะสำหรับผู้เริ่มต้นเขียนโปรแกรมเนื่องจากเป็นรูปแบบคำสั่งที่ง่าย แต่ความสามารถจะน้อยกว่าภาษาอื่น ๆ เนื่องจากเป็นภาษาที่พัฒนามานานแล้ว

```
10 PRINT "HELLO WIKIPEDIA!"
20 GOTO 10
```

ข้อดี : เป็นภาษาระดับสูงที่มีรูปแบบคำสั่งใช้งานได้ง่าย นำไปประยุกต์สร้างโปรแกรมได้ทั่วไป ทั้งงานด้านธุรกิจและวิทยาศาสตร์ เหมาะกับผู้ที่เริ่มต้นฝึกเขียนโปรแกรม

ข้อเสีย : ถูกพัฒนาและใช้ในยุคแรกของเครื่องไมโครคอมพิวเตอร์ จึงทำให้ประสิทธิภาพของคำสั่งมีน้อยกว่าภาษาอื่น ๆ และเนื่องจากรูปแบบของการเขียนโปรแกรมจะไม่เป็นโครงสร้าง ดังนั้นจึงไม่เหมาะกับการเขียนโปรแกรมที่มีการเชื่อมต่อกับระบบฐานข้อมูล

ภาษาฟอร์แทรน (FORTRAN)

เป็นภาษาระดับสูงเกิดขึ้นเมื่อปี ค .ศ. 1950 คำว่า FORTRAN ย่อมาจากคำว่า FORmular TRANslator ภาษานี้เป็นภาษาที่มีประสิทธิภาพสูงในการคำนวณ เหมาะสำหรับการเขียนโปรแกรมประยุกต์ทางคณิตศาสตร์ที่ทำงานบนเครื่องเมนเฟรม แต่ในปัจจุบันได้มีคอมไพล์เลอร์หลายตัวที่พัฒนาขึ้นสำหรับแปลภาษานี้บนเครื่องคอมพิวเตอร์ทั่วไป

ในอดีสวิชาการเขียนโปรแกรมคอมพิวเตอร์ ภาษานี้เป็นวิชาบังคับสำหรับนักศึกษาทางด้านวิทยาศาสตร์และวิศวกรรมศาสตร์ที่ต้องการนำคอมพิวเตอร์มาช่วยคำนวณในด้านต่าง ๆ เนื่องจากภาษานี้มีความสามารถในการคำนวณดีกว่าภาษาอื่น ๆ

```
program  HelloWorld
      write (*,*)  'Hello, world! '
end  program HelloWorld
```

ข้อดี : เป็นภาษาระดับสูงที่ถูกพัฒนาขึ้นเพื่อใช้ในงานที่ซับซ้อน มีประสิทธิภาพในงานคำนวณ ซึ่งถูกนำไปใช้สร้างโปรแกรมประยุกต์ด้านวิทยาศาสตร์และวิศวกรรมศาสตร์ ที่ทำงานบนเครื่องเมนเฟรม

ข้อเสีย : เนื่องจากเป็นภาษาที่พัฒนาขึ้นเพื่อใช้บนเมนเฟรม จึงทำให้ต้องมีการปรับคำสั่งมากมายเพื่อให้เหมาะสมกับการนำมาใช้สร้างโปรแกรมประยุกต์บนเครื่องไมโครคอมพิวเตอร์ จึงเป็นเหตุให้หมดความนิยมเมื่อมีการพัฒนาภาษาเบสิกมาใช้บนไมโครคอมพิวเตอร์ที่มีการใช้งานกันอย่างแพร่หลาย

ภาษาโคบอล (COBOL)

ภาษานี้เกิดจากความร่วมมือของรัฐบาลสหรัฐ กับองค์กรธุรกิจ และมหาวิทยาลัยต่าง ๆ ถูกประกาศใช้อย่างเป็นทางการเมื่อปี ค .ศ. 1960 ภาษานี้มีชื่อเต็มว่า Business Oriented Language เป็นภาษาที่ใช้เขียนโปรแกรมแบบโครงสร้าง (Structure Program) เหมาะสำหรับการพัฒนาโปรแกรมประยุกต์ทางธุรกิจ การจัดเก็บข้อมูล งานทางด้านบัญชี และการเชื่อมต่อคอมพิวเตอร์ภายในองค์กร

```
IDENTIFICATION  DIVISION.
 Program-Id. Hello-World.
*
 ENVIRONMENT  DIVISION.
*
 DATA  DIVISION.
*
 PROCEDURE  DIVISION.
 Para1.
       DISPLAY  "Hello, world.".
*
       Stop  Run.
```

ข้อดี : เป็นภาษาที่นิยมใช้ในงานธุรกิจ ได้แก่ การจัดเก็บข้อมูล งานบัญชี การเงินและสินค้าคงคลัง เพราะเป็นภาษาที่มีการเขียนโปรแกรมเชิงโครงสร้าง จึงสามารถประยุกต์ให้ทำงานเชื่อมต่อกับฐานข้อมูลได้ ปัจจุบันได้พัฒนาเป็นการเขียนโปรแกรมเชิงวัตถุ (OOP)

ข้อเสีย : ผู้พัฒนาโปรแกรมต้องศึกษาโครงสร้างของโปรแกรม เพื่อการใช้คำสั่งได้ถูกต้อง

ภาษาปาสคาล (PASCAL)

ภาษานี้เกิดขึ้นเมื่อปี 1970 ชื่อของภาษาเป็นการตั้งชื่อตามนักคณิตศาสตร์ที่ประดิษฐ์เครื่องคำนวณในยุคแรก ที่ชื่อ Blaise Pascal ภาษานี้เป็นภาษาระดับสูงที่ใช้เขียนโปรแกรมเชิงโครงสร้างได้ ตัวแปลภาษาที่ได้รับความนิยมอย่างมากคือ โปรแกรมเทอร์โบปาสคาล (Turbo Pascal) ของบริษัทบอร์แลนด์ ในปัจจุบันในประเภทไทยได้ใช้โปรแกรมนี้ในการสอนการโปรแกรมเบื้องต้นให้กับนักเรียนนักศึกษาทั่วไป

```
Program   HelloWorld(output);
Begin
      Writeln('Hello, World!')
End.
```

ข้อดี : เป็นภาษาที่เขียนเชิงโครงสร้าง เหมาะกับการศึกษาสำหรับผู้เริ่มต้นเขียนโปรแกรม นอกจากนี้ คำสั่งได้ถูกออกแบบให้ทำงานอย่างมีประสิทธิภาพ จึงสามารถพัฒนาไปใช้ในงานทั่วไป งานด้านธุรกิจ วิทยาศาสตร์และวิศวกรรม

ข้อเสีย : เป็นภาษาที่ยังคงใช้งานยาก เมื่อเทียบกับการเขียนโปรแกรมเชิงวัตถุ

ภาษาซี C

ภาษานี้พัฒนาขึ้นในห้องปฏิบัติการเบลล์ (Bell Laboratory) ของบริษัท เอทีแอนด์ที ในปี ค .ศ. 1970 เพื่อใช้บนระบบปฏิบัติการยูนิกซ์ (Unix) ต่อมาได้มีตัวแปลภาษาออกมาหลายตัว และได้ถูกใช้อย่างแพร่หลายบนเครื่องคอมพิวเตอร์ทั่วไป ภาษานี้เป็นภาษาที่มีความยืดหยุ่นสูง สามารถทำงานบนระบบปฏิบัติการต่าง ๆ ได้เป็นอย่างดี สามารถใช้ควบคุมฮาร์ดแวร์ได้โดยตรง แต่ชุดคำสั่งจะมีกฎเกณฑ์และรายละเอียดต่าง ๆ จำนวนมาก

```
#include  "stdio.h"
int   main(void)
{
     printf("Hello, world\n");
     return  0;
}
```

ข้อดี : เป็นภาษาที่มีความยืดหยุ่นสามารถทำงานบนเครื่องคอมพิวเตอร์ส่วนบุคคล และระบบปฏิบัติการต่าง ๆ ได้ นอกจากนี้ยังสามารถเข้าถึงฮาร์ดแวร์ของคอมพิวเตอร์ได้โดยตรง

ข้อเสีย : คำสั่งของภาษาจะไม่เหมือนคำศัพท์ภาษาอังกฤษโดยตรง จึงอาจจะจดจำยากขึ้น และวิธีการใช้คำสั่งจะมีกฎเกณฑ์รายละเอียดจำนวนมาก จึงไม่เหมาะกับผู้เริ่มต้นเขียนโปรแกรม

ภาษาซีพลัสพลัส (C++)

ภาษานี้พัฒนาต่อมาจากภาษาซี โดยเพิ่มการเขียนโปรแกรมแบบ Class เข้าไป ทำให้ภาษาซีมีความสามารถในการทำงานสูงขึ้น สามารถนำมาเขียนโปรแกรมเชิงวัตถุ (Object-oriented programming) ได้ ทำให้ภาษานี้ได้รับความนิยมอย่างสูง แต่จะไม่เหมาะสำหรับผู้เริ่มต้นเขียนโปรแกรมเนื่องจากโครงสร้างของภาษามีความซับซ้อนมากขึ้น

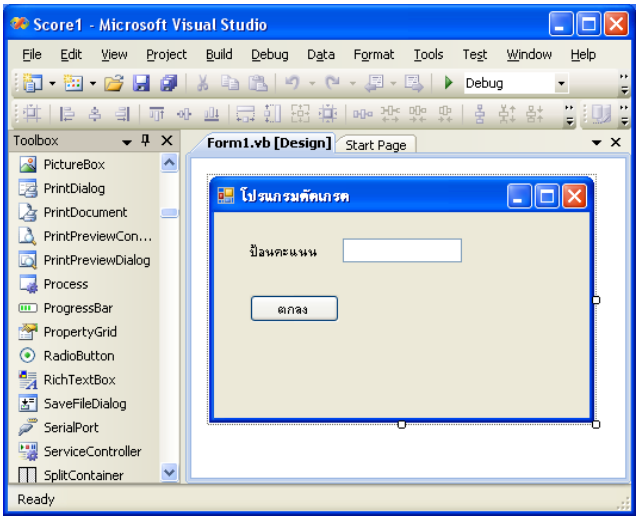
```
#include <iostream>
int  main()
{
      std::cout << “Hello  World!” << std::endl;
      return  0;
}
```

ข้อดี : เป็นภาษาที่มีรูปแบบการเขียนโปรแกรมเชิงวัตถุ สามารถทำงานเข้าถึงการทำงานของฮาร์ดแวร์ได้โดยตรง จึงเหมาะกับการพัฒนาโปรแกรมประยุกต์ และได้รับความนิยมเป็นอย่างสูง

ข้อเสีย : เป็นภาษาระดับสูง และมีรูปแบบการเขียนโปรแกรมที่ซับซ้อนขึ้น ไม่เหมาะสำหรับผู้เริ่มต้นเขียนโปรแกรม

วิชวลเบสิก (VISUAL BASIC)

ภาษานี้พัฒนาขึ้นโดยบริษัทไมโครซอฟต์ ชุดคำสั่งต่าง ๆ จะคล้ายกับภาษา BASIC เดิม และเป็นภาษาที่ได้รับความนิยมอย่างมากในการเขียนโปรแกรมบนระบบปฏิบัติการวินโดว์ เนื่องจากผู้เขียนโปรแกรมสามารถสร้างหน้าจอในการติดต่อกับผู้ใช้ได้ง่าย ปัจจุบันภาษานี้ถูกนิยมใช้ในการเขียนโปรแกรมขนาดใหญ่ ๆ จำนวนมาก ภาษานี้เหมาะสำหรับผู้พัฒนาโปรแกรมแต่ไม่เหมาะผู้ที่จะเริ่มต้นเขียนโปรแกรม



ข้อดี : เป็นการพัฒนาโปรแกรมที่เหมาะกับการทำงานบนระบบปฏิบัติการแบบ GUI เช่น Windows เพราะมีเครื่องมือที่ใช้ในการสร้างกราฟิก เหมาะกับการสร้างโปรแกรมขนาดใหญ่

ข้อเสีย : ไม่เหมาะกับผู้เริ่มต้นเขียนโปรแกรม เพราะจะเหมาะกับผู้ที่เคยมีพื้นฐานการเขียนโปรแกรมภาษาอื่นมาก่อน การเขียนโปรแกรมมีความซับซ้อน

ภาษาจาวา (JAVA)

ภาษานี้เกิดขึ้นในปี ค .ศ. 1990 โดยบริษัท Sun Microsystem ที่พัฒนาให้เป็นภาษาสำหรับการเขียนโปรแกรมเชิงวัตถุ ภาษาจาวานี้สามารถทำงานได้บนเครื่องคอมพิวเตอร์ทุกระบบ เนื่องจากเวลาคอมไพล์ออกมาแล้วจะได้ข้อมูลแบบ ไบต์โค้ด (Bytecode) เครื่องคอมพิวเตอร์ที่จะใช้ภาษานี้จะต้องติดตั้ง Java Virtual Machine ก่อนเพื่อให้โปรแกรมทำงานได้ ปัจจุบันภาษานี้ได้ถูกพัฒนามาหลายรูปแบบ มีทั้งการเขียนโปรแกรมบนระบบเครือข่าย การเขียนโปรแกรมบนโทรศัพท์มือถือเป็นต้น

```
// Hello.java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

ข้อดี : เป็นโปรแกรมภาษาที่สามารถทำงานได้ทุกระบบ จึงเหมาะกับการพัฒนาโปรแกรม หรือเกมที่ต้องการเข้าถึงผู้ใช้ได้ในทุกระบบ

ข้อเสีย : เป็นมาตรฐานที่มีความซับซ้อน การพัฒนาและการศึกษาโปรแกรมจึงยากกว่าภาษาอื่น ๆ

การพัฒนาโปรแกรมเพื่อประยุกต์ใช้ในงานแต่ละด้าน จำเป็นอย่างยิ่งที่จะต้องเลือกใช้ภาษาที่เหมาะสม และเนื่องจากภาษาคอมพิวเตอร์มีเป็นจำนวนมาก ดังนั้นจำเป็นจะต้องพิจารณาสิ่งต่าง ๆ ในการเลือกใช้ภาษาคอมพิวเตอร์ดังนี้

- **ภาษามาตรฐานที่ใช้ในองค์กร**

การพัฒนาโปรแกรมไว้ใช้ในองค์กร โดยส่วนใหญ่จะยึดภาษาใดภาษาหนึ่งไว้เป็นหลัก เพราะไม่ต้องเสียเวลาศึกษาโครงสร้างของภาษาในการเขียนโปรแกรมใหม่ ทำให้สะดวกต่อการเขียน การดูแล และการบริหารระบบ

ตัวอย่างเช่น องค์กรที่ใช้ภาษา JAVA เป็นมาตรฐาน ก็จะมุ่งเน้นใช้ภาษา JAVA ในการสร้างโปรแกรมทั้งหมด เพราะนักพัฒนาโปรแกรมแต่ละคนไม่ได้เก่งในการเขียนโปรแกรมได้ทุกภาษา นอกจากนี้ควรเป็นภาษาที่สามารถสร้างบุคลากรมาทำงานเขียนโปรแกรมและบริหารได้อย่างต่อเนื่องได้ ตัวอย่างเช่น องค์กรที่ใช้ภาษา JAVA เริ่มต้นในการเขียนโปรแกรม ต่อจากนั้นโปรแกรมเมอร์ชุดเก่าลาออกไป หากองค์กรไม่สามารถหาโปรแกรมเมอร์ที่ชำนาญภาษา JAVA มาแทนที่ได้ ก็จะทำให้งานนั้นหยุดชะงักไป

- **คุณสมบัติและความเหมาะสม**

แต่ละภาษาจะมีคุณสมบัติและความเหมาะสมกับงานเฉพาะด้าน เช่น งานด้านธุรกิจ งานด้านบัญชี งานด้านวิทยาศาสตร์ และงานด้านวิศวกรรม ตัวอย่างเช่น การเขียนโปรแกรมด้านบัญชีจะนิยมใช้ภาษา COBOL เป็นต้น

- **การทำงานร่วมกับโปรแกรมอื่น**

การเขียนโปรแกรมการทำงานในบางด้าน อาจจะต้องทำงานร่วมกับโปรแกรมอื่นด้วย เช่น การเขียนโปรแกรมสินค้าคงคลัง เราจำเป็นต้องเขียนโปรแกรมทั้งส่วนที่ติดต่อกับผู้ใช้และส่วนติดต่อกับฐานข้อมูล ดังนั้นการเลือกใช้ภาษาที่รองรับกัน จะช่วยให้โปรแกรมทำงานได้อย่างมีประสิทธิภาพ

ตัวอย่างเช่น หากเรามีระบบฐานข้อมูลบนโปรแกรม Microsoft Access และเมื่อมีการพัฒนาโปรแกรมส่วนติดต่อกับผู้ใช้ใหม่ โดยจะต้องทำงานร่วมกับฐานข้อมูลบน Access ที่เคยมีอยู่ งานลักษณะนี้ควรเลือกใช้ Visual Basic ซึ่งจะทำงานเข้ากันมากกว่าการเลือกใช้ JAVA เพราะ Visual Basic ถูกออกแบบให้สามารถรองรับการติดต่อกับฐานข้อมูลบน Access ได้เป็นอย่างดี

- **การทำงานร่วมกับระบบอื่น ๆ**

โปรแกรมประยุกต์บางตัว อาจจะมีเป้าหมายให้สามารถทำงานข้ามระบบได้ เช่น โปรแกรมรับรายการอาหารจากลูกค้า และการออกใบเสร็จเก็บเงินลูกค้าของภัตตาคาร ซึ่งจะต้องทำงานร่วมกันระหว่างโปรแกรมรับรายการสินค้าบนระบบเครื่อง PDA ของพนักงานขาย และส่งข้อมูลไปยังระบบเครื่องคอมพิวเตอร์เซิร์ฟเวอร์ที่จะบันทึกรายการอาหารของลูกค้าแต่ละโต๊ะ ไปคำนวณราคา และพิมพ์ในเสร็จรับเงินออกมา

2.3 ขั้นตอนการพัฒนาโปรแกรม

การเขียนโปรแกรมคอมพิวเตอร์ให้ทำงานได้ตามที่เราต้องการนั้น ผู้เขียนโปรแกรมจะต้องรู้ว่าจะให้โปรแกรมทำอะไร มีข้อมูลอะไรที่ต้องให้กับโปรแกรมบ้าง และต้องการ เอาต์พุตอย่างไรจากโปรแกรมรวมทั้งรูปแบบการแสดงผลด้วย ผู้ที่ทำการเขียนโปรแกรมจะต้องทราบถึงขั้นตอนวิธีการของการแก้ปัญหาของโปรแกรมด้วยว่าจะต้องทำอย่างไร โดยเขียนเป็นลำดับขั้นตอนขึ้นมาก่อนแล้วจดบันทึกเอาไว้ จากนั้นจึงนำลำดับขั้นตอนที่เขียนขึ้นมาพัฒนาเป็นโปรแกรม ถ้าหากผู้เขียนโปรแกรมไม่ได้วางแผนขั้นตอนการทำงานต่าง ๆ ไว้ก่อน หากต้องการปรับปรุงแก้ไขโปรแกรมในภายหลังจะทำให้เสียเวลามากในศึกษาโปรแกรมก่อนที่จะทำการแก้ไข ถ้าหากโปรแกรมมีความซับซ้อนไม่มากการศึกษาโปรแกรมเพื่อแก้ไขปัญหาอาจไม่นานนัก แต่ถ้าหากโปรแกรมมีความซับซ้อนมากจะทำให้ขั้นตอนการศึกษาปัญหายิ่งใช้เวลามากขึ้นไปด้วย โดยทั่วไปแล้วขั้นตอนการพัฒนาโปรแกรมแบ่งได้ดังนี้

1. กำหนดและวิเคราะห์ปัญหา (Problem Definition and Problem Analysis)
2. เขียนผังงานและซูโดโค้ด (Pseudocoding)
3. เขียนโปรแกรม (Programming)
4. ทดสอบและแก้ไขโปรแกรม (Program Testing and Debugging)
5. ทำเอกสารและบำรุงรักษาโปรแกรม (Program Documentation and Maintenance)

การกำหนดและวิเคราะห์ปัญหา

ขั้นตอนนี้เป็นขั้นตอนแรกสุดที่นักเขียนโปรแกรมจะต้องทำ การให้คอมพิวเตอร์แก้ปัญหาต่างๆ ให้เรานั้นเราจะต้องมีแนวทางที่แก้ไขปัญหาที่เหมาะสมให้กับคอมพิวเตอร์ เพื่อให้การทำงานเป็นไปอย่างมีประสิทธิภาพ ถ้าหากผู้ที่เขียนโปรแกรมไม่สามารถทำความเข้าใจกับปัญหาที่ต้องการแก้ไขได้ การนำคอมพิวเตอร์มาใช้ในการแก้ปัญหาต่าง ๆ ก็ไม่สามารถทำได้ การกำหนดและวิเคราะห์ปัญหามีขั้นตอนย่อย ๆ ดังนี้

1. **กำหนดขอบเขตของปัญหา** โดยกำหนดรายละเอียดให้ชัดเจนว่าจะให้คอมพิวเตอร์ทำอะไร ตัวแปร ค่าคงที่ที่ต้องใช้เป็นลักษณะใด ถ้าหากเราไม่กำหนดขอบเขตของปัญหาจะทำให้คอมพิวเตอร์ตัดสินใจได้ยากว่าข้อมูลต่าง ๆ ที่เกิดขึ้นนั้นถูกหรือผิด

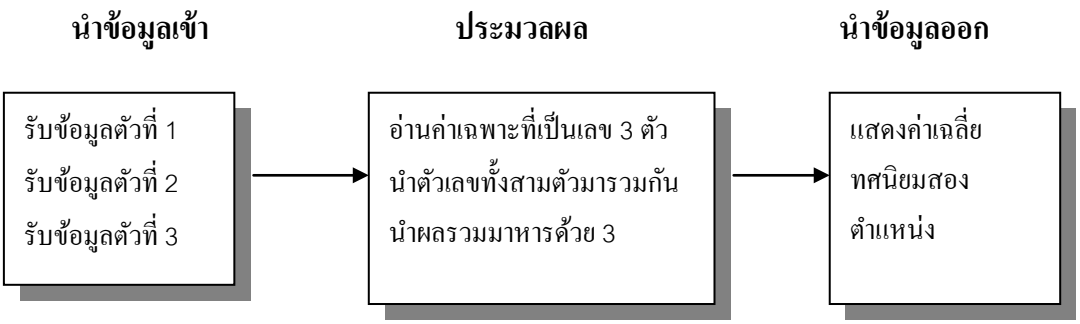
2. **กำหนดลักษณะของข้อมูลเข้าและออกจากระบบ** (Input/Output Specification) โดยต้องรู้ว่าข้อมูลที่จะส่งเข้าไปเป็นอย่างไร มีอะไรบ้าง เพื่อให้โปรแกรมทำการประมวลผลและแสดงผลลัพธ์ เช่นการรับค่าจากคีย์บอร์ต การใช้เมาส์ การกำหนดปุ่มต่าง ๆ ลักษณะการแสดงผลทางหน้าจอว่าจะให้มีรูปร่างอย่างไร โดยคำนึงถึงผู้ใช้เป็นหลักในการออกแบบโปรแกรม ตัวอย่างเช่นถ้าหากต้องการรับข้อมูลเข้าไปประมวลผล ก็ต้องพิจารณาว่าข้อมูลนั้นเป็นตัวอักษรหรือตัวเลข ถ้าเป็นตัวเลขก็ต้องพิจารณาต่อว่าเป็นเลขจำนวนเต็มหรือทศนิยม เอาต์พุตที่แสดงออกทางจอภาพจะให้แสดงทศนิยมกี่ตำแหน่งเป็นต้น

3. **กำหนดวิธีการประมวลผล** (Process Specification) โดยต้องรู้ว่าจะให้คอมพิวเตอร์ประมวลผลอย่างไรจึงได้ผลลัพธ์ตามต้องการ

ตัวอย่างที่ 2.1 ถ้าหากต้องการออกแบบโปรแกรมให้คอมพิวเตอร์รับค่าข้อมูล 3 ค่า และแสดงค่าเฉลี่ยทางจอภาพ เราอาจกำหนดและวิเคราห์ปัญหาได้ดังนี้

1. รับข้อมูลจากคีย์บอร์ต
   1.1 รับข้อมูลเฉพาะที่เป็นตัวเลขมาเก็บในตัวแปร
   1.2 ถ้าข้อมูลเท่ากับ 0 ให้รับใหม่
2. หาค่าเฉลี่ย
   2.1 รวมค่าทุกค่าที่รับมาเข้าด้วยกัน
   2.2 นำค่าผลรวมที่ได้หารด้วย 3
   2.3 นำค่าผลลัพธ์ไปเก็บในตัวแปร
3. แสดงผลลัพธ์ทางจอภาพ
   3.1 แสดงคำว่าค่าเฉลี่ยเท่ากับ
   3.2 แสดงผลลัพธ์โดยมีทศนิยมสองตำแหน่ง

จะเห็นว่าเราจะนำปัญหามาแจกแจงย่อยว่าจะต้องทำอะไรบ้าง โดยข้อมูลที่รับเข้าไปคือตัวเลขสามตัว การประมวลผลคือการหาค่าเฉลี่ย ส่วนเอาต์พุตคือการพิมพ์ผลลัพธ์ เราสามารถเขียนการทำงานของระบบได้ดังแผนภาพในรูปที่ 2.3



รูปที่ 2.3 ขั้นตอนการทำงานของระบบ

การเขียนผังงานและซูโดโค้ด

หลังจากที่ได้วิเคราะห์ปัญหาแล้ว ขั้นตอนต่อไปจะต้องใช้เครื่องมือช่วยในการออกแบบโปรแกรม ซึ่งยังไม่ได้เขียนเป็นโปรแกรมจริง ๆ แต่จะช่วยให้เขียนโปรแกรมได้ง่ายขึ้น และทำให้ผู้อื่นนำโปรแกรมของเราไปพัฒนาต่อได้ง่ายขึ้น โดยเขียนเป็นลำดับขั้นตอนการทำงานของโปรแกรมที่เรียกว่าอัลกอริทึม (Algorithm) ซึ่งจะแสดงขั้นตอนการแก้ปัญหา โดยใช้ประโยคที่ชัดเจนไม่คลุมเครือ และมีรายละเอียดการทำงานพอสมควรเพียงพอที่จะนำไปเขียนเป็นโปรแกรมให้ทำงานจริง โดยอัลกอริทึมนั้นอาจเขียนให้อยู่ในรูปของรหัสจำลองหรือซูโดโค้ด (Pseudo-code) หรือเขียนเป็นผังงาน (Flowchart) ก็ได้ โดยซูโดโค้ดจะเป็นคำอธิบายขั้นตอนการทำงานของโปรแกรม เป็นคำย่อไม่มีรูปแบบเฉพาะตัว โดยแต่ละส่วนจะเป็นแนวทางในการเขียนโปรแกรมซึ่งทำให้เขียนโปรแกรมเป็นภาษาต่าง ๆ ได้ง่ายขึ้น ส่วนผังงานจะใช้สัญลักษณ์ต่าง ๆ แทนการทำงานและทิศทางของโปรแกรม

การเขียนโปรแกรม

หลังจากที่ผ่านขั้นตอนทั้งสองแล้ว ขั้นต่อไปจะต้องเขียนเป็นโปรแกรมเพื่อให้คอมพิวเตอร์สามารถประมวลผลได้ โดยเปลี่ยนขั้นตอนการทำงานให้อยู่ในรูปรหัสภาษาคอมพิวเตอร์ การเขียนโปรแกรมจะต้องเขียนตามภาษาที่คอมพิวเตอร์เข้าใจโดยอาจใช้ภาษาระดับสูง หรือระดับต่ำซึ่งสามารถเลือกได้หลายภาษา การเขียนโปรแกรมแต่ละภาษาจะต้องทำตามหลักไวยากรณ์ (syntax) ที่กำหนดไว้ในภาษานั้น นอกจากนี้การเลือกใช้ภาษาจะต้องพิจารณาถึงความถนัดของผู้เขียนโปรแกรมด้วย

การทดสอบและแก้ไขโปรแกรม

หลังจากเขียนโปรแกรมจะต้องทดสอบความถูกต้องของโปรแกรมที่เขียนขึ้น หาจุดผิดพลาดของโปรแกรมว่ามีหรือไม่ และตรวจสอบจนไม่พบที่ผิดอีก จุดผิดพลาดของโปรแกรมนี้เรียกว่าบัก (Bug) ส่วนการแก้ไขข้อผิดพลาดให้ถูกต้องเรียกว่า ดีบัก (debug) โดยทั่วไปแล้วข้อผิดพลาดจากการเขียนโปรแกรมจะมีสองประเภทคือ

1. การเขียนคำสั่งไม่ถูกต้องตามหลักการเขียนโปรแกรมภาษานั้น ๆ ซึ่งเรียกว่า Syntax Error หรือ Coding Error ข้อผิดพลาดประเภทนี้เรามักพบตอนแปลภาษาโปรแกรมเป็นรหัสภาษาเครื่อง
2. ข้อผิดพลาดทางตรรก หรือ Logic Error เป็นข้อผิดพลาดที่โปรแกรมทำงานได้ แต่ผลลัพธ์ออกมาไม่ถูกต้อง

ทำเอกสารและบำรุงรักษาโปรแกรม

ขั้นตอนนี้จะทำให้ผู้ใช้สามารถใช้งานโปรแกรมได้อย่างมีประสิทธิภาพ และสะดวกในการตรวจสอบข้อผิดพลาดโดยเขียนเป็นเอกสารประกอบโปรแกรมขึ้นมา โดยทั่วไปแล้วแบ่งออกเป็นสองประเภทคือ

1. คู่มือการใช้ หรือ User Document หรือ User guide ซึ่งจะอธิบายการใช้โปรแกรม
2. คู่มือโปรแกรมเมอร์ หรือ Program Document หรือ Technical Reference ซึ่งจะอำนวยความสะดวกในการแก้ไขโปรแกรมและพัฒนาโปรแกรมในอนาคต โดยจะมีรายละเอียดต่างๆ เกี่ยวกับโปรแกรม เช่นชื่อโปรแกรม การรับข้อมูล การพิมพ์ผลลัพธ์ขั้นตอนต่าง ๆ ในโปรแกรม เป็นต้น

ส่วนการบำรุงรักษาโปรแกรม (Maintainance) เป็นการที่ผู้เขียนโปรแกรมจะต้องคอยตรวจสอบการใช้โปรแกรมจริง เพื่อแก้ไขข้อผิดพลาดซึ่งอาจเกิดขึ้นในภายหลัง รวมทั้งพัฒนาโปรแกรมให้ทันสมัยอยู่เสมอเมื่อเวลาผ่านไป

แบบฝึกหัดท้ายบท

ตอนที่ 1 จงเลือกคำตอบที่ถูกต้องที่สุดเพียงหนึ่งข้อ

1. ข้อใดเป็นลักษณะของภาษาระดับสูง
   ก.ทำงานได้โดยไม่ต้องมีโปรแกรมระบบ ข . มีภาษาใกล้เคียงกับภาษามนุษย์
   ค. เป็นภาษาคอมพิวเตอร์ยุคใหม่ ง . เป็นภาษาที่ใช้กับงานขั้นสูง
2. ภาษาคอมพิวเตอร์ภาษาใดที่ต้องใช้ตัวอินเทอร์พรีเตอร์เป็นตัวแปลภาษา
   ก. ภาษา Basic ข. ภาษาปาสคาล
   ค. ภาษาซี ง . ภาษาฟอร์แทน
3. โปรแกรมแอสเซมเบอร์คืออะไร
   ก. ตัวแปลภาษา Basic ข. ตัวแปลภาษาซี
   ค. ตัวแปลภาษาแอสเซมบลี ง . ถูกทุกข้อ
4. ข้อใดจัดว่าเป็นซอฟต์แวร์ระบบ
   ก. โปรแกรมฆ่าไวรัส ข . โปรแกรม Window

ค. โปรแกรมพิมพ์รายงาน ง . โปรแกรมภาษา

5. จุดประสงค์ของตัวแปลภาษาคืออะไร

ก. ตรวจสอบการทำงานของระบบ ข . ใช้แปลภาษาหนึ่งเป็นอีกภาษาหนึ่ง

ค. แปลโปรแกรมต้นฉบับให้ทำงานได้ ง . ใช่แปลคำศัพท์

6. ในการเขียนโปรแกรมภาษาซีจะต้องเก็บโปรแกรมต้นฉบับเป็นนามสกุลอะไร

ก. .OBJ ข. .BAS

ค. .C ง. .CPP

7. เมื่อต้องการพัฒนาโปรแกรมจะต้องทำสิ่งใดก่อน

ก. วิเคราะห์ปัญหา ข . เขียนซูโดโค้ด

ค. เขียนโปรแกรม ง . เลือกภาษาที่ต้องใช้เขียน

8. โปรแกรมดูภาพยนตร์จัดว่าเป็นโปรแกรมประเภทใด

ก. ซอฟต์แวร์ระบบ ข . ซอฟต์แวร์อรรถประโยชน์

ค. ซอฟต์แวร์สำเร็จรูป ง . ซอฟต์แวร์ภาษา

9. การแปลภาษาคอมพิวเตอร์เป็นรหัสภาษาเครื่องที่มีการแปลทีละบรรทัดเรียกว่าอะไร

ก. คอมไพเลอร์ ข . อินเทอร์พรีเตอร์

ค. แอสเซมเบอร์ ง . รันไทม์

10. ถ้าหากเขียนโปรแกรมสำหรับงานคำนวณ แต่ผลลัพธ์ไม่ถูกต้องตามต้องการ ข้อผิดพลาดนี้เรียกว่าอะไร

ก. bug ข. syntax error

ค. logic error ง. coding error

ตอนที่ 2 จงทำเครื่องหมาย √ หน้าข้อที่ถูก และเครื่องหมาย × หน้าข้อที่ผิด

.....1. ภาษาระดับต่ำคือภาษาที่ใช้กับคอมพิวเตอร์รุ่นที่ใช้ไมโครโปรเซสเซอร์ก่อน 80486

.....2. ภาษาระดับสูงคือภาษาคอมพิวเตอร์ที่คล้ายกับภาษาพูด

.....3. คอมพิวเตอร์จะทำงานได้จะต้องแปลภาษาคอมพิวเตอร์ให้เป็นภาษาเลขฐานสองเสียก่อน

.....4. ระบบปฏิบัติการ Windows จัดว่าเป็นโปรแกรมระบบ

.....5. สิ่งที่ได้จากการคอมไพล์ภาษาซีคือไบต์โค๊ด

ตอนที่ 3 จงตอบคำถามต่อไปนี้

1. ภาษาเครื่องคืออะไร แตกต่างจากภาษาระดับสูงอย่างไร

.....

……

2. จงบอกชื่อภาษาระดับสูงที่ใช้ทั่วไป

……

……

3. แอสเซมเบอร์ (Assembler) คืออะไร

……..

……..

4. การคอมไพล์คืออะไร

……..

……..

5. โปรแกรมต้นฉบับ (Source Code) คืออะไร มีประโยชน์อย่างไร

……..

……..

6. โปรแกรมเอนกประสงค์ต่างจากโปรแกรมระบบปฏิบัติการอย่างไร

……..

……..

7. ซูโดโค้ดจะช่วยผู้พัฒนาโปรแกรมได้อย่างไร

……..

……..

8. การดีบักคืออะไร

……

……

บทที่ 3

การกำหนดและวิเคราะห์ปัญหา

วงจรการออกแบบและพัฒนาโปรแกรมนั้นเป็นสิ่งที่ต้องศึกษาทำความเข้าใจ เพื่อให้โปรแกรมที่สร้างขึ้นมีลักษณะที่ดี โดยมีขั้นตอนการพัฒนาโปรแกรมทั้งหมด 6 ขั้นตอนในการเตรียมงานเขียนโปรแกรม ขั้นตอนแรกของการพัฒนาโปรแกรมคือการกำหนดและวิเคราะห์ปัญหา เพื่อเป็นการพิจารณาว่างานที่ต้องการนั้นต้องการนั้นจะทำอะไร ผลลัพธ์เป็นอย่างไร ใช้ข้อมูลใดเป็นอินพุตและมีวิธีการประมวลผลอย่างไร โดยต้องคำนึงถึงความเป็นไปได้ที่คอมพิวเตอร์สามารถทำงานได้ตามที่ได้วางแผนไว้ด้วย การวิเคราะห์งานที่ถูกต้องจะทำให้สามารถเขียนโปรแกรมได้เร็ว ตัวอย่างเช่นถ้าหากต้องการคำนวณหาค่าจ้างล่วงเวลาของพนักงานตามจำนวนชั่วโมง ผู้เขียนโปรแกรมจะต้องทราบว่าค่าจ้างคำนวณได้จากสูตร

ค่าจ้าง = Hours * PayRate

หรือถ้าต้องการเขียนโปรแกรมเพื่อคำนวณหาปริมาตรของทรงกระบอก ผู้เขียนโปรแกรมจะต้องทราบว่าการหาปริมาตรคำนวณได้จากสูตร

ปริมาตรทรงกระบอก = π x รัศมี^{2} x ความสูง

3.1 วงจรการพัฒนาโปรแกรม

การเขียนโปรแกรมให้ทำงานงานหนึ่งนั้น สามารถเขียนได้หลายวิธี โดยแต่ละวิธีในการเขียนอาจได้ผลลัพธ์ออกมาเหมือนกัน แม้ว่าจะมีวิธีการเขียนหรือขั้นตอนการทำงานในโปรแกรมไม่เหมือนกันก็ตาม การเขียนโปรแกรมในงานบางงานผลลัพธ์ที่ได้จะไม่ถูกต้องเสมอไป ตัวอย่างเช่นการเขียนโปรแกรมสำหรับหารเลข โดยให้โปรแกรมรับค่าตัวเลขทางแป้นพิมพ์สองค่า แล้วนำค่านั้นมาหารกัน อาจเป็นไปได้ว่าคอมพิวเตอร์สามารถแสดงผลลัพธ์การหารออกมาได้ แต่ถ้าหากมีการป้อนค่าเป็นศูนย์เข้าไปคอมพิวเตอร์ก็ไม่สามารถคำนวณได้ ถ้าหากผู้เขียนโปรแกรมไม่ได้ตรวจสอบตรงจุดนี้ก็ทำให้โปรแกรมที่เขียนออกมาเป็นโปรแกรมที่ไม่สมบูรณ์

โปรแกรมบางโปรแกรมทำงานได้ดี ทำงานได้ถูกต้อง แต่ถ้าหากต้องการพัฒนาต่อหรือปรับปรุงโปรแกรมให้ดีขึ้นทำได้ยาก ก็ไม่ถือว่าโปรแกรมนั้นมีคุณลักษณะของโปรแกรมที่ดี อาจเนื่องมาจากผู้ที่เขียนโปรแกรมไว้ก่อนหน้านั้นไม่ได้มีการจดบันทึก ไม่ได้มีการเขียนคำอธิบายขั้นตอนต่าง ๆ ให้ผู้ที่ต้องการพัฒนาต่อสามารถเข้าใจได้ง่าย ๆ สำหรับโปรแกรมที่มีคุณลักษณะของโปรแกรมที่ดี ควรมีลักษณะดังต่อไปนี้

1. มีความถูกต้องและเชื่อถือได้

โปรแกรมที่ดีต้องให้ผลลัพธ์ที่ถูกต้องแม่นยำ ไม่คลาดเคลื่อน โปรแกรมจึงต้องมีความสมบูรณ์มากที่สุด คือ ผ่านการทดสอบที่ครอบคลุม โปรแกรมต้องนิ่ง ไม่ค่อยเกิดปัญหา เพราะอาจถูกนำไปใช้ในการตัดสินใตสำคัญ ๆ ของผู้บริหาร

2. มีความเป็นมิตรต่อผู้ใช้

ปัจจุบันมีผู้ใช้โปรแกรมในการทำงานมากขึ้น ในจำนวนนี้มีผู้ใช้ที่เป็นผู้เริ่มต้นและไม่มีพื้นฐานทางด้านคอมพิวเตอร์เลย และมีแนวโน้มพิ่มมากขึ้น การสร้างให้โปรแกรมใช้งานง่ายและสะดวกจึงเป็นเรื่องที่มีความสำคัญ

3. ค่าใช้จ่ายต่ำ

ก่อนการพัฒนาต้องวางแผนและประเมินค่าใช้จ่าย เมื่อพัฒนาก็ต้องควบคุมค่าใช้จ่ายให้เป็นไปตามแผน ในมุมมองของผู้ใช้โปรแกรมจะต้องทำงานได้คุ้มกับเงินที่จ่ายไป ในมุมมองของผู้พัฒนา ค่าใช้จ่ายในการพัฒนาต้องต่ำกว่าราคาที่เสนอแก่ลูกค้า

4. ต้องอ่านง่ายและสามารถนำกลับมาใช้ใหม่

โปรแกรมที่มีความสามารถสูง มักมีโครงสร้างที่ใหญ่และซับซ้อน จึงควรมีการออกแบบเป็นโมดูล (Module) ย่อย ๆ ที่มีอิสระต่อกันและเรียบง่าย เพื่อให้ผู้พัฒนาโปรแกรมคนอื่นสามารถเข้าใจ นำไปพัฒนาต่อให้เหมาะสมกับยุคสมัยได้

5. มีความปลอดภัย

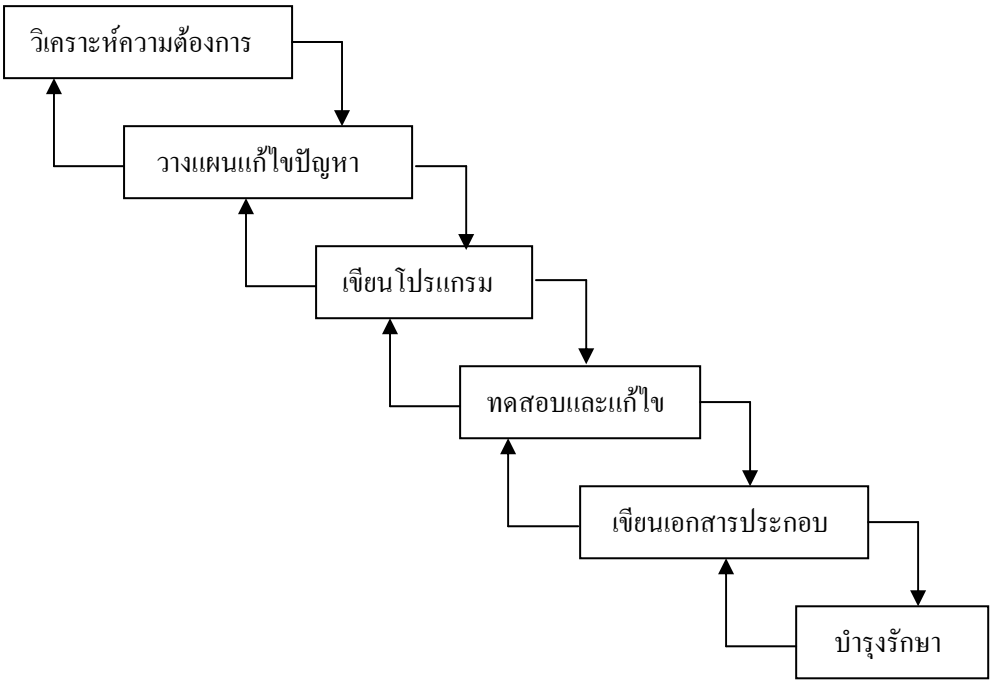
ข้อมูลสำคัญมีแนวโน้มในการเก็บไว้ในคอมพิวเตอร์เพิ่มมากขึ้น รวมถึงเผยแพร่ในอินเทอร์เน็ต ทำให้เกิดความเสี่ยงในเรื่องของความปลอดภัยของข้อมูล เช่น การแก้ไขข้อมูลโดยไม่ได้รับอนุญาต การลักลอบขโมย ไปจนถึงการทำลายข้อมูล โปรแกรมที่ดีจึงต้องมีความปลอดภัยสูง

6. ใช้เวลาในการพัฒนาไม่นาน

ปัจจุบันทั้งเทคโนโลยีและวิธีการทำงานเปลี่ยนแปลงอย่างรวดเร็ว ทำให้เราไม่สามารถใช้เวลานานเกินไปในการพัฒนาโปรแกรม ไม่เช่นนั้นแล้วกว่าโปรแกรมจะเสร็จ ความต้องการอาจเปลี่ยนไปแล้ว และการส่งมอบงานก็ต้องเป็นไปตามที่ประเมินไว้

ในการเขียนโปรแกรมหรือพัฒนาโปรแกรมนั้น โปรแกรมเมอร์ต้องมีการเตรียมงานเกี่ยวกับการเขียนโปรแกรมอย่างเป็นขั้นตอน เรียกขั้นตอนเหล่านี้ว่า วงจรการพัฒนาโปรแกรม (Program Development Life Cycle: PDLC) ประกอบด้วย 6 ขั้นตอนดังนี้

1. ขั้นวิเคราะห์ความต้องการ (Requirement Analysis & Feasibility Study)
2. ขั้นวางแผนแก้ไขปัญหา (Algorithm Design)
3. ขั้นดำเนินการเขียนโปรแกรม (Program Coding)
4. ขั้นทดสอบและแก้ไขโปรแกรม (Program Testing & Debugging)
5. ขั้นการเขียนเอกสารประกอบ (Documentation)
6. ขั้นบำรุงรักษาโปรแกรม (Program Maintenance)



ในการทำกิจกรรมต่าง ๆ ในแต่ละขั้นตอนจะต้องมีการตรวจสอบความถูกต้องให้แน่นอนก่อนที่จะทำกิจกรรมถัดไป สำหรับในบางกรณีถ้าหากพบว่ามีปัญหาบางอย่างเกิดขึ้นก็อาจมีการย้อนไปตรวจสอบกิจกรรมที่เคยทำมาก่อนหน้านี้แล้วก็ได้ สำหรับกิจกรรมในขั้นตอนที่ 5 อาจทำไปพร้อม ๆ กับกิจกรรมในขั้นตอนที่ 1 ถึงที่ 4 เลยก็ได้

วิธีการทางคอมพิวเตอร์เป็นขั้นตอนในการจัดทำโปรแกรมที่ช่วยให้การเขียนโปรแกรมดำเนินไปอย่างมีประสิทธิภาพและได้ผลลัพธ์ตามที่มุ่งหมาย เพราะแต่ละขั้นตอนจะช่วยให้เกิดความเป็นระเบียบในการเขียนโปรแกรม ทำให้การเรียบเรียงแนวคิดมีความชัดเจน ไม่สับสน และเกิดความง่ายต่อการเขียนหรือพัฒนาโปรแกรม แม้ว่าในปัจจุบันจะมีวิธีการสมัยใหม่เกิดขึ้น เช่น เทคโนโลยีการเขียนโปรแกรมเชิงวัตถุ แต่วิธีการทางคอมพิวเตอร์ยังเป็นสิ่งที่จำเป็นและน่ากระทำ โดยเฉพาะอย่างยิ่งผู้ที่เพิ่งเริ่มต้นใหม่กับงานเขียนโปรแกรม เพราะช่วยให้แนวคิดของการพัฒนาโปรแกรมเป็นระเบียบไม่สับสน

3.2 หลักเกณฑ์ในการวิเคราะห์ปัญหา

การวิเคราะห์ปัญหาหรือการวิเคราะห์ความต้องการ เป็นการทำความเข้าใจปัญหาและค้นหาสิ่งที่ต้องการ นั่นคือจะต้องศึกษาโดยละเอียดว่า ต้องการผลลัพธ์อะไร ต้องใช้ข้อมูลอะไรเพื่อให้ได้ผลลัพธ์ตามต้องการ และมีขั้นตอนการประมวลผลอย่างไรบ้าง บางครั้งจะเรียกขั้นตอนนี้ว่า การวิเคราะห์ปัญหาหรือการวิเคราะห์งาน การวิเคราะห์ความต้องการจึงเป็นขั้นตอนแรกก่อนที่จะเริ่มต้นเขียนโปรแกรม และถือว่าเป็นขั้นตอนที่สำคัญที่สุด ก่อนถึงขั้นวางแผนแก้ไขปัญหาและดำเนินการเขียนโปรแกรม

ในการวิเคราะห์ปัญหานั้นมีประเด็นที่ต้องพิจารณาออกมาเป็นข้อ ๆ ดังนี้

- ต้องการอะไร
  จะต้องอธิบายว่างานที่ต้องการให้คอมพิวเตอร์ทำนั้น ต้องการให้ทำอะไร โดยอาจเป็นการเขียนออกมาเป็นข้อ ๆ ก็ได้

- ต้องการเอาต์พุตอย่างไร
  จะอธิบายถึงลักษณะของผลลัพธ์ที่ต้องการว่าต้องการอย่างไร อาจเป็นการแสดงออกทางจอภาพ แสดงออกทางเครื่องพิมพ์ หรือส่งเป็นเสียงออกทางลำโพง โดยต้องอธิบายรายละเอียดด้วย เช่นถ้าต้องการให้แสดงทางจอภาพ หน้าแรกเป็นอย่างไร ตัวเลขที่แสดงเป็นทศนิยมกี่ตำแหน่ง แสดงเป็นตารางอย่างไร ถ้าหากแสดงออกทางเครื่องพิมพ์จะต้องบอกด้วยว่าจะให้พิมพ์เลขหน้าหรือไม่ มีส่วนหัวหรือท้ายเอกสารหรือไม่
- ข้อมูลเข้าเป็นอย่างไร
  ส่วนนี้มักจะออกแบบหลังจากได้ออกแบบส่วนแสดงผลไปแล้ว โดยเมื่อทราบรูปแบบของผลลัพธ์แล้วจะต้องมาดูว่าถ้าหากต้องการข้อมูลแบบที่ต้องการแล้วข้อมูลทางอินพุตควรเป็นอย่างไร ต้องการข้อมูลเท่าใดจึงประมวลผลมาได้เอาต์พุตแบบที่ต้องการเป็นต้น
- ตัวแปรที่ใช้
  เพื่อบอกว่าจะใช้ตัวแปรอะไรแทนข้อมูลนำเข้า หรือแทนค่าที่อยู่ในระหว่างการประมวลผล ตลอดจนตัวแปรที่ใช้แสดงผลลัพธ์
- วิธีการประมวลผลเป็นอย่างไร
  อธิบายถึงลำดับขั้นตอนของการประมวลผล วิธีการแก้ปัญหาเพื่อให้ได้ผลลัพธ์ตามที่ต้องการ โดยต้องพิจารณาว่าข้อมูลต่าง ๆ ที่รับเข้าไปนั้นจะต้องเก็บในตัวแปรกี่ตัว เป็นตัวแปรประเภทใด การประมวลผลมีขั้นตอนกระทำกับตัวแปรนั้น ๆ อย่างไรบ้าง ถ้าหากต้องการประมวลผลกับข้อมูลหลายค่าจะรับข้อมูลเข้าไปทีละค่าแล้วประมวลผล หรือรับข้อมูลเข้าไปทั้งหมดแล้วประมวลผล การเขียนคำอธิบายวิธีการประมวลผลนี้อาจเป็นการเขียนออกมาเป็นข้อ ๆ ในลักษณะของรหัสเทียม หรือเขียนเป็นผังงานก็ได้

วิธีการประมวลผลนั้นมักจะเขียนในลักษณะอัลกอริธึมที่เป็นการจัดลำดับความคิดออกเป็นขั้นตอน เพื่อใช้ในการแก้ปัญหาสำหรับการเขียนโปรแกรม โดยจะแสดงลำดับของขั้นตอนเชิงคำนวณซึ่งแปลงข้อมูลขาเข้าของปัญหาที่กำลังพิจารณาไปเป็นผลลัพธ์ตามที่ต้องการ โดยขั้นตอนต่าง ๆ ที่เขียนขึ้นจะต้องแปลไปเป็นคำสั่งของคอมพิวเตอร์ได้ อัลกอริธึมที่ทำงานได้ดูต้องจะต้องแก้ไขปัญหาและหาคำตอบได้ทุกกรณี

เมื่อได้วิเคราะห์ปัญหาออกมาแล้วจะต้องลองทดสอบดูด้วยตนเอง โดยลองป้อนข้อมูลเข้าไปแล้วดูว่าการแก้ปัญหาเป็นไปตามที่ต้องการหรือไม่

ตัวอย่างที่ 3.1 จงเขียนแนวทางการแก้ปัญหาด้วยคอมพิวเตอร์ สำหรับให้คอมพิวเตอร์คำนวณหาค่าจ้างพนักงานเป็นรายชั่วโมง จากนั้นแสดงค่าจ้างที่คำนวณได้

วิธีทำ

ต้องการอะไร	ต้องการทราบค่าจ้างของพนักงานแต่ละคน
ต้องการเอาต์พุตอย่างไร	ต้องการเอาต์พุตเป็นค่าจ้างสุทธิของพนักงานทางจอภาพ
ข้อมูลเข้า	รหัสพนักงาน , ชื่อพนักงาน , จำนวนชั่วโมงทำงานเก็บในตัวแปรชื่อ Hours , ค่าจ้างรายชั่วโมงเก็บในตัวแปรชื่อ PayRate

วิธีการประมวลผล

กำหนดวิธีการคำนวณ

ค่าจ้างสุทธิ = จำนวนชั่วโมง x อัตราต่อชั่วโมง

ขั้นตอนการประมวลผล

1. เริ่มต้น
2. รับรหัสพนักงาน , ชื่อพนักงาน , จำนวนชั่วโมงทำงาน , ค่าจ้างรายชั่วโมง
3. คำนวณ ค่าจ้างสุทธิ = Hours x PayRate
4. แสดงผลลัพธ์ เป็นรหัสพนักงาน ชื่อ และค่าจ้างสุทธิ
5. จบการทำงาน

ตัวอย่างที่ 3.2 จงเขียนแนวทางการแก้ปัญหาด้วยคอมพิวเตอร์ในการหาปริมาตรทรงกระบอก และแสดงค่าปริมาตรที่คำนวณได้

วิธีทำ

ต้องการอะไร ต้องการคำนวณปริมาตรของรูปทรงกระบอก

ต้องการเอาต์พุตอย่างไร ค่าปริมาตรทรงกระบอกเป็นรูปแบบเลขทศนิยมทางจอภาพ

ข้อมูลเข้า ความสูงของทรงกระบอก (h) และรัศมีของทรงกระบอก(r) เป็นรูปแบบเลขทศนิยม

วิธีการประมวลผล

วิธีการคำนวณ

ปริมาตรทรงกระบอก = π x รัศมี^{2} x ความสูง

ขั้นตอนการประมวลผล

1. เริ่มต้น
2. รับค่าความสูงและค่ารัศมีของทรงกระบอก
3. คำนวณค่าปริมาตร จาก ปริมาตร = π x r^2 x h
4. แสดงค่าปริมาตรทรงกระบอกทางจอภาพ
5. จบการทำงาน

ตัวอย่างที่ 3.3 จงเขียนโปรแกรมเพื่อรายงานผลสอบของนักศึกษาวิชาคอมพิวเตอร์ โดยให้แสดงคะแนนรวมและเกรดออกมา

วิธีทำ

ต้องการอะไร ต้องการพิมพ์คะแนนผลสอบและเกรดของนักศึกษา

ต้องการเอาต์พุตอย่างไร แสดงคะแนนรวมและเกรดของนักศึกษาแต่ละคน

ข้อมูลเข้า รหัสประจำตัวนักศึกษา (ID) ,ชื่อนักศึกname , คะแนนสอบกลางภาค (mid) , คะแนนสอบย่อย (test) , คะแนนสอบปลายภาค (final)

วิธีการประมวลผล

วิธีการคำนวณ

คะแนนรวม = คะแนนกลางภาค + คะแนนสอบย่อย + คะแนนปลายภาค

ถ้า คะแนนรวม >= 80 ได้เกรด “A”

ถ้า คะแนนรวม >= 70 และ <80 ได้เกรด “B”

ถ้า คะแนนรวม >= 60 และ < 70 ได้เกรด “C”

ถ้า คะแนนรวม >= 50 และ <60 ได้เกรด “D”

ถ้าคะแนนรวม < 50 ได้เกรด “F”

ขั้นตอนการประมวลผล

1. เริ่มต้น
2. รับค่าตัวแปร ID , name , mid , test , final
3. คำนวณคะแนนรวมและเกรด

   Total = mid + final + test

   ถ้า Total >= 80 , Grade = “A” ไปข้อ 4

   ถ้า Total >= 70 , Grade = “B” ไปข้อ 4

   ถ้า Total >= 60 , Grade = “C” ไปข้อ 4

   ถ้า Total >= 50 , Grade = “D” ไปข้อ 4

   ถ้า Total < 50 , Grade = “F”
4. แสดง id , name , Total , Grade ของนักศึกษา
5. กลับไปข้อ 2 เพื่อรับจนครบทุกคน ถ้าครบแล้วไปข้อ 6
6. หยุดทำงาน

ตัวอย่างที่ 3.4 จงวิเคราะห์ปัญหาและเขียนอัลกอริทึมของการบวกตัวเลขจำนวนสามค่าที่อ่านเข้ามาและแสดงผลออกทางเครื่องพิมพ์

วิธีทำ

ข้อมูลเข้า ตัวเลขสามค่าให้อยู่ในชื่อตัวแปร number1 , number2 และ number3

ข้อมูลเอาต์พุต ผลรวมให้มีชื่อตัวแปรเป็น total

วิธีการประมวลผล

อ่านค่าตัวเลขจำนวนสามค่า

บวกตัวเลขทั้งสามค่า

แสดงผลรวมของตัวเลข

อัลกอริทึม สามารถเขียนได้ดังนี้

```
Add_three_number
        1. Read number1 , number2 , number3
        2. total = number1 + number2 + number3
        3. Printf total1
END
```

ตัวอย่างที่ 3.5 จงวิเคราะห์ปัญหาและเขียนอัลกอริทึมสำหรับหาค่าเฉลี่ยของอุณหภูมิประจำวัน โดยรับค่าอุณหภูมิสูงสุดและอุณหภูมิต่ำสุดเป็นเลขจำนวนเต็มเข้าไปและให้แสดงค่าอุณหภูมิเฉลี่ยออกทางจอภาพ

วิธีทำ

ข้อมูลเข้า รับค่าอุณหภูมิสูงสุดอยู่ในตัวแปรชื่อ max_temp

อุณหภูมิต่ำสุดอยู่ในตัวแปรชื่อ min_temp

ข้อมูลเอาต์พุต ค่าอุณภูมิเฉลี่ยทางจอภาพโดยใช้ตัวแปรชื่อ avg_temp

วิธีการประมวลผล

1. รับค่าอุณหภูมิสุงสุดและอุณหภูมิต่อสุด
2. หาค่าเฉลี่ยโดยโดย avg_temp = (max_temp + min_temp)/2
3. แสดงค่า avg_temp ทางจอภาพ

อัลกอริทึมสามารถเขียนได้ดังนี้

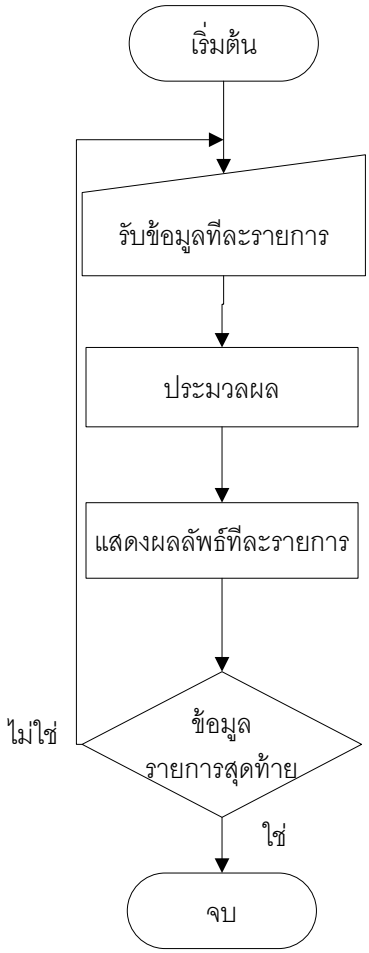
```
Find_average_temperature
        1. READ max_temp , min_temp
        2. avg_temp = (max_temp + min_temp) / 2
        3. Output avg_temp to the screen
END
```

3.3 วิธีการประมวลผล

วิธีการประมวลผล เป็นขั้นตอนการประมวลผลเพื่อให้ได้ผลลัพธ์ตามรูปแบบที่ต้องการ โดยใช้ข้อมูลนำเข้าและตัวแปรต่าง ๆ ตามที่ได้กำหนดไว้แล้ว ขั้นตอนของวิธีการประมวลผลจะขึ้นอยู่กับวิธีการเขียนโปรแกรม โดยทั่วไปการเรียงคำสั่งในโปรแกรมเพื่อประมวลผลข้อมูลมี 2 วิธี คือ การรับข้อมูลทีละค่าเข้าไปประมวลผล และการรับข้อมูลเข้าไปทั้งหมดแล้วจึงประมวลผล

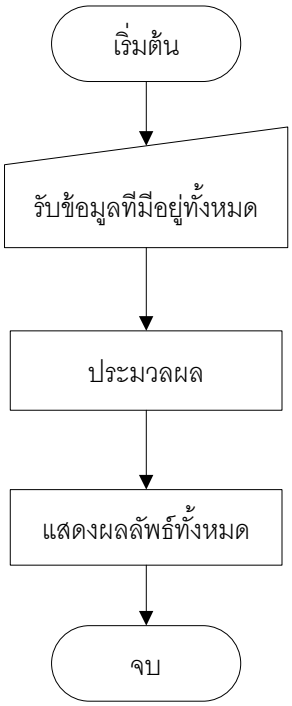
- **รับข้อมูลเข้าสู่คอมพิวเตอร์ทีละรายการ**

วิธีการนี้จะรับข้อมูลรายการแรกเข้ามาประมวลผลแล้วแสดงผลลัพธ์ โดยอาจรับจากแป้นพิมพ์หรือรับจากไฟล์ก็ได้ หลังจากนั้นก็จะรับข้อมูลรายการใหม่ เพื่อนำไปประมวลผลและแสดงผลลัพธ์อีก เป็นเช่นนี้เรื่อย ๆ ไปจนกว่าจะหมดข้อมูลที่ต้องการประมวลผล วิธีนี้เป็นวิธีที่สะดวกและนิยมใช้กันโดยทั่วไป เช่น ในร้านขายสินค้าปลีก ร้านขายหนังสือ ตัวอย่างการประมวลผลเขียนได้ดังแผนภาพต่อไปนี้



- **รับข้อมูลทั้งหมดในครั้งเดียว**

วิธีการนี้จะรับข้อมูลทั้งหมดมาเก็บในหน่วยความจำก่อน จากนั้นประมวลผลแล้วแสดงผลลัพธ์ของทุกรายการในครั้งเดียว วิธีนี้จะต้องใช้เนื้อที่หน่วยความจำมากขึ้น ขณะที่ป้อนข้อมูลเข้าไป



รูปแบบผลลัพธ์และลักษณะของข้อมูล

ในส่วนของวิธีการประมวลผลนี้จะต้องมีการกำหนดรูปแบบการแสดงผลลัพธ์ด้วยว่าจะต้องมีการแสดงอย่างไร รวมถึงวิธีการรับข้อมูลด้วยว่าถ้าหากรับเข้ามาทีละค่า ตัวแปรจะต้องมีลักษณะใด และหากใช้วิธีรับข้อมูลเข้ามาทั้งหมดแล้วประมวลผลจะต้องรับข้อมูลอย่างไร ซึ่งมีรูปแบบการดำเนินการดังนี้

1. พิจารณาว่ารูปแบบของผลลัพธ์ต้องมีหัวตาราง (Table Heading) หรือไม่ ถ้ามีต้องพิมพ์หัวตารางก่อน
2. ถ้าต้องการคำนวณยอดรวมทั้งหมด เช่น ยอดรวมเงินเดือนของพนักงานทุกคน ก็ต้องมีการกำหนดตัวแปรหนึ่งเพื่อเป็นที่เก็บค่าของยอดรวมไว้ตั้งแต่ต้น การกำหนดตัวแปรดังกล่าวมักจะกำหนดให้มีค่าเท่ากับศูนย์ในตอนต้นโปรแกรม เช่น sum = 0
3. พิจารณาข้อมูลที่นำเข้า (Input Data) หรือตัวเลขที่เกี่ยวข้อง ถ้าตัวเลขที่ใช้มีลักษณะเปลี่ยนไปไม่มีระบบกล่าวคือ มีความห่างของตัวเลขไม่แน่นอน ก็ต้องใช้วิธีรับค่าตัวแปรนำเข้า เช่น เงินเดือนของพนักงานเป็นข้อมูลตัวเลขที่มีความห่างของค่าที่ไม่แน่นอน นั่นคือ พนักงานคนแรกมีเงินเดือน 7500 คนที่สอง 9000 คนที่สาม 13000 คนที่สี่ 20000 ไปเรื่อย ๆ ซึ่งค่าความห่างของตัวเลขไม่เท่ากัน

ลำดับค่า	7500	9000	13000	20000	……..

ช่วงห่างตัวเลขมีค่าไม่แน่นอน และไม่เท่ากัน

ลักษณะข้อมูลเช่นนี้ ต้องใช้ตัวแปรเข้ามารับค่า แต่ถ้าข้อมูลที่เกี่ยวข้องเป็นระบบและมีช่วงห่างเท่ากัน เช่น

ลำดับข้อมูล	1	2	3	4	5	6	7
ค่าข้อมูล	5	10	15	20	25	30	35

ข้อมูลลักษณะดังกล่าวอาจใช้วิธีกำหนดตัวแปรให้มีค่าเท่ากับค่าเริ่มต้นของข้อมูลแทนการรับเข้ามาก็พอ และเขียนโปรแกรมให้กำหนดค่าเอง เช่น n =1 และ number = 5 เป็นต้น

ทั้งการกำหนดและการรับข้อมูลมักเป็นขั้นตอนต่อจากข้อ 1 หรือข้อ 2 เมื่อได้กำหนดและรับข้อมูลแล้ว ขั้นต่อไปคือการประมวลผล และพิมพ์ผลลัพธ์

4. ในการเขียนโปรแกรมรับข้อมูลทีละรายการ จะต้องมีการกำหนดจุดจบของการประมวลผล หรือมีการทดสอบว่าข้อมูลหมดหรือยัง โดยอาจใช้วิธีทดสอบข้อมูลสุดท้าย หรือใช้วิธีการกำหนดตัวแปรสำหรับนับรอบ (Loop) เพื่อให้นับจำนวนรอบการประมวลผลจนกว่าจะครบตามจำนวนครั้งที่ต้องการ ดังรายละเอียดต่อไปนี้

การทดสอบข้อมูลสุดท้าย

การทดสอบข้อมูลสุดท้ายมี 2 วิธี ดังนี้

1. เลือกตัวแปรที่เป็นข้อมูลนำเข้า ตัวใดตัวหนึ่งที่มีค่าเป็นไปไม่ได้เป็นตัวทดสอบ ลำดับของการทดสอบจะอยู่ถัดจากการรับข้อมูล ดังนั้น การทดสอบว่าข้อมูลเข้ามาหมดหรือยังจึงกระทำก่อนมีการประมวลผล หากทดสอบแล้วพบว่าตัวแปรมีค่าเท่ากับค่าที่เป็นไปไม่ได้ก็จะเลิกจากการประมวลผลข้อมูลทีละรายการ
2. ใช้วิธีกำหนดตัวแปรอีก 1 ตัว เพื่อรับค่าว่าข้อมูลหมดหรือยัง วิธีนี้มักจะวางคำสั่งทดสอบอยู่หลังจากเสร็จสิ้นการแสดงผลลัพธ์ในแต่ละรอบแล้ว นั่นคือข้อมูลรายการสุดท้ายจะผ่านการประมวลผลมาแล้ว หากโปรแกรมพบว่าเป็นรายการสุดท้ายก็จะเลิกการประมวลผล

พิจารณาการเขียนโปรแกรมด้วยภาษาเบสิคง่าย ๆ ต่อไปนี้

คำสั่ง	ความหมาย
10 INPUT X	รับค่าตัวเลขทางแป้นพิมพ์ แล้วเก็บไว้ในตัวแปร X ครั้งละ 1 ค่า
20 IF X < 0 THEN GOTO 60	ถ้าตัวเลขในตัวแปร X มีค่าติดลบ ให้ทำคำสั่งในบรรทัดที่ 60 แต่ถ้าไม่ใช่ให้ทำคำสั่งในบรรทัดที่ 30 ต่อไป
30 Y = X + 200	นำค่าที่อยู่ในตัวแปร X ขณะนั้นบวกกับ 200 แล้วเก็บในตัวแปร Y
40 PRINT X , Y	แสดงค่าที่อยู่ในตัวแปร X และตัวแปร Y
50 GOTO 10	ย้อนกลับไปทำคำสั่งในบรรทัดที่ 10 นั่นคือ รับค่าตัวเลขทางแป้นพิมพ์ตัวใหม่ และเก็บไว้ในตัวแปร X (ค่าเดิมถูกแทนด้วยค่าใหม่)
60 PRINT “GOOD BYE”	ถ้าค่าที่รับเข้ามาทางแป้นพิมพ์ เป็นค่าลบ ก็ทำคำสั่งในบรรทัดที่ 60 นั่นคือแสงดคำว่า GOOD BYE บนหน้าจอ
70 END	ให้หยุดการทำงาน

จากตัวอย่างโปรแกรมภาษาเบสิก คำสั่งในบรรทัดที่ 20 จะใช้สำหรับทดสอบว่าข้อมูลเข้ามาหมดหรือยัง โดยทดสอบว่าถ้าหากข้อมูลที่เข้ามาเป็นค่าลบหมายความว่าข้อมูลเข้ามาครบทั้งหมดแล้ว โปรแกรมจะกระโดดไปยังบรรทัดที่ 60 เพื่อพิมพ์คำว่า “GOOD BYE” ซึ่งเป็นการจบโปรแกรม สำหรับตัวอย่างต่อไปจะเป็นการประกาศตัวแปรอีกหนึ่งตัวชื่อว่า LC เป็นตัวแปรทดสอบ

คำสั่ง	ความหมาย
10 INPUT X , LC	รับค่าตัวเลขเข้ามาเก็บในตัวแปร X และ LC ถ้าหากค่าใน LC เป็น 1 แสดงว่าข้อมูลยังไม่หมด
20 Y = X + 200	นำค่าในตัวแปร X บวกกับ 200 แล้วเก็บไว้ในตัวแปร Y
30 PRINT X , Y	แสดงผลลัพธ์ค่า X และ Y ออกทางจอภาพ
40 IF LC = 1 THEN GOTO 10 ELSE GOTO 50	ถ้าหากค่าที่อยู่ในตัวแปร LC เป็น 1 ให้ไปทำคำสั่งที่บรรทัด 10 แต่ถ้าไม่ใช่ ให้ไปทำคำสั่งที่บรรทัด 50 เป็นต้นไป
50 PRINT “GOOD BYE”	แสดงคำว่า “GOOD BYE” ออกทางจอภาพ
60 END	จบโปรแกรม

จากตัวอย่างเมื่อโปรแกรมทำงานจะให้ใส่ค่าตัวเลขสองค่า โดยค่าแรกเป็นค่าข้อมูลเก็บไว้ในตัวแปร X ส่วนค่าที่สองเป็นข้อมูลทดสอบ เก็บในตัวแปร LC โดยถ้าหากค่าในตัวแปร LC เป็นค่า 1 หมายความว่ายังไม่ใช่ข้อมูลตัวสุดท้าย เมื่อโปรแกรมทำงานมาถึงบรรทัดที่ 40 จะทดสอบว่าค่าในตัวแปร LC เป็น 1 ใช่หรือไม่ ถ้าใช้จะกระโดดไปยังบรรทัดที่ 10 เพื่อรับข้อมูลครั้งต่อไป

3.4 การทดสอบขั้นตอนวิธีการแก้ปัญหา

หลังจากที่ได้วิเคราะห์ปัญหาแล้ว ขั้นตอนสุดท้ายคือขั้นตอนสำหรับทดสอบว่าวิธีการแก้ปัญหาที่ได้สร้างขึ้นนั้นสามารถใช้งานได้หรือไม่ ซึ่งทดสอบโดยการสมมติข้อมูลที่ใช้เป็นข้อมูลอินพุตในการแก้ปัญหาโดยยกตัวอย่างข้อมูลประมาณสองหรือสามกรณี จากนั้นให้ลองแทนค่าลงในขั้นตอนต่าง ๆ ของการประมวลผลแล้วพิจารณาว่าได้คำตอบตามต้องการหรือไม่ ถ้าหากคำตอบผิดพลาดจะต้องกลับไปแก้ไขว่าสำดับขั้นตอนใดที่ทำงานไม่ถูกต้องและให้กลับไปแก้ไขขั้นตอนการทำงานใหม่ การทดสอบขั้นตอนการแก้ปัญหานี้ควรทดสอบกับข้อมูลหลาย ๆ ชุด ถ้าหากทดสอบกับข้อมูลชุดเดียวแล้วทำงานถูกต้องก็ไม่ได้หมายความว่าขั้นตอนการทำงานที่ออกแบบขึ้นทำงานได้อย่างถูกต้องแล้ว

การตรวจสอบในแต่ละขั้นตอนควรสร้างตารางขึ้นมาแสดงการทำงานในแต่ละขั้นตอนของการประมวลผล หากเกิดข้อผิดพลาดขึ้นมาจะได้ดูได้ง่ายว่าผิดพลาดในขั้นตอนใด

ตัวอย่างที่ 3.6 จงตรวจสอบขั้นตอนวิธีในการแก้ปัญหาของโจทย์ในตัวอย่างที่ 3.4

วิธีทำ จากตัวอย่างที่ 3.4 มีอัลกอริทึมการประมวลผลดังนี้

```
Add_three_number
        1. Read number1 , number2 , number3
        2. total = number1 + number2 + number3
        3. Printf total
END
```

การตรวจสอบขั้นตอนวิธีทำได้ดังนี้

1. สร้างตัวเลขทดสอบทางอินพุตขึ้นมาสองชุด โดยให้ชุดแรกมีค่าเป็น 10,20 และ 30 ตัวเลขชุดที่สองมีค่าเป็น 40,41 และ 42

ตัวแปร	ข้อมูลชุดแรก	ข้อมูลชุดที่สอง
number1	10	40
number2	20	41
number3	30	42

2. ลองคิดคำตอบของการประมวลผลด้วยตนเอง

ตัวแปร	ข้อมูลชุดแรก	ข้อมูลชุดที่สอง
Total	60	123

3. สร้างตารางขึ้นมาแสดงการทำงานตามอัลกอริทึมแต่ละขั้นโดยให้แสดงค่าของตัวแปรจากการประมวลผลในแต่ละขั้นตอนและสิ่งที่กระทำกับตัวแปรนั้น ๆ

หมายเลขลำดับ	number1	number2	number3	Total
ชุดที่ 1				
1	10	20	30	
2				60
3				Print
ชุดที่ 2				
1	40	41	42	
2				123
3				Print

4. ตรวจสอบผลลัพธ์ของขั้นตอนที่ 2 และขั้นตอนที่ 3 (60 และ 123) ว่าได้ผลลัพธ์ตรงกันหรือไม่โดยดูที่ตัวแปร Total

ตัวอย่างที่ 3.7 ถ้าหากต้องการพัฒนาโปรแกรมให้รับข้อมูลตัวเลขจำนวนเต็มทางแป้นพิมพ์จำนวน 9 ตัว แล้วให้ระบบนับว่าตัวเลขเหล่านั้นมีค่าที่เป็นเลขบวกกี่ตัว เป็นเลขลบกี่ตัว และเป็นศูนย์กี่ตัว โดยให้แสดงผลลัพธ์ทางจอภาพ

วิธีทำ จากปัญหาดังกล่าวสามารถวิเคราะห์งานได้ดังนี้

สิ่งที่ต้องการ ต้องการทราบจำนวนตัวเลขที่เป็นบวก เป็นลบ และเป็นศูนย์

ผลลัพธ์ที่ต้องการ แสดงจำนวนตัวเลขที่เป็นบวก,ลบ และศูนย์ทางจอภาพ

ข้อมูลนำเข้า รับข้อมูลเข้าทางแป้นพิมพ์ทีละตัว

ตัวแปรที่ใช้ ให้ X แทนตัวเลขที่รับเข้าทางแป้นพิมพ์ทีละตัว

num_p แทนตัวนับจำนวนที่เป็นบวก

num_n แทนตัวนับจำนวนที่เป็นลบ

num_z แทนตัวนับจำนวนที่เป็นศูนย์

วิธีการประมวลผล สำหรับปัญหานี้จะให้รับข้อมูลเข้าทางแป้นพิมพ์ทีละค่า โดยมีวิธีการดังนี้

กำหนดวิธีการคิด

ถ้าตัวเลขเป็นจำนวนบวก ให้บวก 1 เข้ากับตัวนับจำนวนบวก

ถ้าตัวเลขเป็นจำนวนลบ ให้บวก 1 เข้ากับตัวนับจำนวนลบ

ถ้าตัวเลขเป็นศูนย์ ให้บวก 1 เข้ากับตัวนับจำนวนศูนย์

กรรมวิธีการประมวลผล

1. กำหนดให้ตัวนับ num_p, num_n และ num_z มีค่าเป็นศูนย์
2. อ่านค่าตัวเลขทางแป้นพิมพ์มาเก็บในตัวแปร X
3. ถ้าอ่านไม่ได้ ไปข้อ 7
4. ถ้า X เป็นจำนวนบวก ให้บวก 1 กับ num_p แล้วไปข้อ 2
5. ถ้า X เป็นจำนวนลบ ให้บวก 1 กับ num_n แล้วไปข้อ 2
6. ถ้า X เป็นค่าศูนย์ ให้บวก 1 กับ num_z แล้วไปข้อ 2
7. พิมพ์ค่าตัวนับทางจอภาพทั้งสามตัว
8. หยุด

สำหรับกรรมวิธีการประมวลผล ถ้าหากในข้อ 6 ตัดคำว่า “ถ้า X เป็นค่าศูนย์” ทิ้งไป คำตอบก็จะได้เท่าเดิม เนื่องจากถ้าหากผ่านข้อ 4 และ 5 มาได้ก็แสดงว่าตัวเลขนั้นมีค่าเป็นศูนย์อยู่แล้ว สำหรับการตรวจสอบวิธีการประมวลผลสามารถทำได้ดังนี้

1. ลองยกตัวอย่างข้อมูลขึ้นมา 1 ชุด ถ้าหากตัวเลขทั้ง 9 ตัวมีค่าเป็น 13,7,-4,0,-5,2,6,-2 และ 35
2. คิดคำตอบการประมวลผลด้วยตนเอง จะต้องได้จำนวนบวก 5 ตัว จำนวนลบ 3 ตัว และศูนย์ 1 ตัว

ตัวแปร	ข้อมูลชุดแรก
num_p	5
num_n	3
num_z	1

3. ถ้าหากสร้างตารางขึ้นมาเพื่อแสดงตามขั้นตอนวิธีการประมวลผลจะได้ดังนี้

หมายเลขลำดับ	ตัวเลข (X)	num_p	num_n	num_z
1	-	0	0	0
2	13	0	0	0
3	13	0	0	0
4	13	1	0	0
2	7	1	0	0
3	7	1	0	0
4	7	2	0	0
2	-4	2	0	0
3	-4	2	0	0
4	-4	2	0	0
5	-4	2	1	0
2	0	2	1	0
3	0	2	1	0
4	0	2	1	0
5	0	2	1	0
6	0	2	1	1
2	-5	2	1	1
3	-5	2	1	1
4	-5	2	1	1
5	-5	2	2	1
2	2	2	2	1
3	2	2	2	1
4	2	3	2	1
2	6	3	2	1
3	6	3	2	1

หมายเลขลำดับ	ตัวเลข (X)	num_p	num_n	num_z
4	6	4	2	1
2	-2	4	2	1
3	-2	4	2	1
4	-2	4	2	1
5	-2	4	3	1
2	35	4	3	1
3	35	4	3	1
4	35	5	3	1
2	-	5	3	1
7	-	5	3	1

4. จะพบว่าคำตอบในขั้นตอนที่ 7 จะตรงกับคำตอบที่คิดเอาไว้ แสดงว่าแนวทางการประมวลผลนี้ถูกต้อง ซึ่งสามารถนำไปพัฒนาเป็นโปรแกรมต่อไปได้

เมื่อแน่ใจแล้วว่าวิธีการแก้ปัญหาที่ออกแบบขึ้นเป็นวิธีที่ถูกต้องจึงดำเนินการเขียนโปรแกรม และโปรแกรมที่เขียนขึ้นก็ต้องตรวจสอบจุดผิดพลาดเช่นกัน จุดผิดพลาดของโปรแกรมเรียกว่า “บัก” (Bug) สำหรับการดำเนินการแก้ไขข้อผิดพลาดนั้นเรียกว่า “ดีบัก” (DeBug) สำหรับโปรแกรมที่ทำงานไม่ได้ตามวัตถุประสงค์ เรียกว่า โปรแกรมมี Error การเกิด Error ของโปรแกรมมักมีมาจาก 2 สาเหตุหลักคือ

1. ข้อผิดพลาดทางไวยากรณ์ของภาษา (Syntax Error) เป็นข้อผิดพลาดที่เกิดจากการเขียนโค้ดคำสั่ง (Source Code) ที่ไม่ตรงกับไวยากรณ์ (Syntax) ของโปรแกรมนั้น ๆ ข้อผิดพลาดนี้เป็นข้อผิดพลาดที่ง่ายที่สุดในการตรวจหาและแก้ไข เนื่องจากจะพบในระหว่างที่มีการแปลภาษาโปรแกรม ถ้ามีข้อผิดพลาดประเภทนี้อยู่ ตัวแปลภาษาคอมพิวเตอร์จะแจ้งให้ทราบทันที

ตัวอย่างเช่น ถ้าเขียนโปรแกรมด้วยภาษาปาสคาล สั่งให้คอมพิวเตอร์พิมพ์ข้อความออกมาทางหน้าจอด้วยคำสั่ง WXRITLN(‘The Area is : ‘,my_area); เมื่อมีการแปลภาษาโปรแกรม คอมพิวเตอร์จะตรวจสอบไวยากรณ์ของภาษา พบว่าคำสั่ง WXRITLN ไม่มีอยู่ในภาษาปาสคาล ดังนั้นในบรรทัดนี้คอมพิวเตอร์จะแสดงข้อผิดพลาดประเภท Syntax Error ออกมาให้เห็น

2. ข้อผิดพลาดที่เกิดจากการตีความหมายของปัญหาผิดไป (Logical Error) เป็นข้อผิดพลาดที่เกิดจากการออกแบบอัลกอริธึมให้ทำงานผิดจากวัตถุประสงค์หรือความต้องการ การแก้ไขข้อผิดพลาดประเภทนี้จะต้องทำโดยการตรวจไล่โปรแกรมทีละคำสั่งเพื่อหาข้อผิดพลาดนั้นให้พบ เช่น ผู้เขียนโปรแกรมต้องการนำค่า A ไปบวกกับค่า B แต่เขียนเครื่องหมายเป็นลบ ก็จะทำให้ค่าที่คำนวณได้ไม่ถูกต้อง ซึ่งสามารถตรวจสอบข้อผิดพลาดนี้โดยใช้ ข้อมูลทดสอบ (Test Data) หรือข้อมูลที่ทราบคำตอบในขั้นปฏิบัติการ (Execution Run) ถ้าโปรแกรมถูกต้อง ผลลัพธ์ที่ได้จะต้องตรงกับคำตอบที่เราทราบอยู่ก่อนแล้ว

การตรวจสอบโปรแกรมเพื่อหาข้อผิดพลาดดังกล่าว อาจจะทำเป็นขั้นตอน ดังนี้

1. การตรวจสอบก่อนนำโปรแกรมเข้าเครื่องคอมพิวเตอร์ เรียกวิธีการนี้ว่า “Desk Checking” วิธีการนี้จะตรวจสอบว่าโปรแกรมสามารถให้ผลลัพธ์ตามต้องการได้หรือไม่ เป็นการตรวจสอบข้อผิดพลาดทางตรรก โดยกำหนดข้อมูลชุดหนึ่งขึ้นมาแล้วแทนค่าตามขั้นตอนต่าง ๆ ที่เขียนเป็นโปรแกรมไว้ตั้งแต่ต้นจนจบ โดยสมมติว่าเป็นการปฏิบัติงานของเครื่องคอมพิวเตอร์ วิธีการเช่นนี้จะช่วยลดข้อผิดพลาดทางตรรกะได้มาก ก่อนจะส่งโปรแกรมเข้าเครื่องต่อไป
2. การตรวจสอบโดยเครื่องคอมพิวเตอร์ เมื่อผ่านการตรวจสอบในขั้นแรกแล้ว ก็ส่งโปรแกรมเข้าเครื่องคอมพิวเตอร์ พร้อมข้อมูลสมมุติที่ทราบคำตอบ ขั้นแรกเครื่องคอมพิวเตอร์จะทำการตรวจ Syntax Error หรือข้อผิดพลาดทางไวยากรณ์ ถ้ามีที่ผิดเครื่องคอมพิวเตอร์จะพิมพ์ข่าวสารข้อผิดพลาดนี้ออกมา ถ้าได้คำตอบตรงกับที่ทราบก็ค่อนข้างมั่นใจได้ว่าโปรแกรมน่าจะถูกต้อง เมื่อทดสอบโปรแกรมแล้ว สามารถนำโปรแกรมพร้อมกับข้อมูลจริงเข้าเครื่องคอมพิวเตอร์ เพื่อทำการประมวลผลต่อไป

3.5 รูปแบบการเขียนโปรแกรมคอมพิวเตอร์

ในการเขียนโปรแกรมคอมพิวเตอร์นั้นมีลักษณะการเขียนโปรแกรมอยู่หลายลักษณะ ซึ่งขึ้นกับภาษาคอมพิวเตอร์ที่เลือกใช้ด้วย รูปแบบการเขียนโปรแกรมที่นิยมใช้กันมีดังนี้

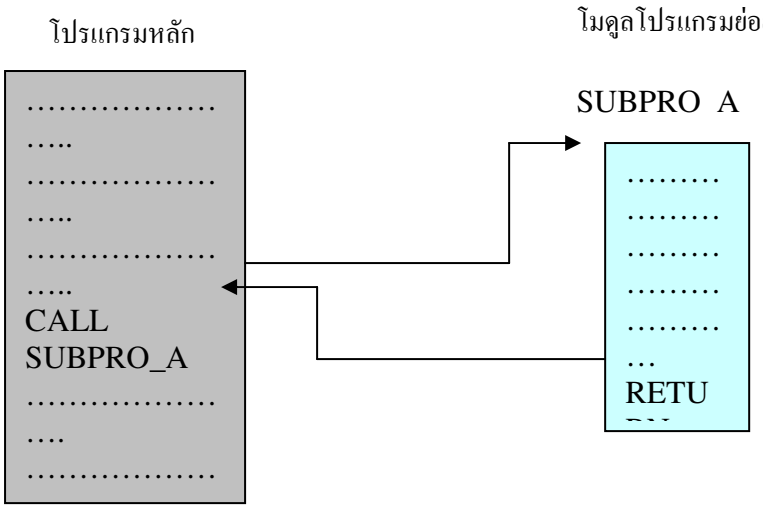
การเขียนโปรแกรมแบบบนลงล่าง (Top-Down Programming)

เป็นการวางแผนการเขียนโปรแกรมแบบโมดูลาร์เป็นลำดับ โดยมองปัญหาตั้งแต่ต้นจนจบการทำงาน ถ้าหากโมดูลใดมีความซับซ้อนก็จะให้แบ่งโมดูลนั้นออกเป็นโมดูลย่อย และให้โมดูลหลักเรียกโมดูลย่อยต่าง ๆ มาใช้งานเป็นลำดับ ในการออกแบบโปรแกรมจะเขียนโครงสร้างต่าง ๆ เป็นลักษณะสี่เหลี่ยมผืนผ้า โดยเขียนการทำงานแต่ละส่วนเรียงลำดับกันไป จากนั้นจึงนำโครงสร้างที่ออกแบบขึ้นมาพัฒนาเป็นโปรแกรมคอมพิวเตอร์

การเขียนโปรแกรมแบบโมดูลาร์ (Modular Programming)

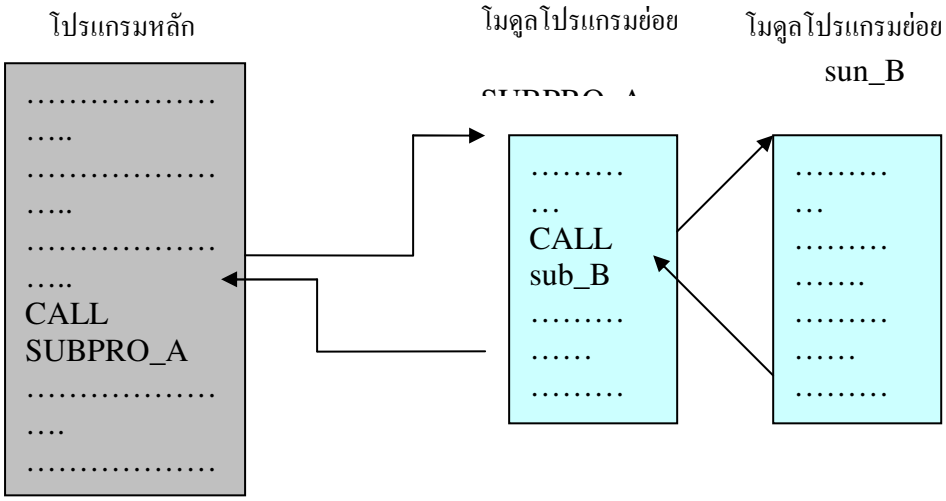
เป็นการเขียนโปรแกรมที่แยกการทำงานต่าง ๆ ออกเป็นโปรแกรมย่อยขนาดเล็กหลาย โปรแกรม แต่ละโปรแกรมเรียกว่า โมดูล (Module) ซึ่งสามารถทำงานได้อิสระจากโมดูลอื่น ๆ ทำให้ถ้าหากโปรแกรมมีความผิดพลาดก็สามารถแก้ไขโมดูลที่ผิดพลาดได้ง่ายขึ้น สำหรับในการเขียนโปรแกรมจะต้องมีโมดูลหลักที่ทำหน้าที่ควบคุมการทำงานทั้งหมด ว่าจะเรียกโมดูลใดมาใช้งานก่อน โมดูลแต่ละโมดูลสามารถเรียกโมดูลย่อย ๆ ได้อีก และแต่ละโมดูลอาจมีการส่งการควบคุมไปยังโมดูลอื่น ๆ ได้อีกด้วย

การเขียนโปรแกรมในลักษณะโมดูลโปรแกรมย่อยจะมีสองประเภทคือ โปรแกรมย่อยภายใน (Internal Subroutine) ที่เป็นโปรแกรมย่อยที่ผู้เขียนโปรแกรมเขียนขึ้นเพื่อเป็นส่วนหนึ่งของโปรแกรมหลัก แต่สามารถเรียกใช้ได้อีกประเภทหนึ่งคือโปรแกรมย่อยภายนอก (External Subroutine) ซึ่งเป็นโปรแกรมย่อยที่เก็บไว้ในไลบรารี (library) ที่สามารถเรียกใช้ได้ และสามารถนำไปใช้ในงานอื่น ๆ ได้อีก



รูปโปรแกรมหลักมีการเรียกโมดูลโปรแกรมย่อย

จากรูปจะเห็นว่าโปรแกรมหลัก หรือโมดูลหลักจะมีการเรียกโมดูลโปรแกรมย่อย เมื่อทำโมดูลโปรแกรมย่อยจบแล้ว โปรแกรมจะกลับมาทำโปรแกรมหลักต่อไป การเรียกโมดูลโปรแกรมย่อยนี้ สามารถเรียกใช้ได้หลาย ๆ ครั้งตามต้องการ สำหรับการเขียนโปรแกรมแต่ละภาษานั้นจะมีคำสั่งสำหรับเรียกโมดูลโปรแกรมย่อยแตกต่างกันไป และโมดูลโปรแกรมย่อยก็จะมีคำสั่งเพื่อบอกว่าโปรแกรมย่อยนั้นจบลงแล้ว



รูปโปรแกรมที่โมดูลโปรแกรมย่อยสามารถเรียกโมดูลโปรแกรมย่อยอื่นอีก

การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming)

เป็นการเขียนโปรแกรมแนวใหม่ที่มองการทำงานต่าง ๆ เป็นวัตถุ หรือออปเจ็ต โดยที่จะมีการบอกว่าออปเจ็ตนั้นคืออะไร ใช้ทำอะไร เมื่อโปรแกรมต้องการใช้ ก็สามารถนำออปเจ็ตนั้นมาใช้ได้ ปรับขนาดได้ เปลี่ยนสีได้ ในการเขียนโปรแกรมจะทำการสร้างออปเจ็ตต่าง ๆ เพื่อสำหรับเรียกใช้งานในภายหลัง

แบบฝึกหัดท้ายบท

ตอนที่ 1 จงเลือกคำตอบที่ถูกต้องที่สุดเพียงหนึ่งข้อ

1. ขั้นตอนการวิเคราะห์ปัญหาขั้นตอนใดที่ผู้พัฒนาโปรแกรมต้องศึกษาถึงวิธีการประมวลผลมากที่สุด

   ก. สิ่งที่ต้องการ ข. รูปแบบการแสดงผลทางเอาต์พุต

   ค. การประมวลผล ง. ลักษณะของข้อมูลเข้า

2. ขั้นตอนในเป็นขั้นตอนสุดท้ายของการวิเคราะห์ปัญหา

   ก. ทำความเข้าใจกับปัญหา

   ข. ทำความเข้าใจกับลักษณะข้อมูลเข้าออก

   ค. ทดสอบขั้นตอนวิธีการแก้ปัญหา

   ง. ทดสอบรูปแบบข้อมูลเข้าและข้อมูลออก

3. ข้อใดคือความหมายของอัลกอริทึม

   ก. การทำความเข้าใจกับปัญหา

   ข. การอธิบายขั้นตอนการทำงานเป็นข้อ ๆ

   ค. การคิดวิธีการแก้ปัญหา

   ง. การทดสอบขั้นตอนการทำงาน

4. ถ้าหากต้องการให้คอมพิวเตอร์คำนวณหาพื้นที่ของสามเหลี่ยม ท่านคิดว่าข้อมูลใดคือข้อมูลอินพุต

   ก. พื้นที่ , ความสูง

   ข. พื้นที่ , ความยาวฐาน

   ค. ความยาวฐาน , ความสูง

   ง. พื้นที่ , ความยาวฐาน และความสูง

5. การเขียนโปรแกรมคอมพิวเตอร์จะเขียนจากขั้นตอนใด

   ก. พิจารณาข้อมูลอินพุต

   ข. พิจารณาข้อมูลเอาต์พุต

   ค. การอธิบายวิธีการประมวลผล

   ง. สิ่งที่ต้องการทางเอาต์พุต

6. การทดสอบความถูกต้องของขบวนการแก้ปัญหาควรจะทำกับข้อมูลตัวอย่างกี่ชุด
    ก. อย่างน้อย 1 ชุด
    ข. อย่างน้อย 2 ชุด
    ค. ขึ้นกับลักษณะของปัญหา
    ง. ทุกกรณีที่สามารถเป็นไปได้ในปัญหานั้น ๆ
7. ข้อใดคือการวิเคราะห์สิ่งที่ต้องการทางเอาต์พุต
    ก. การวิเคราะห์ว่าโปรแกรมนั้นต้องทำงานบนเครื่องรุ่นใด
    ข. การวิเคราะห์ว่าต้องการผลลัพธ์อะไรจากการประมวลผล
    ค. การวิเคราะห์รูปแบบการพิมพ์ผลลัพธ์
    ง. ถูกทุกข้อ
8. ท่านคิดว่าการประมวลผลที่รับข้อมูลเข้าไปประมวลผลทีละค่า เหมาะกับงานประเภทใด
    ก. การจำหน่ายตั๋วชมภาพยนตร์ที่มีที่นั่งจำกัด
    ข. การซื้อขายสินค้า
    ค. การจำหน่ายตั๋วคอนเสิร์ตที่มีที่นั่งไม่จำกัด
    ง. ถูกทุกข้อ
9. เหตุใดจึงต้องตั้งชื่อตัวแปรในการประมวลผล
    ก. เนื่องจากการประมวลผลต้องมีที่เก็บข้อมูล
    ข. เป็นสัญลักษณ์ให้เข้าใจง่ายในการประมวลผล
    ค. สามารถนำไปใช้ในขั้นตอนการเขียนโปรแกรมได้
    ง. ถูกทุกข้อ
10. การรับข้อมูลเข้าไปทุกรายการแล้วประมวลผลทีเดียวเหมาะกับงานประเภทใด
    ก. การหาค่าเงินเดือนเฉลี่ยของพนักงาน
    ข. ระบบการยืมคืนหนังสือของห้องสมุด
    ค. การบันทึกประวัติข้อมูลบุคคล
    ง. ถูกทุกข้อ

ตอนที่ 2 จงทำเครื่องหมาย √ หน้าข้อที่ถูก และเครื่องหมาย × หน้าข้อที่ผิด

.....1. การเขียนคำอธิบายวิธีการประมวลผลทำให้สามารถช่วยการเขียนโปรแกรมได้ง่ายขึ้น

.....2. ถ้าข้อมูลเข้าเป็นตัวเลข จะต้องพิจารณาชนิดของตัวเลขด้วยว่าเป็นทศนิยมหรือจำนวนเต็ม

……3. การวิเคราะห์ปัญหาจะต้องดูสิ่งที่ต้องการทางเอาต์พุตและสิ่งที่รับเข้าทางอินพุตเป็นอันดับแรก

.....4. วิธีการประมวลผลไม่ควรมีหลายข้อ

.....5. ลักษณะการพิมพ์ออกทางเครื่องพิมพ์ ถือว่าเป็นสิ่งที่ต้องการทางเอาต์พุต

ตอนที่ 3 จงตอบคำถามต่อไปนี้

1. จงบอกความหมายของการวิเคราะห์ปัญหา

.....

.....

.....

2. จงบอกความหมายของการวิเคราะห์รูปแบบของข้อมูลอินพุตและเอาต์พุต

.....

.....

.....

.....

3. ถ้าหากต้องการสร้างโปรแกรมคำนวณราคาขายสินค้าโดยจะต้องได้กำไร 20% และรวมภาษี 7% จงวิเคราะห์ปัญหานี้ และเขียนวิธีการประมวลผลออกมาเป็นข้อ ๆ

.....

.....

.....

.....

.....

.....

.....

4. จงวิเคราะห์ปัญหาของการคำนวณการเปลี่ยนค่าของอุณหภูมิจากหน่วยของศาเซลเซียส(Celsius) เป็นหน่วยของศาฟาเรนไฮต์(Fahrenheit) โดยสามารถคำนวณได้จากสูตร

$$F = (C \times 9)/5 + 32$$

โดยที่ F คืออุณหภูมิในหน่วยของศาฟาเรนไฮต์

C คืออุณหภูมิในหน่วยของศาเซลเซียส

พร้อมทั้งทดสอบการทำงานของขั้นตอนวิธี

บทที่ 5

การเขียนอัลกอริธึมโปรแกรม

เมื่อได้ทำการวิเคราะห์ปัญหาและวางแผนการแก้ปัญหาแล้ว ก่อนที่จะลงมือเขียนโปรแกรมเราจะต้องออกแบบขั้นตอนการทำงานหรืออัลกอริทึม (Algorithm) ก่อน โดยจะ ใช้เป็นเครื่องมือในการแสดงขั้นตอนการทำงานของระบบงานใด ๆ เพื่อให้การเขียนโปรแกรมเป็นไปได้รว ดเร็วและง่ายขึ้น โดยเราอาจเขียนอัลกอริทึมในลักษณะผังงาน (Flowchart) ตามที่ได้ศึกษามาในบทที่ 4 หรือเขียนเป็นรหัสลำลองที่เรียกว่าซูโดโค้ด (Psudocodes) ก็ได้

5.1 ความหมายของอัลกอริธึม

อัลกอริธึม (Algorithm) หมายถึงลำดับของขั้นตอนเชิงคำนวณที่แปลข้อมูลด้านอินพุตของปัญหาไปเป็นผลลัพธ์ที่ต้องการ ขั้นตอนต่าง ๆ ในอัลกอริธึมสามารถเปลี่ยนไปเป็นคำสั่งที่ให้คอมพิวเตอร์ทำงานได้ ดังนั้นถ้าหากทำตามอัลกอริธึมแล้ว ปัญหาจะต้องถูกแก้ได้สำเร็จและได้คำตอบที่ถูกต้องสำหรับทุกกรณีตามที่กำหนดในอัลกอริธึม ดังนั้นเราจะไม่ยอมรับอัลกอริธึมที่การทำงานติดอยู่ในวงวนไม่สิ้นสุด หรืออัลกอริธึมประเภทที่ทำงานแล้วได้คำตอบถูกบ้าง ผิดบ้าง ดังนั้นจุดประสงค์ของการออกแบบอัลกอริธึมสำหรับแก้ปัญหาหนึ่ง ๆ ก็คือการทำงานที่ถูกต้อง และทำงานได้อย่างมีประสิทธิภาพ นอกจากการบรรยายอัลกอริธึมด้วยผังงานแล้ว ยังสามารถเขียนบรรยายอัลกอริธึมด้วยคำบรรยายสั้น ๆ ที่ได้ใจความได้ โดยการเขียนรหัสลำลอง หรือซูโดโค้ด (pseudo code) ซึ่งคล้ายกับโปรแกรมภาษาคอมพิวเตอร์ แต่จะบรรยายด้วยคำที่ง่าย ๆ

ซูโดโค้ดเป็นคำอธิบายขั้นตอนการทำงานของโปรแกรมโดยใช้ถ้อยคำผสมระหว่างภาษาอังกฤษและภาษาการเขียนโปรแกรมแบบโครงสร้างที่เข้าใจง่ายมาแสดงลำดับการทำงานของโปรแกรม หรืออาจใช้ภาษาไทยก็ได้ โดยให้ผู้เขียนโปรแกรมสามารถพัฒนาขั้นตอนต่าง ๆ ให้เป็นโปรแกรมได้ง่ายขึ้น แต่ส่วนใหญ่แล้วคำที่ใช้มักเป็นคำเฉพาะ (Reserve Word) ที่มีอยู่ในภาษาการเขียนโปรแกรมและมักจะเขียนด้วยตัวอักษรตัวใหญ่ ซูโดโค้ดที่ดีจะต้องมีความชัดเจน สั้น และได้ใจความ ข้อมูลต่าง ๆ ที่ใช้จะถูกเขียนอยู่ในรูปของตัวแปร ซูโดโค้ดนี้บางครั้งจะเรียกว่าอัลกอริทึมก็ได้ รูปแบบทั่วไปจะเป็นดังนี้

```
Algorithm < ชื่อของอัลกอริทึม >
1. ……..
2. ……..                    ←——— ขบวนการทำงานต่าง ๆ
……..
……..
END
```

ตัวอย่างเช่นในการเขียนซูโดโค้ดสำหรับให้คอมพิวเตอร์หาค่าเฉลี่ยจากข้อมูลที่รับเข้าทางแป้นพิมพ์อาจเขียนได้ดังนี้

```
Algorithm  การหาค่าเฉลี่ย
1. ตัวนับ = 0
2. ผลรวม = 0
3. รับค่าทางแป้นพิมพ์เก็บไว้ใน (ข้อมูล)
4. ถ้า ข้อมูล มากกว่า 0
                เพิ่มค่าตัวนับขึ้นหนึ่งค่า
                ผลรวม  =  ผลรวม + ค่าข้อมูล
                ย้อนกลับไปทำขั้นตอนที่ 3
   ถ้าไม่มากกว่าไปทำขั้นตอนที่ 5
5. ค่าเฉลี่ย  =  ผลรวมหารด้วยตัวนับ
6. แสดงค่าเฉลี่ยทางจอภาพ โดยมีทศนิยมสองตำแหน่ง
7. จบ
```

จะเห็นว่าขั้นตอนการหาค่าเฉลี่ยได้เขียนไว้จะเข้าใจได้ง่าย เราสามารถทราบได้ว่าในการทำงานต่าง ๆ จะต้องใช้ตัวแปรใดบ้าง แต่ละขั้นตอนมีการประมวลผลอย่างไร แต่โดยทั่วไปแล้วซูโดโค้ดจะถูกเขียนด้วยภาษาอังกฤษ ดังต่อไปนี้

```
Algorithm Avarage_Sum
1. count  =  0
2. sum  =  0
3. INPUT (value)
4. IF  value > 0  THEN
                count  =  count +1
                sum    =  sum + value
                GOTO 3
   ELSE GOTO 5
5. avarage = sum / count
6. OUTPUT (avarage)
7. END
```

ในการคำนวณหาพื้นที่สามเหลี่ยมเราอาจเขียนซูโดโค้ดได้ดังต่อไปนี้

```
ซูโดโค้ดหาพื้นที่สามเหลี่ยม
เริ่มต้น
        1. รับค่าความยาวของด้านที่เป็นฐานมาเก็บในตัวแปร X
        2. รับค่าความยาวของส่วนสูงมาเก็บในตัวแปร Y
        3. คำนวณพื้นที่โดย ARRAY = (X * Y)/2
        4. แสดงผลพื้นที่
จบ
```

หรืออาจเขียนเป็นภาษาอังกฤษได้เป็น

```
START
        1. READ X
        2. READ Y
        3. Compute ARRAY = (X * Y)/2
        4. Print ARRAY
END
```

5.2 คุณสมบัติพื้นฐานในการประมวลผลของคอมพิวเตอร์

ก่อนที่จะศึกษาวิธีการเขียนอัลกอริธึมควรมีความเข้าใจและคำนึงถึงคุณสมบัติพื้นฐานของระบบคอมพิวเตอร์ก่อน เพื่อนำไปประยุกต์ใช้ในขั้นตอนการเขียนอัลกอริธึมได้อย่างถูกต้อง แม้ว่าคอมพิวเตอร์จะเป็นอุปกรณ์แบบอิเล็กทรอนิกส์ที่ทำงานได้รวดเร็ว ไม่มีความผิดพลาด ไม่มีการทำงานในลักษณะเครื่องจักรที่มีกลไกการเคลื่อนไหว แต่การออกแบบการทำงานภายในก็ยังต้องอาศัยการกำหนดกลไกเชิงวิธีการทำงานไว้ด้วย ถ้าหากกำหนดกลไกการทำงานผิด คอมพิวเตอร์ก็จะทำงานแบบผิด ๆ ตามไปด้วย คุณสมบัติพื้นฐานของกลไกการทำงานมีดังนี้

- **คุณสมบัติด้านหน่วยความจำ**

ในงานเขียนโปรแกรมคอมพิวเตอร์ ต้องเกี่ยวข้องกับการใช้งานพื้นที่หน่วยความจำของระบบคอมพิวเตอร์ ในภาษาคอมพิวเตอร์ให้แทนสัญลักษณ์การกำหนดพื้นที่หน่วยความจำด้วยการกำหนดชื่อเป็นตัวแปรสำหรับใช้งาน เพื่อใช้อ้างถึงข้อมูลในหน่วยความจำนั่นเอง ตัวอย่างเช่น

X = 4 หมายถึง กำหนดค่า 4 เก็บไว้ในตัวแปร X

X = Y + Z หมายถึง การนำค่าในตัวแปร Y มาบวกกับค่าของตัวแปร Z แล้วเก็บผลลัพธ์ที่ได้ไว้ในตัวแปร X

X = X + 1 หมายถึง การเอาค่าในตัวแปร X ไปบวกเพิ่มอีก 1 ค่า แล้วเอาค่าผลลัพธ์ใหม่ไปเก็บในตัวแปร X แทนค่าเดิม

- **คุณสมบัติด้านการคำนวณ**

คุณสมบัติด้านการคำนวณในระบบคอมพิวเตอร์นั้น ในระดับพื้นฐานจะสามารถดำเนินการบวก ลบ คูณ หาร ได้ แต่ลักษณะการพิจารณาเลือกประมวลผลงานคำนวณของคอมพิวเตอร์นั้น มีความแต่ต่างจากระบบการคำนวณของมนุษย์ คือ คอมพิวเตอร์คำนวณโดยพิจารณาลำดับความสำคัญของสัญลักษณ์เครื่องหมายการคำนวณที่ปรากฏในนิพจน์การคำนวณนั้น ๆ เป็นสำคัญ มิใช่เพียงการคำนวณโดยยึดหลักเรียงจากซ้ายไปขวาแบบที่เราบวกเลขที่เขียนเป็นสมการบนกระดานเท่านั้น เช่น 2 + 3 x 4 จะได้ผลลัพธ์เป็น 14 แต่ถ้าเขียนเป็น 2 x 3 + 4 จะได้ผลลัพธ์เป็น 10

สัญลักษณ์ที่ใช้ในการคำนวณและลำดับการทำงานของการคำนวณ มีดังนี้

สัญลักษณ์	ความหมายในการทำงาน	ลำดับการทำงาน
+	บวก	3
-	ลบ	3
*	คูณ	2
/	หาร	2
** หรือ ^	ยกกำลัง	1

- **คุณสมบัติด้านการเปรียบเทียบเชิงตรรกะ**

ประสิทธิภาพการทำงานของเครื่องคอมพิวเตอร์ที่มีเหนือกว่าเครื่องคำนวณชนิดอื่นอย่างหนึ่งก็คือ ความสามารถในการประมวลผลเชิงเปรียบเทียบ โดยใช้หลักการทำงานของพีชคณิตมาทำงานเชิงตรรกะนั่นเอง

ประสิทธิภาพการทำงานของคำสั่งแต่ละภาษาก็คือ ความสามารถในการเปรียบเทียบเงื่อนไขที่ผู้เขียนคำสั่งกำหนดขึ้น เพื่อให้คอมพิวเตอร์สามารถประมวลผล ลักษณะการตัดสินใจพิจารณาเงื่อนไข โดยหากเงื่อนไขเป็นจริงให้เครื่องทำงานตามคำสั่งหนึ่ง หรือหากเงื่อนไขเป็นเท็จก็ให้เครื่องทำตามอีกคำสั่งหนึ่ง

การเขียนคำสั่งที่มีลักษณะของเงื่อนไขเพื่อให้เครื่องใช้ในการพิจารณาเลือกทิศทางการทำงานตามเงื่อนไขที่กำหนดนั้น มีสิ่งที่เกี่ยวข้องกับผู้พัฒนางานโปรแกรมก็คือต้องศึกษาวิธีการใช้สัญลักษณ์ในการเขียนประโยคคำสั่งแบบเงื่อนไขของแต่ละภาษาที่กำหนดให้เลือกใช้โดยทั่วไปมีสัญลักษณ์ของพีชคณิตบูลลีน (Boolean) ดังลักษณะต่อไปนี้

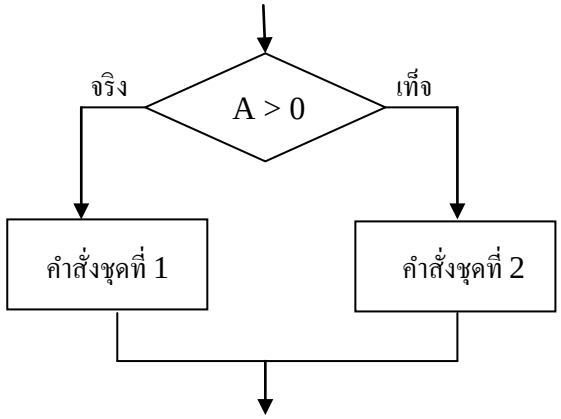
การใช้สัญลักษณ์เขียนประโยคคำสั่งเงื่อนไขแบบ 1 ประโยค

การเขียนประโยคคำสั่งเงื่อนไขแบบ 1 เงื่อนไขนั้น มีสัญลักษณ์ในการใช้งานดังนี้

สัญลักษณ์	ความหมายในการทำงาน
=	เท่ากับ
<	น้อยกว่า
>	มากกว่า
<=	น้อยกว่าหรือเท่ากับ
>=	มากกว่าหรือเท่ากับ
<>	ไม่เท่ากับ

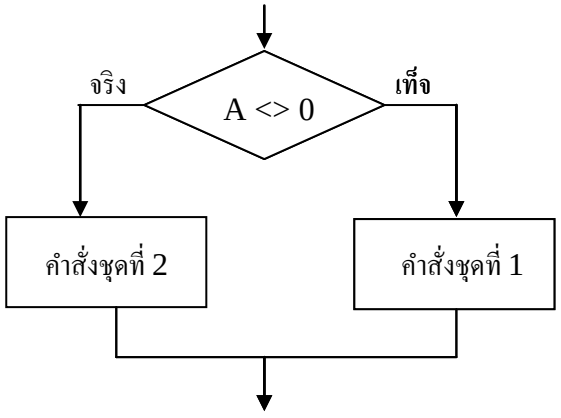
ตัวอย่างที่ 5.1 รูปแบบการเขียนนิพจน์โดยใช้เงื่อนไขแบบ 1 ประโยค เทียบกับการเขียนผังงาน

ถ้า A > 0 แล้ว (ถ้าเงื่อนไขเป็นจริง)
คำสั่งชุดที่ 1
(ถ้าเงื่อนไขเป็นเท็จ)
คำสั่งชุดที่ 2



หรือ

ถ้า A <> 0 แล้ว (ถ้าเงื่อนไขเป็นจริง)
คำสั่งชุดที่ 2
(ถ้าเงื่อนไขเป็นเท็จ)
คำสั่งชุดที่ 1



การใช้สัญลักษณ์เชื่อมประโยคคำสั่งเงื่อนไข

การเขียนคำสั่งกำหนดเงื่อนไขการทำงานในลักษณะเชื่อมประโยคเงื่อนไข 2 ประโยค ต้องใช้สัญลักษณ์คำสั่งเพิ่มเติม เพื่อให้ได้ข้อสรุปของการทำงานว่าเป็นค่าจริง (True) หรือค่าเป็นเท็จ (False) ดังนี้

- สมมติให้ผลลัพธ์การเปรียบเทียบในประโยคเงื่อนไขที่ 1 เป็นสัญลักษณ์ A
- สมมติให้ผลลัพธ์การเปรียบเทียบในประโยคเงื่อนไขที่ 2 เป็นสัญลักษณ์ B

เมื่อมีการทำงานหาข้อสรุปตั้งแต่ 2 เงื่อนไขประกอบกัน โดยใช้หลักการของพีชคณิตบูลลีน จะได้ดังตารางต่อไปนี้

A	B	(A) AND (B)	(A) OR (B)	NOT (A)
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

- การใช้สัญลักษณ์ AND ในการเชื่อมประโยคเงื่อนไขนั้น จะให้ผลลัพธ์สรุปเงื่อนไขว่าเป็นจริงได้ก็ต่อเมื่อประโยคทั้งสองประโยคมีค่าเป็นจริงเท่านั้น ถ้ามีประโยคใดประโยคหนึ่งเป็นเท็จ จะสรุปได้ว่าค่าที่ได้จะเป็นเท็จทันที
- การใช้สัญลักษณ์ OR ในการเชื่อมประโยคเงื่อนไขนั้น จะให้ผลลัพธ์สรุปเงื่อนไขว่าเป็นจริงได้ ก็ต่อเมื่อมีประโยคใดประโยคหนึ่งเป็นจริง จะสามารถสรุปได้ว่าค่านั้นเป็นจริงทันที
- การใช้สัญลักษณ์ NOT นำหน้าประโยคเงื่อนไขนั้น จะให้ผลลัพธ์สรุปในทางตรงกันข้าม

ตัวอย่างที่ 5.2 การเขียนนิพจน์ใช้เงื่อนไขแบบ 2 ประโยค โดยใช้สัญลักษณ์ AND

ถ้า (X > 10) AND (X <= 15) แล้ว (เงื่อนไขเป็นจริง)

คำสั่งชุดที่ 1

(เงื่อนไขเป็นเท็จ)

คำสั่งชุดที่ 2

จากตัวอย่าง ถ้าหากเงื่อนไขที่กำหนดเป็นจริง โปรแกรมจะทำคำสั่งชุดที่ 1 แต่ถ้าเงื่อนไขเป็นเท็จ โปรแกรมจะทำคำสั่งชุดที่ 2 จากเงื่อนไขที่กำหนดในตัวอย่าง ค่าในหน่วยความจำ X ต้องมีค่ามากกว่า 10 แต่ไม่เกิน 15 จึงจะทำให้ข้อสรุปของประโยคเงื่อนไขดังกล่าวเป็นจริง นอกเหนือจากข้อสรุปนี้จะเป็นเท็จ

ตัวอย่างที่ 5.3 การเขียนนิพจน์ใช้เงื่อนไขแบบ 2 ประโยค โดยใช้สัญลักษณ์ OR

ถ้า (X > 50) OR (X <= 60) แล้ว (เงื่อนไขเป็นจริง)

คำสั่งชุดที่ 1

(เงื่อนไขเป็นเท็จ)

คำสั่งชุดที่ 2

จากตัวอย่างถ้าค่าในหน่วยความจำ X มีค่ามากกว่า 50 หรือมีค่าน้อยกว่าหรือเท่ากับ 60 แล้วจะทำให้ได้ข้อสรุปของประโยคเงื่อนไขนี้เป็น “จริง”

- **คุณสมบัติด้านการแสดงผลค่าข้อมูล**

เป็นการอ่านค่าข้อมูลจากพื้นที่หน่วยความจำที่เขียนคำสั่งนำไปเก็บไว้ หรือจากการคำนวณที่ต้องมีการนำค่าไปเก็บไว้ เพื่อนำมาแสดงผลลัพธ์ในรูปแบบ และในตำแหน่งงานที่ต้องการ

- **คุณสมบัติการจัดลำดับการทำงาน**

คอมพิวเตอร์จะทำงานทีละคำสั่ง ตามลำดับจากบนลงล่าง กล่าวคือ คอมพิวเตอร์จะทำงานตามคำสั่งที่อยู่บรรทัดบนสุดก่อน แล้วจึงทำงานตามคำสั่งที่อยู่ในลำดับต่อมา จนถึงคำสั่งในบรรทัดสุดท้าย

5.3 พื้นฐานการเขียนซูโดโค้ด

แม้ว่าการเขียนซูโดโค้ดจะไม่มีรูปแบบที่แน่นอน แต่โดยทั่วไปแล้วมักจะทำกันดังลักษณะต่อไปนี้

การรับข้อมูลเข้าและการแสดงผลข้อมูล

ในการรับข้อมูลจะนิยมใช้คำว่า READ หรือ GET หรือ INPUT ตามด้วยตัวแปรที่ต้องการใช้เก็บข้อมูล ถ้าหากมีตัวแปรหลายตัวจะใช้เครื่องหมายคอมมา (“,”) คั่น ส่วนการแสดงผลมักใช้คำว่า PRINT หรือ WRITE สำหรับการรับข้อมูลมีรูปแบบดังนี้

```
READ    VAR1, VAR2,…..
```

หรือ

```
INPUT   VAR1, VAR2,……
```

หมายความว่ารับข้อมูลเข้ามาเก็บในตัวแปรตัวที่ 1,2,……โดย VAR มาจากคำว่า Variable หมายถึงตัวแปร

ตัวอย่าง

```
READ    X,Y       ;หมายความว่ารับข้อมูลมาเก็บในตัวแปร X และ Y ตามลำดับ
```

ตัวอย่าง

```
Read student_name                   ;อ่านค่ามาเก็บในตัวแปรชื่อ student_name
Get  system_date                    ;อ่านค่ามาเก็บในตัวแปรชื่อ system_date
Read  number_1, number_2            ;อ่านค่ามาเก็บในตัวแปร number_1,number_2
```

สำหรับการแสดงข้อมูลมีรูปแบบดังนี้

```
PRINT   VAR1, VAR2,…….
```

หรือ

```
WRITE   VAR1, VAR2,…….
```

หมายความว่าให้แสดงข้อมูลที่ถูกเก็บในตัวแปรตัวที่ 1,2,……

ตัวอย่าง

```
WRITE   DATA1          ;แสดงข้อมูลที่อยู่ในตัวแปร DATA1
PRINT   A, B           ;แสดงข้อมูลที่อยู่ในตัวแปร A และตัวแปร B
```

การคำนวณ

ในการประมวลผลแบบคำนวณจะขึ้นต้นด้วยคำว่า Compute หรือ Calculate แล้วตามด้วยตัวแปรที่ต้องการเก็บค่าจากการคำนวณ เครื่องหมายเท่ากับและนิพจน์การคำนวณ ซึ่งประกอบไปด้วยเครื่องหมายการกระทำทางคณิตศาสตร์

ตัวอย่าง

```
Compute         ARRAT  =  (X * Y)/2
```

หมายความว่าให้ตัวแปร ARRAT มีค่าเท่ากับการนำตัวแปร X คูณกับตัวแปร Y แล้วหารด้วยสอง

ตัวอย่าง

```
Compute         Profit  =  Price – Cost
```

หมายความว่ากำไรคำนวณได้จากตัวแปรที่เก็บราคาลบด้วยต้นทุน

การกำหนดค่า

การกำหนดค่าเริ่มต้นให้กับตัวแปรจะใช้คำว่า INIT และ SET ถ้าหากเป็นการประกาศตัวแปรจะต้องระบุด้วยว่าเป็นตัวแปรประเภทใด และถ้าเป็นการประกาศค่าคงที่จะใช้เครื่องหมายเท่ากับในการกำหนดค่า

ตัวอย่าง

```
INIT    A, B    : INTEGER      ;ให้ตัวแปร A และ B เป็นตัวแปรที่เก็บเลขจำนวนเต็ม
INIT    Y       :REAL          ;ให้ตัวแปร Y เป็นตัวแปรที่เก็บเลขทศนิยม
SET     A = 8                  ;กำหนดให้ตัวแปร A มีค่าเท่ากับ 8
```

ตัวอย่างที่ 5.4 การเขียนซูโดโค้ดคำนวณราคาเครื่องคอมพิวเตอร์รวมกับภาษี 7%

```
ComputeTotal
        init  total, price, vat  :  real          \** ประกาศตัวแปร total, price, vat
        init  vat  =  0.07                        \** ประกาศ vat เป็นค่าคงที่ 0.07
        read  price                               \** รับราคาจากการป้อนข้อมูล
        compute  total  =  price + (price x 0.07) \**คำนวณราคาบวก vat
        print  price, total                       \**แสดงผลราคา และราคารวม
end                                               \**จบโปรแกรม
```

5.4 การเขียนซูโดโค้ดสำหรับตัดสินใจและทดสอบทางเลือก

การเขียนโปรแกรมที่ต้องมีการเลือกทิศทางการทำงานนั้นในโปรแกรมจะต้องมีการเปรียบเทียบหรือตัดสินใจเกินขึ้นในโปรแกรม วิธีการตัดสินใจของโปรแกรมจะต้องมีการเปรียบเทียบเงื่อนไขเกิดขึ้น โดยผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จอย่างใดอย่างหนึ่งเท่านั้น

การตัดสินใจเพื่อเลือกทำระหว่างทางสองทางจะใช้คำว่า IF หรือ IF-THEN-ELSE และ ENDIF โดยจะเปรียบเทียบเงื่อนไข ถ้าเงื่อนไขเป็นจริงจะทำกลุ่มคำสั่ง (Statement) กลุ่มหนึ่ง ถ้าเป็นเท็จจะทำกลุ่มคำสั่งอีกกลุ่มหนึ่ง โดยมีรูปแบบดังนี้

```
IF  ตรวจสอบเงื่อนไข      THEN          ;เริ่มต้น
        กลุ่มคำสั่ง 1                  ;ถ้าเป็นจริงทำกลุ่มคำสั่งนี้
ELSE
        กลุ่มคำสั่ง 2                  ;ถ้าเป็นเท็จทำกลุ่มคำสั่งนี้
ENDIF                                  ;จบการทำงาน
```

ตัวอย่างที่ 5.5 ตัวอย่างนี้จะนำตัวเลขในตัวแปร number มาทดสอบถ้ามากกว่า 0 ให้แสดงว่าเป็นเลขบวก แต่ถ้าไม่มากกว่า 0 ให้แสดงว่าเป็นเลขลบ

```
IF  number > 0 THEN
        PRINT  POSITIVE NUMBER
ELSE
        PRINT  NEGATIVE NUMBER
ENDIF
```

จากตัวอย่างหมายความว่าถ้าค่า number มีค่ามากกว่า 0 ให้คอมพิวเตอร์พิมพ์คำว่า POSITIVE NUMBER ถ้าไม่มากกว่าจะพิมพ์คำว่า NEGATIVE NUMBER

สำหรับกรณีที่มีทางเลือกมากกว่าสองทางจะใช้คำว่า CASE , OF และ ENDCASE โดยจะตรวจสอบว่าตัวแปรที่อยู่หลัง CASE มีค่าเท่ากับค่าคงที่ตัวใด ก็จะทำกลุ่มคำสั่งที่อยู่หลังค่าคงที่ตัวนั้น ตัวอย่างเช่น

```
CASE  num OF
        1 :     PRINT 11111
        2 :     PRINT 22222
        3 :     PRINT 33333
ENDCASE
```

จากตัวอย่างถ้าค่าในตัวแปร num เป็น 1 จะให้พิมพ์คำว่า 11111 ถ้าตัวแปร num มีค่าเป็น 2 จะให้พิมพ์คำว่า 22222

5.5 การเขียนซูโดโค้ดแบบวนซ้ำ

ในการทำซ้ำหมายความว่าให้ระบบทำงานซ้ำ ๆ ตามเงื่อนไขที่กำหนด โดยจะมีการเปรียบเทียบเงื่อนไขในการทำซ้ำ แบ่งออกได้สามรูปแบบดังนี้

1. การทำซ้ำที่ทราบจำนวนครั้งในการทำซ้ำ การทำซ้ำแบบนี้จะมีการตั้งจำนวนการทำซ้ำไว้และมีการเพิ่มค่าในแต่ละรอบ จะใช้คำว่า FOR , DO และ ENDFOR โดยมีคำว่า IN STEPS OF เป็นการบอกค่าที่เพิ่มในแต่ละรอบ ถ้าไม่มีคำว่า IN STEPS OF หมายความว่าเพิ่มค่ารอบละหนึ่ง โดยมีรูปแบบดังนี้

```
FOR     กำหนดรอบการทำซ้ำ
        Statement
ENDFOR
```

2. การทำซ้ำจนระบบมีเงื่อนไขอย่างหนึ่งจึงหยุดทำ จะใช้คำว่า REPEAT – UNTIL ดังรูปแบบต่อไปนี้

```
REPEAT
        Statement_1
        ……..
UNTIL (Condition)
```

3. ถ้าเงื่อนไขเป็นจริงจะทำคำสั่งภายใน จะใช้คำว่า WHILE - ENDWHILE โดยจะตรวจสอบเงื่อนไขก่อนที่ทำชุดคำสั่งภายใน ดังรูปแบบต่อไปนี้

```
WHILE (Condition)
        Statement_1
        ……..
ENDWHILE
```

ตัวอย่างที่ 5.6 จงเขียนซูโดโค้ดให้มีการพิมพ์ตัวเลขตั้งแต่ 1 – 10 โดยใช้ repeat-until

```
Print number 1 to 10
    init  num  : integer              /**ประกาศตัวแปร num เป็นตัวเลข
    init  num = 1                     /**ให้ค่า num เริ่มต้นเท่ากับ 1
    repeat                            /**การทำงานวนซ้ำ (จนกว่า num = 10)
        print  num                    /**พิมพ์ค่าของ num จากค่าเริ่มต้น 1 - 10
        increase  num (num+1)         /**เพิ่มค่า num ขึ้น 1 เพื่อลดการวนซ้ำ
    until  num > 10                   /**เมื่อ num มากกว่า 10 จบการทำงานแบบวนซ้ำ
```

ตัวอย่างที่ 5.7 จงเขียนซูโดโค้ดให้มีการพิมพ์ตัวเลขตั้งแต่ 1 – 10 โดยใช้ while

```
Print number 1 to 10
        init  num  :  integer              /**ประกาศตัวแปร num เป็นตัวเลข
        init  num  =  1                    /**กำหนดให้ค่า num เริ่มต้นเท่ากับ 1
        while (num <= 10)                  /**ถ้า num น้อยกว่า 10 ให้ทำคำสั่งวนซ้ำ
                print  num                 /**พิมพ์ค่า num จากค่าเริ่มต้น 1 – 10
                increase num (num + 1)     /**เพิ่มค่า num ขึ้น 1 ค่า เพื่อลดการวนซ้ำ
        endwhile                           /**จบการทำงานแบบวนซ้ำ
```

ตัวอย่างที่ 5.8 จงเขียนซูโดโค้ดสำหรับบวกเลข 1 + 2 + 3 + ….+100 แล้วพิมพ์ผลลัพธ์ออกมา

วิธีทำ ตัวอย่างนี้สามารถนำรูปแบบของ while มาใช้ได้ โดยให้โปรแกรมกำหนดค่าเริ่มต้นให้กับตัวนับเป็น 1 และเพิ่มค่าขึ้นครั้งละหนึ่ง จนกว่ามีการตรวจสอบว่าค่าตัวนับมีค่าน้อยกว่าหรือเท่ากับ 100 มีค่าเป็นเท็จจึงหยุดนับ โดยสามารถเขียนซูโดโค้ดได้ดังนี้

```
Sum_number_1_100
        init  i, sum  :  integer           /**ประกาศตัวแปร i และ sum เป็นจำนวนเต็ม
        init  i = 0                        /**กำหนดค่าเริ่มต้นให้ i มีค่าเป็น 0
        init  sum  =  0
        while (i <= 100)                           /**ทำซ้ำเมื่อ i น้อยกว่าหรือเท่ากับ 100
                compute  sum  =  sum + i           /**นำค่า i มาบวกกับ sum แล้วเก็บใน sum
                compute  i  =  i + 1               /**เพิ่มค่าในตัวแปร i ขึ้นหนึ่งค่า
        endwhile
        print  sum                                 /**แสดงค่าผลรวมที่เก็บอยู่ในตัวแปร sum
```

ตัวอย่างที่ 5.9 จงเขียนซูโดโค้ดให้มีการพิมพ์ตัวเลขตั้งแต่ 1 – 10 โดยใช้ for

```
Number 1-10
        init  num  :  integer              /**ประกาศตัวแปร num เป็นตัวเลข
        for  num  =  1  to  10  do         /**ตั้งค่า num = 1 และให้วนรอบ 10 ครั้ง
                print  num                 /**พิมพ์ค่าของ num จากค่าเริ่มต้น 1 - 10
        endfor                             /**จบการทำงานแบบวนซ้ำ
```

5.6 การเขียนซูโดโค้ดเพื่อเรียกโปรแกรมย่อยและกระโดดข้าม

สำหรับการเรียกโปรแกรมย่อยหรือโพซีเยอร์จะใช้คำว่า CALL แล้วตามด้วยชื่อโปรแกรมย่อยหรือโพซีเยอร์ มีรูปแบบดังนี้

CALL ชื่อโปรแกรมย่อย

การกระโดดข้ามไปทำชุดคำสั่งใด ๆ จะใช้คำว่า LABEL กำหนดตำแหน่งที่จะกระโดดมา และใช้คำว่า GOTO ในตำแหน่งที่จะกระโดด ตัวอย่างเช่น

```
START :
            Statement_1
            ……..
AB1:        …….
            ……
            GOTO AB1
END
```

แบบฝึกหัดท้ายบท

ตอนที่ 1 จงเลือกคำตอบที่ถูกต้องที่สุดเพียงหนึ่งข้อ

1. ท่านคิดว่า รหัสเทียม ถูกเขียนขึ้นมาในขั้นตอนใด
   - ก. การทดลองแก้ปัญหา
   - ข. การเขียนขั้นตอนวิธีการแก้ปัญหา
   - ค. การทดสอบการแก้ปัญหา
   - ง. การพิจารณาลักษณะข้อมูลอินพุตและเอาต์พุต
2. ท่านคิดว่า รหัสเทียมจะถูกนำไปใช้ในขั้นตอนใดต่อไป
   - ก. การเขียนโปรแกรม
   - ข. การอธิบายการทำงานของโปรแกรม
   - ค. การพิจารณาข้อมูลอินพุตและเอาต์พุต
   - ง. การแก้ไขโปรแกรม
3. ในการเขียนรหัสเทียมข้อใดไม่ถูกต้อง
   - ก. ใช้คำว่า READ ในการรับข้อมูล
   - ข. ใช้คำว่า FOR ในการทำซ้ำ
   - ค. ใช้คำว่า MEM แทนชื่อตัวแปรในหน่วยความจำ
   - ง. ใช้คำว่า CALL ในการเรียกโพซีเยอร์หรือโปรแกรมย่อย
4. ถ้าหากมีการเขียนซูโดโค้ดดังต่อไปนี้ ท่านคิดว่าทำอะไร

   NUMBER

```
INIT  NUM : INTEGER
INIT  NUM = 1
WHILE  (NUM <= 100)
        PRINT  NUM
        INCRAESE NUM
ENDWHILE
```

ก. หาผลรวมของตัวเลขตั้งแต่ 1 ถึง 100

ข. พิมพ์ตัวเลขตั้งแต่ 1 ถึง 100

ค. พิมพ์ตัวเลขที่มีค่าน้อยกว่า 100

ง. ไม่สามารถหาค่าได้

5. จากข้อ 5 ท่านคิดว่าคำว่า INCRAESE NUM คืออะไร

ก. เพิ่มค่าตัวแปร NUM ขึ้นหนึ่งค่า

ข. ลดค่าตัวแปร NUM ลงหนึ่งค่า

ค. นำค่า NUM บวกกับค่า NUM ที่ผ่านมา

ง. ไม่สามารถบอกได้

ตอนที่ 2 จงตอบคำถามต่อไปนี้

1. ท่านคิดว่ารหัสเทียมหรือซูโดโค้ดต่างจากอัลกอริทึมอย่างไร

……

……..

……..

2. จงอธิบายข้อดีและข้อเสียของการเขียนรหัสเทียมก่อนเขียนโปรแกรม และไม่เขียนรหัสเทียมก่อนเขียนโปรแกรม

……..

……..

……..

……..

3. ท่านคิดว่าในการกำหนดค่าตัวแปร เหตุใจจึงต้องระบุชนิดของตัวแปร

……

……..

……..

4. จงเขียนซูโดโค้ดในการรับค่าตัวเลขเข้าไป 10 ค่า แล้วให้แสดงผลค่าสูงสุดและค่าต่ำสุด

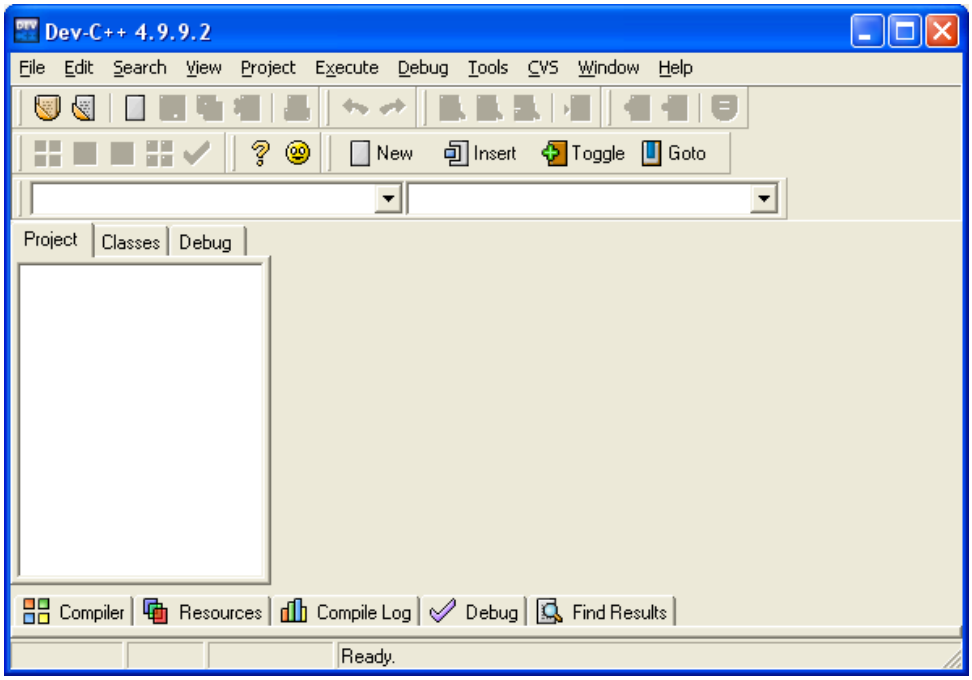
บทที่ 6
การเขียนโปรแกรมภาษาซี

การเขียนโปรแกรมคอมพิวเตอร์นั้นมีภาษาให้เลือกอยู่หลายภาษา สำหรับภาษาซีเป็นภาษาที่เหมาะสำหรับผู้เริ่มต้นการเขียนโปรแกรมคอมพิวเตอร์ หากเริ่มศึกษาด้วยภาษานี้แล้วการศึกษาการเขียนโปรแกรมภาษาอื่น ๆ จะทำได้ง่ายขึ้น ภาษาซีเป็นภาษาที่มีการพัฒนามายาวนาน ในปัจจุบันได้พัฒนาให้สามารถเขียนโปรแกรมเชิงวัตถุได้ ซึ่งอยู่ในรูปแบบของภาษา C++ และ C#

6.1 ขั้นตอนการพัฒนาโปรแกรมด้วยภาษาซี

ขั้นตอนการเขียนโปรแกรมคอมพิวเตอร์ด้วยภาษาระดับสูงนั้นโปรแกรมเมอร์จะต้องเขียนโปรแกรมต้นฉบับที่เรียกว่าซอสซ์โค้ดขึ้นมาก่อนสำหรับภาษาซีโปรแกรมนี้จะมีนามสกุลเป็น .c จากนั้นจึงจะใช้โปรแกรมคอมไพเลอร์มาแปลภาษาให้เป็นภาษาที่เครื่องคอมพิวเตอร์สามารถทำงานได้ จากนั้นจึงทำการเชื่อมโยงหรือลิงค์กับไฟล์ต่าง ๆ ที่เกี่ยวข้องให้สามารถทำงานบนเครื่องคอมพิวเตอร์ที่ใช้งานอยู่ได้

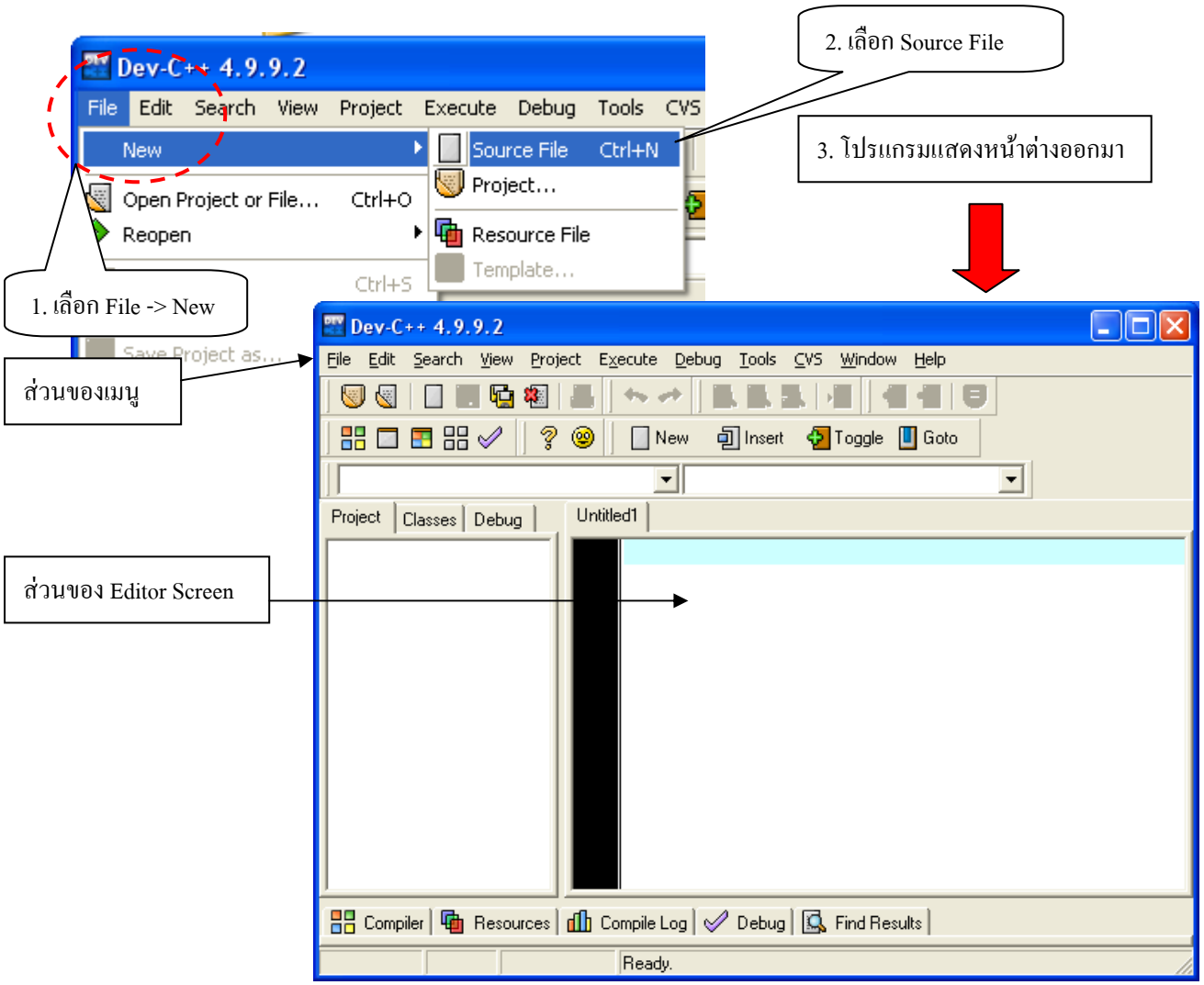
ในการเขียนโปรแกรมภาษาซีนั้นมีตัวแปลภาษาหรือคอมไพเลอร์ให้ใช้งานหลายตัว โดยออกมาในรูปของซอฟต์แวร์ที่ทำให้โปรแกรมเมอร์สามารถเขียนโปรแกรมได้ง่าย โดยขบวนการทั้งสามขั้นตอนจะทำได้ในครั้งเดียว โดยซอฟต์แวร์ลักษณะนี้ได้รวมโปรแกรมพัฒนางาน หรือ ไอดีอี (Integrated Development Environment : IDE) เอาไว้ด้วย สำหรับโปรแกรมที่แนะนำในการศึกษาภาษาซี ในที่นี้ได้แก่ DEV C++ ซึ่งเป็นโปรแกรมที่มีขนาดเล็ก สามารถใช้เริ่มต้นศึกษาการเขียนโปรแกรมได้เป็นอย่างดี และยังสามารถเขียนโปรแกรมแบบ C++ ได้อีกด้วย เมื่อเปิดโปรแกรมขึ้นมา หน้าตาของโปรแกรมเป็นดังนี้



ส่วนประกอบที่สำคัญของโปรแกรม DEV C++ ในการใช้งานเบื้องต้น มีดังนี้

1. **ส่วนของเมนู** เป็นคำสั่งต่าง ๆ ที่จำเป็นสำหรับใช้งานโปรแกรม เช่น การสร้างไฟล์ใหม่ การคอมไพล การรันโปรแกรม นอกจากนี้ในด้านล่างของเมนูยังมีทูลบาร์ที่รวบรวมคำสั่งต่าง ๆ ที่ใช้งานบ่อย ๆ ในรูปของไอคอนให้ผู้ใช้เลือกใช้ได้ง่าย ๆ อีกด้วย
2. **ส่วนของ Editor Screen** ใช้สำหรับสร้างหรือแก้ไขโปรแกรม โดยที่ผู้เขียนโปรแกรมจะพิมพ์คำสั่งต่าง ๆ ลงในบริเวณนี้

ในการใช้งานเบื้องต้น ถ้าหากต้องการเริ่มเขียนโปรแกรมจะต้องสร้างไฟล์ขึ้นมาใหม่ โดยเลือกเมนู File เลือก New เพื่อสร้างงานใหม่ เลือก Source File โปรแกรมจะแสดง Editor Screen ให้เขียนโปรแกรมและตั้งชื่อโปรแกรมให้มีชื่อเป็น Untitled1 โดยอัตโนมัติ



เมื่อถึงตรงนี้จะสามารถเขียนโปรแกรมภาษาซีได้ โดยการเขียนโปรแกรมจะต้องเขียนตามโครงสร้างที่กำหนด ถ้าหากเขียนผิดโปรแกรมจะแจ้งข้อผิดพลาดออกมาตอนคอมไพโปรแกรม

6.2 โครงสร้างโปรแกรมภาษาซี

การเขียนโปรแกรมแต่ละภาษานั้นโครงสร้างของโปรแกรมจะต่างกัน ในบทนี้จะกล่าวถึงโครงสร้างของการเขียนโปรแกรมด้วยภาษาซี (C/C++) รวมทั้งการเขียนโปรแกรมอย่างง่าย ลักษณะโครงสร้างของภาษาซีแบ่งออกได้เป็น 5 ส่วนดังต่อไปนี้

1. พรีโพรเซสเซอร์ไดเร็กทีฟ (Preprocessor directives)
2. ส่วนการกำหนดค่า (Global declarations)
3. ส่วนฟังก์ชันหลัก (The main() function)
4. การสร้างฟังก์ชันและการใช้ฟังก์ชัน (Uses-defined function)
5. ส่วนอธิบายโปรแกรม (Program comments)

โครงสร้างของโปรแกรมประกอบด้วยหลายส่วน แต่ในการเขียนโปรแกรมนั้นไม่จำเป็นต้องเขียนหมดทุกส่วน ส่วนใดไม่ใช้ก็สามารถตัดทิ้งได้ แต่ทุกโปรแกรมต้องมีส่วนพรีโพรเซสเซอร์ไดเร็กทีฟ และส่วนฟังก์ชันหลัก รายละเอียดของส่วนต่าง ๆ เป็นดังต่อไปนี้

พรีโพรเซสเซอร์ไดเรกทีฟ (Preprocessor directives)

ส่วนนี้ทุกโปรแกรมต้องมี จะใช้สำหรับเรียกไฟล์ที่โปรแกรมต้องการในการทำงานและกำหนดค่าต่าง ๆ โดยคอมไพล์เลอร์จะกระทำตามคำสั่งก่อนที่จะคอมไพล์โปรแกรม ซึ่งจะต้องเริ่มต้นด้วยเครื่องหมาย ไดเร็กทีฟ (directive) # และตามด้วยชื่อโปรแกรมหรือชื่อตัวแปรที่ต้องการกำหนดค่า ส่วนนี้อาจเรียกอีกชื่อหนึ่งว่าส่วนหัวโปรแกรม (Header Part) สำหรับ Directive ที่ใช้กันบ่อย ๆ ได้แก่

- **#include** เป็นการแจ้งให้คอมไพเลอร์อ่านไฟล์อื่นเข้ามาคอมไพร่วมด้วย รูปแบบการใช้จะทำโดยเขียน #include แล้วตามด้วยชื่อไฟล์ ดังนี้

#include “stdio.h”	หมายความว่า อ่านไฟล์ stdio.h เข้ามาด้วย
#include “Pro1.c”	หมายความว่า อ่านไฟล์ Pro1.c เข้ามาด้วย

การกำหนดชื่อไฟล์ตามหลัง #include นั้นอาจใช้เครื่องหมาย <> คร่อมชื่อไฟล์ก็ได้ ซึ่งจะเป็นการอ่านไฟล์จากไดเร็กทอรีที่กำหนดไว้ก่อน แต่ถ้าใช้ “ “ เป็นการอ่านไฟล์จากไดเร็กทอรีปัจจุบันที่กำลังติดต่ออยู่ และไฟล์ที่จะ include เข้ามานี้จะต้องไม่มีฟังค์ชัน main() โดยมากแล้วจะประกอบด้วยโปรแกรมย่อย ค่าคงที่ หรือข้อกำหนดต่าง ๆ

- **#define** เป็นการกำหนดค่านิพจน์ต่าง ๆ ให้กับชื่อของตัวแปร โดยมีรูปแบบดังนี้

#define NAME VALUE	เช่น
#define END 20	กำหนด END มีค่าเท่ากับ 20
#define A 5*6+3	กำหนด A มีค่าเป็น 5*6+3

ส่วนประกาศ (Global declarations)

ส่วนนี้จะใช้ในการประกาศตัวแปรหรือฟังก์ชันที่ต้องใช้ในโปรแกรม โดยทุก ๆ ส่วนของโปรแกรมสามารถจะเรียกใช้ข้อมูลที่ประกาศไว้ในส่วนนี้ได้ ส่วนนี้บางโปรแกรมอาจไม่มีก็ได้ สำหรับรายละเอียดต่าง ๆ จะได้กล่าวต่อไป

ส่วนฟังก์ชันหลัก (main() function)

ส่วนนี้ทุกโปรแกรมจะต้องมี จะประกอบไปด้วยประโยคคำสั่งต่าง ๆ ที่จะให้โปรแกรมทำงาน โดยนำคำสั่งต่าง ๆ มาต่อเรียงกัน แต่ละประโยคคำสั่งจะจบด้วยเครื่องหมายเซมิโคลอน (Semi colon ;) โดยโปรแกรมหลักนี้จะเริ่มต้นด้วยคำว่า main() ตามด้วยเครื่องหมายปีกกาเปิด { และจบด้วยเครื่องหมายปีกกาปิด } ถ้าหากในโปรแกรมมีหลายฟังก์ชัน ส่วนของฟังก์ชัน main() นี้จะเป็นฟังก์ชันแรกที่โปรแกรมจะทำงาน

ส่วนกำหนดฟังก์ชันขึ้นใช้เอง (Uses – defined functions)

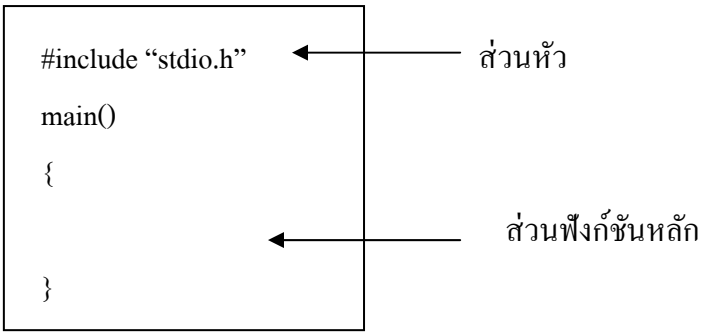
เป็นการเขียนคำสั่งและฟังก์ชันต่าง ๆ ขึ้นใช้ในโปรแกรม โดยต้องอยู่ในเครื่องหมาย { } และสร้างฟังก์ชันหรือคำใหม่ที่ให้ทำงานตามที่เราต้องการให้กับโปรแกรมและสามารถเรียกใช้ได้ ภายในโปรแกรม ตัวอย่างเช่น

```
#include “stdio.h”
main()
{
        function();                 /*เรียกใช้ฟังก์ชันที่สร้างขึ้น */
}
function()                          /*สร้างฟังก์ชันใหม่ โดยให้ชื่อว่า function
{
        return ;                    /*คืนค่าที่เกิดจากการทำฟังก์ชัน
}
```

ส่วนอธิบายโปรแกรม (Program comments)

ส่วนนี้ใช้เขียนคอมเมนต์โปรแกรม เพื่ออธิบายการทำงานต่าง ๆ ทำให้ผู้ศึกษาโปรแกรมในภายหลังทำความเข้าใจโปรแกรมได้ง่ายขึ้น เมื่อคอมไพล์โปรแกรมส่วนนี้จะถูกข้ามไป การเขียนคำอธิบายนี้จะเริ่มด้วยเครื่องหมาย /* และปิดด้วยเครื่องหมาย */ แต่ถ้าหากเขียนคำอธิบายทีละบรรทัดจะใช้เครื่องหมาย // เขียนกำกับบรรทัดคำอธิบายไว้ก็ได้

ดังที่กล่าวมาแล้วว่าส่วนประกอบของโปรแกรมมีหลายส่วน ส่วนใดไม่ใช้สามารถตัดทิ้งได้ภาษา C เป็นภาษาที่การเขียนโปรแกรมเป็นแบบโครงสร้างโมดูล โดยจะมีการเขียนโมดูลต่าง ๆ เก็บไว้ใช้ ซึ่งแต่ละโมดูลสามารถเรียกมาใช้ในภายหลังได้ ในที่นี้จะเขียนโปรแกรมอย่างง่ายที่สุดก่อน จากนั้นจะพัฒนาโปรแกรมให้ใหญ่ขึ้นโดยเขียนโปรแกรมแทรกเข้าไปในโปรแกรมอย่างง่ายนี้ ตัวอย่างการเขียนโปรแกรมขั้นต้นเป็นดังนี้

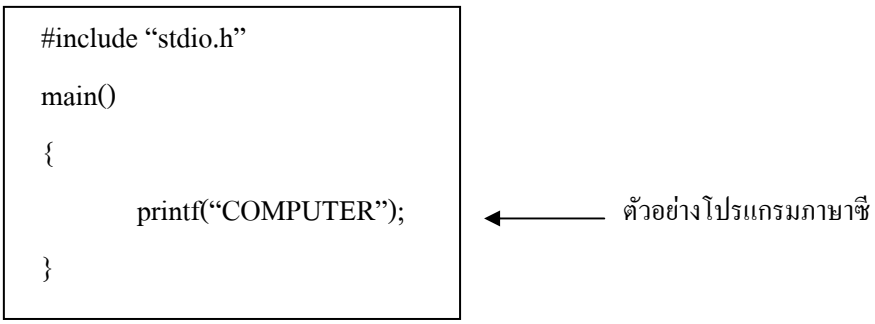


จากตัวอย่างโปรแกรมอย่างง่ายจะประกอบด้วยสองส่วนคือ

1. ส่วนที่หนึ่งเป็นส่วนหัวหรือส่วนเรียกโมดูลอื่น ๆ เข้ามาแปรความหมายร่วม โดยเป็นส่วนของพรีโพรเซสเซอร์ไดเร็กทีฟ ซึ่งอาจเป็นโมดูลมาตรฐานที่มีอยู่แล้วในโปรแกรมหรือโมดูลใหม่ที่สร้างขึ้นมาเอง โดยโมดูลเหล่านี้จะบรรจุคำสั่งหรือข้อกำหนดต่าง ๆ ที่จะใช้ในโปรแกรมที่เขียนขึ้น

2. ส่วนที่สอง เรียกว่าส่วนฟังก์ชันหลักซึ่งเป็นส่วนคำสั่งหรือสเตตเมนต์ (Statement) จะเป็นส่วนที่เก็บคำสั่งต่าง ๆ ที่จะให้โปรแกรมทำงาน โดยจะเริ่มต้นด้วยเครื่องหมายปีกกา { และจบด้วยเครื่องหมายปีกกาปิด } ส่วนนี้จะมีมากกว่าหนึ่งฟังก์ชันก็ได้ แต่ทุกโปรแกรมจะต้องมีฟังก์ชัน main ซึ่งถือว่าเป็นฟังก์ชันหลักที่โปรแกรมจะทำงานได้

โปรแกรมต่อไปนี้จะเป็นโปรแกรมแรกที่จะใช้เขียนโปรแกรมภาษาซี โดยจะใช้ฟังก์ชัน printf() ซึ่งจะทำหน้าที่พิมพ์ข้อมูลออกทางจอภาพ

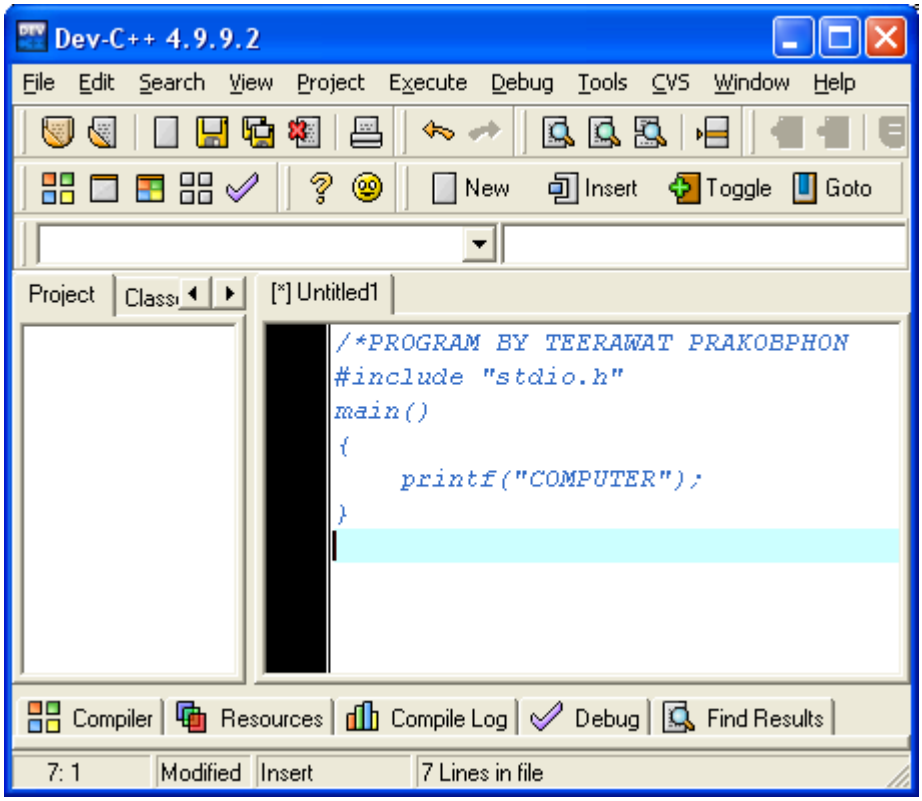


โปรแกรมนี้เมื่อทำงานคอมพิวเตอร์จะพิมพ์คำว่า COMPUTER ออกทางจอภาพ พิจารณาจากโปรแกรมจะเห็นว่าในฟังก์ชันหลักมีการเรียกใช้ฟังก์ชัน printf() ซึ่งจะทำหน้าที่พิมพ์ข้อความหรือสตริง (string) ที่อยู่ในเครื่องหมายคำพูดออกทางหน้าจอ และจบฟังก์ชันด้วยเครื่องหมายเซมิโคลอน โดยฟังก์ชันนี้จะเก็บไว้ใน stdio (ย่อมาจาก standard input output) ซึ่งจะเก็บชุดคำสั่งเกี่ยวกับการส่งข้อมูลเข้าออกเอาไว้ เราจึงต้องเรียก stdio.h ขึ้นมา สำหรับ #include เรียกว่า ไดเรกทีฟ (directive) และ stdio.h เรียกว่า ไฟล์ส่วนหัว (header file) สำหรับในการเขียนโปรแกรมเราสามารถเขียนคำอธิบาย (comments) ต่าง ๆ ได้ โดยคอมไพเลอร์จะไม่แปรความหมายเป็นภาษาเครื่อง ซึ่งจะต้องเปิดด้วยเครื่องหมาย /* และปิดด้วยเครื่องหมาย */ เช่น

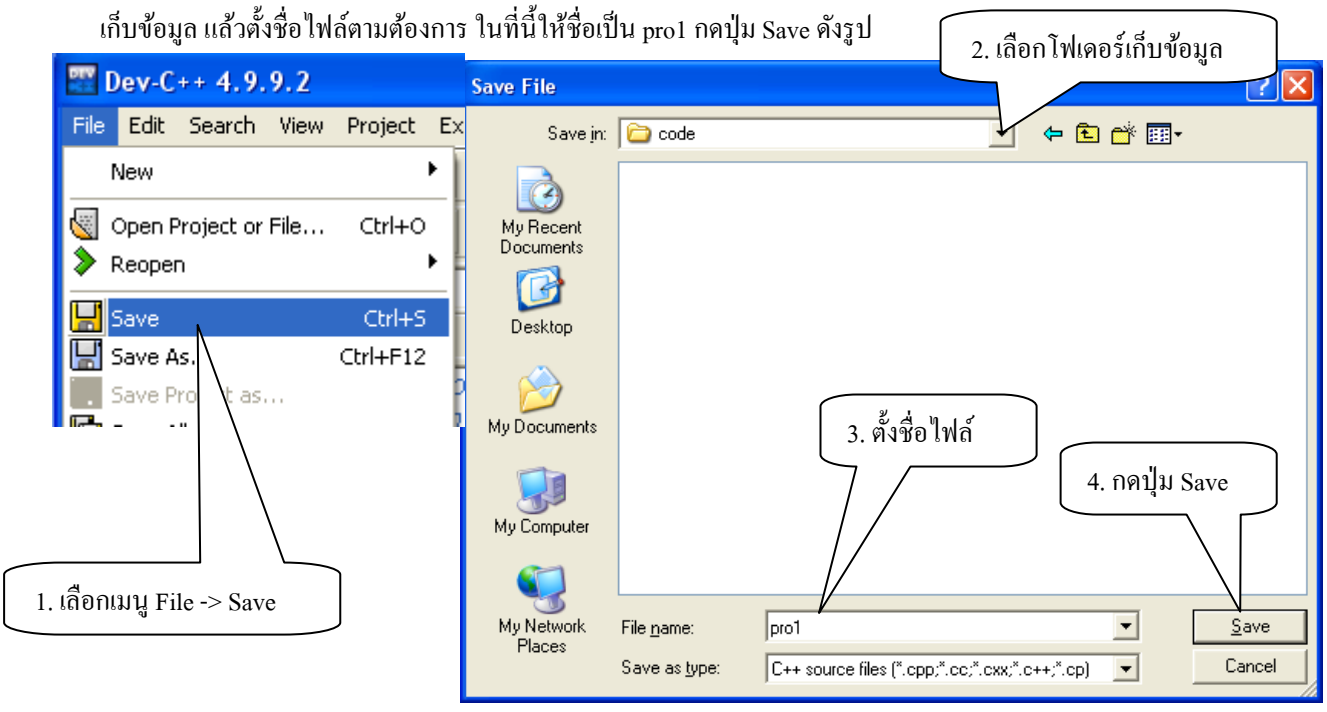
/* PROGRAM BY TEERAWAT PRAKOBPHON */

จากตัวอย่างที่กล่าวมาเมื่อเขียนโปรแกรมด้วย Dev-C++ ทำได้ดังขั้นตอนต่อไปนี้

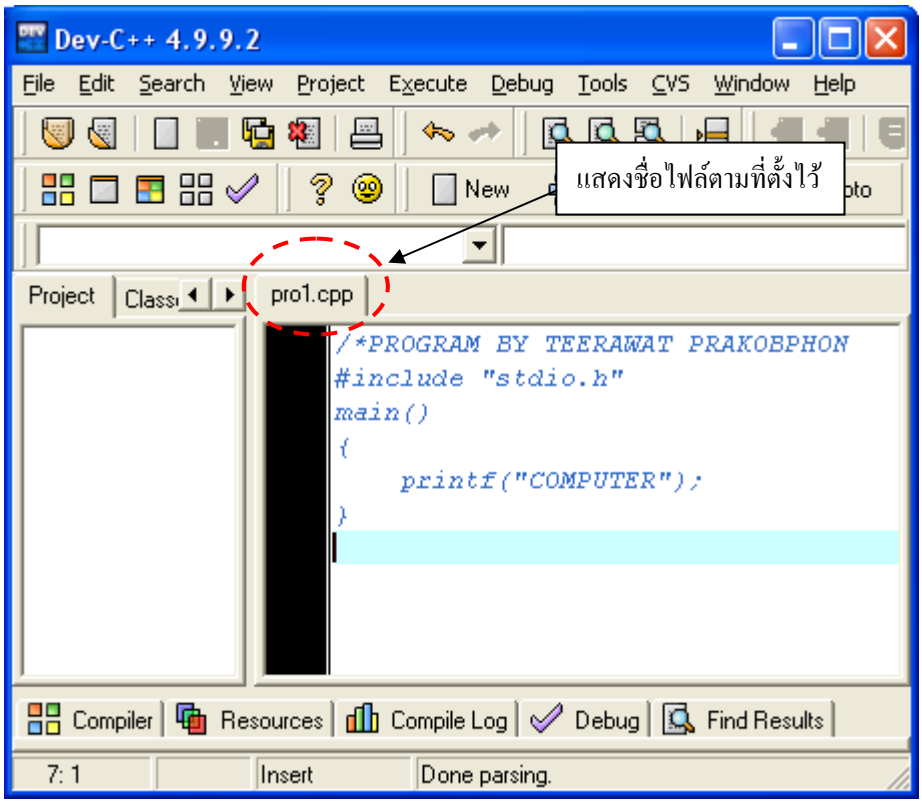
1. เปิดโปรแกรมสร้างไฟล์ใหม่ โดยเลือกเมนู File -> New ->Source File
2. พิมพ์โปรแกรมลงไปดังต่อไปนี้



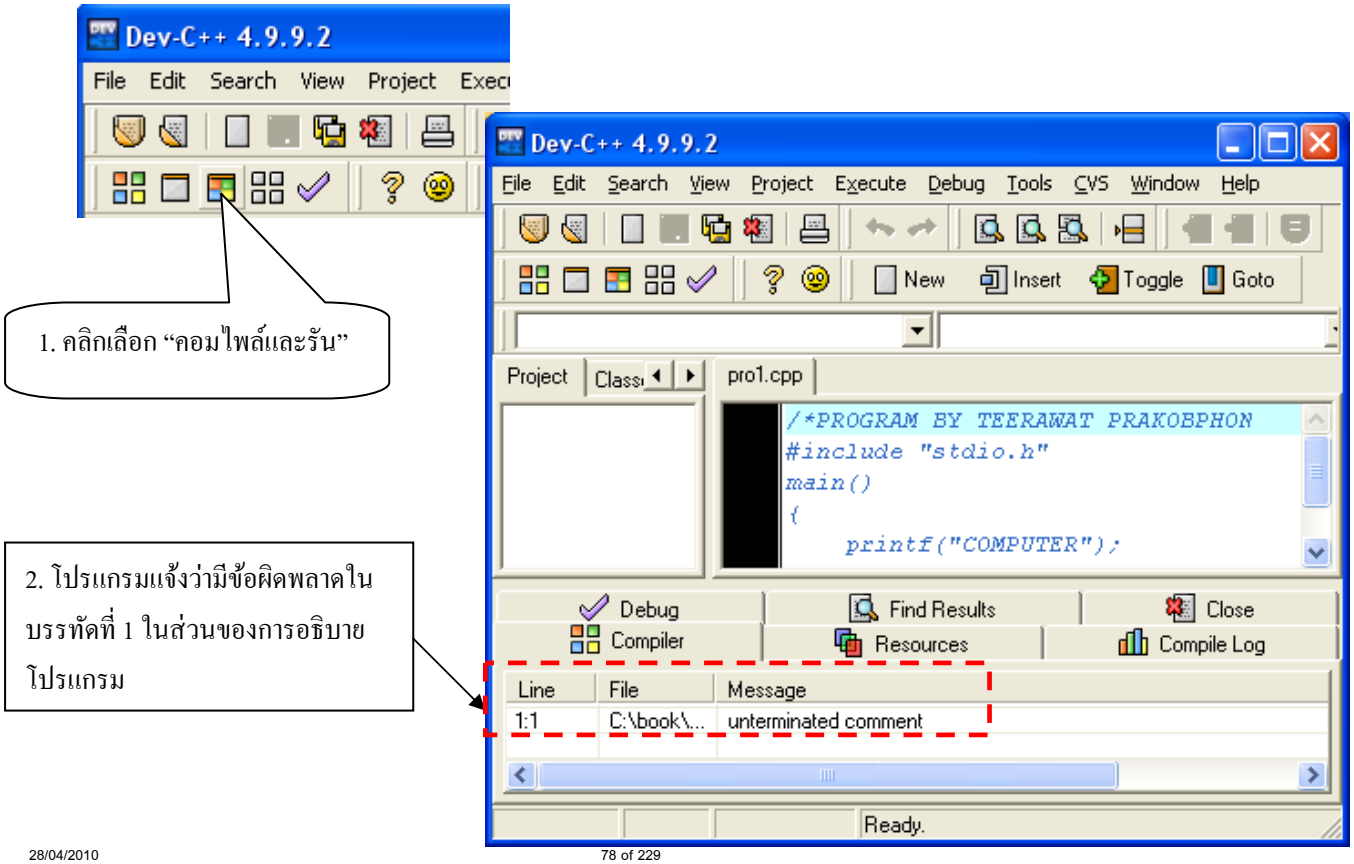
3. จัดเก็บไฟล์ (Save) ไว้ในเครื่องคอมพิวเตอร์ โดยเลือกเมนู File -> Save จากนั้นเลือกโฟเดอร์สำหรับเก็บข้อมูล แล้วตั้งชื่อไฟล์ตามต้องการ ในที่นี้ให้ชื่อเป็น pro1 กดปุ่ม Save ดังรูป



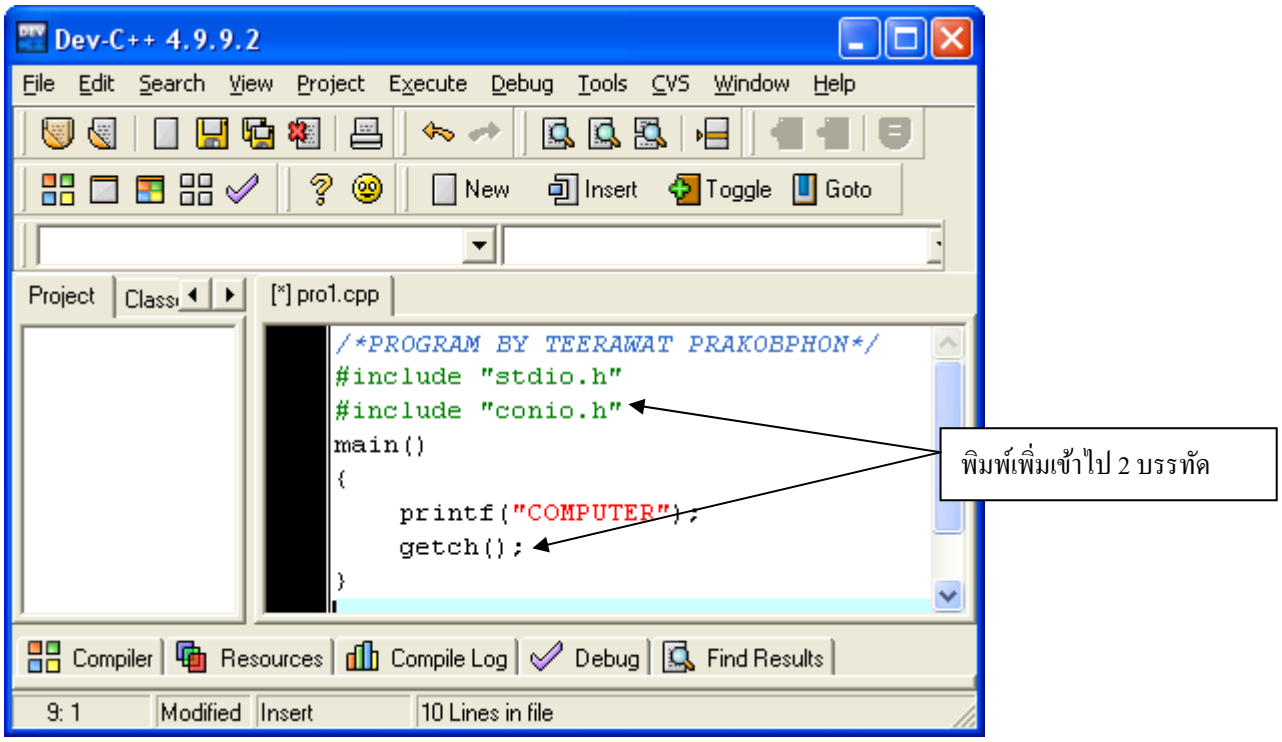
4. โปรแกรมจะบันทึกไฟล์เป็นชื่อ pro1.cpp (เนื่องจากโปรแกรมนี้สามารถเขียนในลักษณะ C++ ได้ จึงมีนามสกุลเป็น cpp)



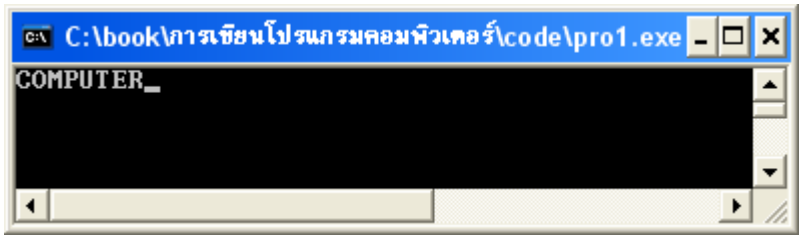
5. คอมไพล์และรันโปรแกรมโดยเลือกเมนู Execute ->Compile & Run หรือคลิกที่ไอคอนดังรูป เมื่อโปรแกรมคอมไพล์จะแจ้งข้อผิดพลาดออกมา เนื่องจากการเขียนคำอธิบายไม่ได้ใส่เครื่องหมาย */ เพื่อปิดคำอธิบาย



ให้แก้ไขโปรแกรมให้ถูกต้อง แล้วคลิก “คอมไพลและรัน” อีกครั้งหนึ่ง โปรแกรมจะรันอย่างรวดเร็ว ไม่สามารถมองได้ทัน ให้ทดลองเพิ่มคำสั่ง getch() ซึ่งเป็นสำหรับรอรับการกดคีย์ แต่คำสั่งนี้ถูกเก็บไว้ใน conio.h ดังนั้นจะต้องเรียก conio.h ขึ้นมาในโปรแกรมด้วย โดยเขียนโปรแกรมดังต่อไปนี้



6. ทดลองคลิกปุ่ม “คอมไพลและรัน” อีกครั้ง จะพบว่าโปรแกรมจะทำฟังก์ชัน printf เป็นฟังก์ชันแรก แล้วแสดงคำว่า “COMPUTER” ออกมา จากนั้นทำฟังก์ชัน getch() เพื่อรอการกดแป้นพิมพ์ ทำให้สามารถดูผลลัพธ์การทำงานได้ทัน โดยโปรแกรมจะแสดงหน้าต่างออกมาดังนี้



จากตัวอย่างที่ผ่านมาจะพบว่าการเขียนโปรแกรมด้วยภาษาซีนั้น ผู้เขียนโปรแกรมจะต้องเขียนโปรแกรมให้ถูกต้องตามโครงสร้างของภาษา ต้องทราบว่าในโปรแกรมต้องใช้ฟังก์ชันหรือคำสั่งใดบ้าง และฟังก์ชันนั้นอยู่ในกลุ่มใด ต้อง include เฮดเดอร์ไฟล์ใดขึ้นมาจึงจะทำงานได้ สำหรับรายละเอียดของฟังก์ชันจะกล่าวต่อไปเมื่อถึงเรื่องที่เกี่ยวข้อง

6.3 ฟังก์ชันพื้นฐานและการประกาศตัวแปร

การเขียนเบื้องต้นด้วยโปรแกรมด้วยภาษาซีนั้นมักจะเป็นการเขียนโปรแกรมรับข้อมูลจากแป้นพิมพ์และแสดงข้อมูลออกทางจอภาพ รวมถึงการประมวลผลทางคณิตศาสตร์อย่างง่ายซึ่งมักจะต้องมีการประกาศตัวแปรสำหรับเก็บข้อมูลด้วย ต่อไปจะกล่าวถึงการรับข้อมูลและการส่งข้อมูลโดยจะกล่าวถึงฟังก์ชันในการรับข้อมูลจากแป้นพิมพ์ และการส่งข้อมูลออกทางจอภาพ โดยคำสั่งดังกล่าวจะอยู่ในไลบรารี stdio.h ซึ่งจะต้องเรียกออกมาก่อนด้วย #include

- **ฟังก์ชัน printf()**

ฟังก์ชันนี้เราได้ทดลองใช้อย่างง่าย ๆ มาแล้ว โดยใช้สำหรับแสดงข้อความหรือตัวแปรฟังก์ชัน printf() มีชื่อเต็มว่า print format เป็นฟังค์ชั่นที่ใช้พิมพ์ข้อความต่าง ๆ ออกทางจอภาพโดยมีรูปแบบดังนี้

```
printf(“ข้อความ”)                                หรือ
printf(“รหัสควบคุมรูปแบบ”,ตัวแปร)                หรือ
printf(“control string”,variable list,……..);
```

โดยค่าที่ส่งเข้าไปในฟังก์ชันจะมีสองส่วน ส่วนแรกเรียกว่าสตริงฟอร์แมต เป็นส่วนที่ใช้ควบคุมลักษณะการแสดงผล และส่วนที่สองหมายถึงค่าคงที่ หรือตัวแปรที่จะให้แสดงผล โดยฟังก์ชันจะพิมพ์ค่าที่อยู่ในตัวแปรในส่วน variable list ในรูปแบบที่กำหนดโดย control string ซึ่งจะศึกษาได้จากตัวอย่างต่อไป

โปรแกรมที่ 6.1

```
/* PROGRAM BY TEERAWAT PRAKOBPHON */
#include “stdio.h”
main()
{
        printf(“COMPUTER\n”);          ข้อความที่จะพิมพ์จะอยู่ในเครื่องหมาย “” ตามด้วย\n
        printf(“\n\nCOMPUTER”);        ข้อความที่จะพิมพ์จะอยู่ในเครื่องหมาย “” โดยมี\n นำหน้า
        printf(“%d\n”,20);
}
```

จากตัวอย่างให้ลอง RUN โปรแกรมดู ซึ่งจะเห็นว่าข้อความที่จะพิมพ์จะอยู่ในเครื่องหมายคำพูด นอกจากนี้ยังมีเครื่องหมายแบ็คสแลชตามด้วย n (\n) ซึ่งเป็นการบอกว่าเมื่อพิมพ์เรียบร้อยแล้วให้ขึ้นบรรทัดใหม่ ส่วนคำสั่งต่อมามี \n นำหน้าสองตัวบอกว่าให้ขึ้นบรรทัดใหม่ก่อนสองครั้ง แล้วค่อยพิมพ์คำว่า COMPUTER เครื่องหมายนี้เรียกว่า รหัสแบ๊ตสแลช (backslash) ซึ่งมีอยู่หลายรูปแบบดังนี้

รหัส	ผลที่ได้
\n	ให้ขึ้นบรรทัดใหม่
\t	ให้เว้น tab เป็นระยะ 8 ช่วง
\xhh	ใส่ตัวอักษร hh เมื่อเป็นเลขฐานสิบหก
\a	ส่งเสียงบีป
\\	เครื่องหมาย Backslash

จากโปรแกรมที่ 6.1 พิจารณาฟังค์ชั่น printf() บรรทัดที่ 3 เครื่องหมาย %d หมายถึงรหัสควบคุมการพิมพ์ (format code) ซึ่งเป็นตัวบอกให้พิมพ์เลขฐานสิบ เครื่องหมาย % นี้เรียกว่า format specification ซึ่งมีใช้หลายรูปแบบดังตาราง

เครื่องหมาย	การใช้งาน
%d	ให้พิมพ์รูปแบบเลขจำนวนเต็มฐานสิบ
%u	ให้พิมพ์รูปแบบเลขจำนวนเต็มไม่มีเครื่องหมาย
%f	ให้พิมพ์รูปแบบเลขทศนิยม
%e	ให้พิมพ์เลขจำนวนจริงในรูปเลขยกกำลัง
%c	ให้พิมพ์ตัวอักษรตัวเดียว (Char)
%s	ให้พิมพ์ชุดตัวอักษร (string) หรือข้อความ
%%	ให้พิมพ์เครื่องหมาย %
%o	ให้พิมพ์เลขฐานแปด
%x	ให้พิมพ์เลขฐานสิบหก

ในการเขียนคำสั่ง printf หนึ่งบรรทัด เราสามารถให้พิมพ์ตัวอักษรหลาย ๆ รูปแบบได้ ดังเช่นคำสั่งต่อไปนี้

```
printf("%s %d %f %c \n","Sam",14,-8.76,'X');
```

จะเห็นว่าคำสั่งนี้มีการพิมพ์หลายรูปแบบและมี variable list หลายตัว ถ้าเป็นสตริงจะอยู่ในเครื่องหมาย " " ถ้าเป็นตัวอักษรจะอยู่ในเครื่องหมาย ' ' สำหรับค่า variable list จะต้องเรียกตามลำดับที่กำหนดใน control string ผลลัพธ์จากการทำคำสั่งจะเป็นดังนี้

```
Sam 14 -8.760000 X
```

จะสังเกตเห็นว่าในการพิมพ์เลขทศนิยมข้อมูลที่ใส่ไปคือ –8.76 แต่ผลลัพธ์ที่ได้จะได้ทศนิยมหลายตำแหน่ง คำสั่ง printf สามารถระบุจำนวนเลขทศนิยมได้ โดยใส่จำนวนทศนิยมและเครื่องหมายจุดระหว่าง % และ f ดังตัวอย่างคำสั่งต่อไปนี้

```
printf("%f %.3f %.2f %.1f",4.5678,4.5678,4.5678,4.5678);
```

ผลลัพธ์ที่ได้จากการทำคำสั่งจะเป็นดังนี้

4.567800 4.568 4.57 4.6

- **ฟังก์ชัน scanf()**

ฟังก์ชันนี้จะตรงข้ามกับฟังก์ชัน printf() โดยจะใช้อ่านค่าจากการกดแป้นพิมพ์ที่อยู่ในรูปรหัส ASCII ไปเก็บในตัวแปรที่กำหนด และสามารถใช้เป็นรหัสควบคุมหรือ controlstring ระบุชนิดของข้อมูลที่จะเก็บในตัวแปรได้ รูปแบบของคำสั่งเป็นดังนี้

```
scanf("control string",&variable list,……..);          หรือ
scanf("รหัสรับข้อมูล",&ตัวแปรเก็บข้อมูล);
```

โดยเครื่องหมาย & เป็นการชี้ไปที่แอดเดรสหน่วยความจำของตัวแปรที่กำหนดเพื่อให้เก็บข้อมูล สำหรับ control string จะเป็นตัวบอกว่าจะให้เก็บข้อมูลในลักษณะใด ตัวอย่างเช่น

```
int  num;                    /*ประกาศตัวแปรชื่อ num สำหรับเก็บเลขจำนวนเต็ม */
scanf("%d",&num);
```

เมื่อคอมพิวเตอร์ทำชุดคำสั่งจะรอรับข้อมูลจากแป้นพิมพ์ เมื่อใส่ข้อมูลและกด Enter จะนำข้อมูลไปเก็บในตัวแปรชื่อ num ในรูปของเลขฐานสิบเนื่องจาก control string เป็น %d สำหรับ control string มีได้หลายตัว เช่นเดียวกับคำสั่ง printf()

พิจารณาชุดคำสั่งต่อไปนี้

```
int  i,j;                              /*ประกาศตัวแปรชื่อ i และ j สำหรับเก็บจำนวนเต็ม */
scanf("%o %x",&i,&j);
printf("%o %x",i,j);
```

เมื่อคอมพิวเตอร์ทำงานจะรับข้อมูลเข้าไป 2 ค่า การใส่ข้อมูลระหว่างค่าแรกและค่าที่สองต้องกดคีย์เว้นวรรคเป็นการแยกค่าของข้อมูล ข้อมูลค่าแรกที่รับเข้าไปจะรับในรูปเลขฐานแปดเนื่องจากใช้รหัสควบคุมเป็น %o ค่าที่สองจะรับในรูปเลขฐานสิบหก คำสั่งต่อมาจะให้คอมพิวเตอร์พิมพ์ข้อมูลที่เก็บอยู่ในรูปเลขฐานแปดและฐานสิบหก พิจารณาชุดคำสั่งต่อไปนี้

```
char  str[80];
printf("Enter a string : ");
scanf("%s",str);                       /*รับสตริงมาเก็บในตัวแปร ไม่ต้องมีเครื่องหมาย & */
printf("Here ' s your string: %s",str);
```

จะเป็นการประกาศตัวแปร str เป็นตัวแปรประเภทสตริงที่เก็บข้อมูลได้ไม่เกิน 80 ตัวอักษร และแสดงข้อความให้พิมพ์สตริง ต่อมาจะรับข้อมูลในรูปสตริงเข้าไปเก็บในตัวแปร str และพิมพ์ออกทางจอภาพ

- **ตัวแปร**

การเก็บข้อมูลของคอมพิวเตอร์นั้นจะเก็บลงในหน่วยความจำส่วนที่เป็น RAM โดยข้อมูลที่เก็บอยู่แต่ละค่าจะอ้างถึงโดยการอ้างไปที่หมายเลขตำแหน่งของหน่วยความจำนั้น สำหรับในการเขียนโปรแกรมจะใช้วิธีการประกาศตัวแปรในการอ้างถึงหน่วยความจำที่ต้องการติดต่อ โดยชื่อของตัวแปรจะเป็นตัวแทนค่าหมายเลขตำแหน่งหน่วยความจำที่ใช้เก็บข้อมูลนั้นเอง เมื่อมีการนำข้อมูลไปเก็บในตัวแปรข้อมูลนั้นจะถูกเปลี่ยนเป็นรหัสเลขฐานสองที่คอมพิวเตอร์เข้าใจได้ ตัวอย่างเช่นถ้าหากต้องการให้ตัวแปรนั้นเก็บเลขจำนวนเต็มคอมพิวเตอร์จะเปลี่ยนเลขจำนวนเต็มเป็นเลขฐานสองที่สอดคล้องกัน ถ้าหากต้องการให้ตัวแปรเก็บตัวอักขระคอมพิวเตอร์จะเปลี่ยนตัวอักขระนั้นเป็นรหัส ASCII หรือรหัส Unicode ตามการประมวลผลของคอมพิวเตอร์นั้น ดังนั้นถ้าหากต้องการให้โปรแกรมรับข้อมูลจากผู้ใช้มาเก็บไว้ หรือมีการคำนวณและเก็บผลลัพธ์จะต้องสร้างตัวแปรสำหรับเก็บข้อมูลที่เป็นผลลัพธ์นั้น พิจารณาโปรแกรมต่อไปนี้

```
#include “stdio.h”
int         feet ,inches;
main()
{
            feet     =  6;
            inches  =  feet * 12;
            printf(“Height in inches is   %d ‘,inches)
}
```

ประกาศตัวแปร

Height in inches is 72

บรรทัดแรกของโปรแกรมจะเป็นอินคลูดไฟล์หรือไฟล์ส่วนหัว ส่วนบรรทัดที่สองจะเป็นการประกาศตัวแปรขึ้นมาสองตัว โดยใช้ int นำหน้าในการประกาศตัวแปร โดยกำหนดให้ตัวแปรชื่อ feet และ inches เป็นตัวแปรประเภท Integer (เก็บเลขจำนวนเต็ม) ในส่วนของโปรแกรมหลักจะเริ่มต้นด้วย main() และสเตตเมนต์ถัดไปจะเป็นการกำหนดค่าให้ feet ให้มีค่าเป็น 6 โดยใช้เครื่องหมายเท่ากับในการกำหนดค่า ส่วนสเตตเมนต์ต่อมาจะนำค่า feet คูณด้วย 12 และเก็บค่าที่ได้ในตัวแปร inches ส่วนสเตตเมนต์ printf จะใช้สำหรับพิมพ์ค่าเอาต์พุตทางจอภาพ ในคำสั่ง printf จะเห็นว่ามีการแสดงผลสองส่วนคือส่วนที่เป็นข้อความ และส่วนที่เป็นตัวแปร โดยใช้เครื่องหมายคอมมา(Comma) คั่น เมื่อโปรแกรมทำงานจะนำข้อมูลที่เป็นตัวแปรไปแสดงผลในตำแหน่งที่เขียนเป็น %d ซึ่งเป็นตัวบอกว่าให้แสดงผลตัวแปรเป็นเลขฐานสิบ

ในส่วนของการประกาศตัวแปรชื่อของตัวแปรจะต้องเป็นไปตามกฎการตั้งชื่อซึ่งจะได้กล่าวต่อไป ถ้าหากชื่อตัวแปรมีความยาวมากกว่า 63 ตัวอักษร โปรแกรมจะรับรู้เพียง 63 ตัวแรกเท่านั้น และในการประกาศตัวแปรถ้าหากมีตัวแปรมากกว่า 1 ตัวจะใช้เครื่องหมาย , ขั้น การกำหนดตัวแปรจะต้องเริ่มต้นด้วยประเภทของข้อมูลตามด้วยชื่อตัวแปร และสามารถประกาศหลาย ๆ ตัวพร้อมกันได้โดยใช้เครื่องหมายคอมมา (,) คั่น

รูปแบบ

ประเภทของข้อมูล	ชื่อตัวแปร, …..; หรือ
Data Type	Variable Name, …. ;

ตัวอย่าง

```
int     feet , inches;
```

ประเภทของข้อมูล — ชื่อตัวแปร

เป็นการบอกว่ามีตัวแปรสองตัว และเป็นตัวแปรประเภท Integer ดังนั้นตัวแปรทั้งสองตัวนี้จะใส่เลขจำนวนเต็มใด ๆ ก็ได้ แต่ต้องอยู่ในช่วง -32767 ถึง +32767 (ขึ้นอยู่กับชนิดของคอมไพเลอร์)

ในการประกาศตัวแปรนั้นมีกฎการตั้งชื่อดังต่อไปนี้

- ชื่อตัวแปรประกอบด้วยตัวอักษร A ถึง Z รวมถึงตัวเลข 0 ถึง 9
- ชื่อตัวแปรต้องขึ้นต้นด้วยตัวอักษร
- ตัวพิมพ์ใหญ่ ตัวพิมพ์เล็กมีความหมายต่างกัน
- ต้องไม่มีเว้นวรรคระหว่างชื่อตัวแปร
- ชื่อต้องไม่ซ้ำกับคำสงวน (Reserved Word) ที่โปรแกรมรู้จัก

นอกจากนี้ในภาษาซียังประกาศตัวแปรพร้อมกับกำหนดค่าให้กับตัวแปรได้ โดยจะใช้เครื่องหมาย = ตามด้วยค่าที่จะให้กับตัวแปร รูปแบบทั่วไปของการกำหนดค่าให้กับตัวแปรจะเป็นดังนี้

```
variable = expression;
```

โดยที่ expression อาจเป็นค่าคงที่ ตัวแปร หรือการกระทำทางคณิตศาสตร์ก็ได้ การทำงานของโปรแกรมจะนำค่าที่อยู่ทางขวาของเครื่องหมาย assignment (=) มาใส่ให้กับตัวแปรที่อยู่ทางซ้าย ในการเขียนโปรแกรมด้วยภาษาซี สามารถใช้เครื่องหมายการกระทำทางคณิตศาสตร์บวก(+) และ (-) ได้ ส่วนการคูณจะใช้เครื่องหมาย * และการหารจะใช้เครื่องหมายสองตัวคือ div และ / ส่วนรายละเอียดจะได้กล่าวต่อไป

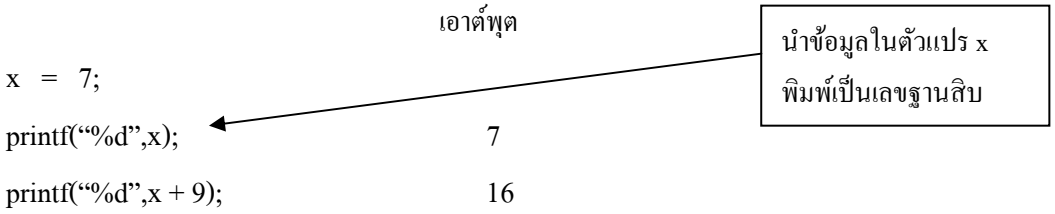
ตัวอย่าง

```
int  x;          /*ประกาศตัวแปรชื่อ x สำหรับเก็บเลขจำนวนเต็ม */
x  =  5;         /*กำหนดค่า 5 ให้กับตัวแปร x  */
```

ตัวอย่าง

```
int x, y;        /*ประกาศตัวแปรชื่อ x และ y เก็บเลขจำนวนเต็ม */
y  =  3;         /*กำหนดให้ y มีค่าเท่ากับ 3 */
x  =  y + 5;     /*นำค่า y บวกกับ 5 แล้วนำไปเก็บในตัวแปร x */
```

ในการนำคำสั่ง printf มาใช้แสดงผลเอาต์พุตจะให้เอาต์พุตเป็นตัวแปร หรือการกระทำทางคณิตศาสตร์ก็ได้ แต่ต้องใช้เครื่องหมาย % ในการกำหนดประเภทของข้อมูล ตัวอย่างเช่น



คำถาม หลังจากคอมพิวเตอร์ทำคำสั่งต่อไปนี้ เอาต์พุตที่ได้จะเป็นอย่างไร

```
yards  =  8;
feet  =  yards * 3;
printf("%d yards is ", yeads);
printf("%d feet" , feet);
```

```
yards  =  8;
feet  =  yards * 3;
printf("%d  yards is \n",yeads);
printf("%d  feet \n", feet);
```

คำตอบ

8 yards is24 feet

8 yards is
24 feet

คำถาม ถ้าหากคอมพิวเตอร์ทำคำสั่งต่อไปนี้ เอาต์พุตที่ได้จะเป็นอย่างไร

```
printf("Yes \n");
printf("No \n");
```

```
printf("Yes \n");
printf("\n");
printf("No \n");
```

```
printf("Yes");
printf("No");
```

คำตอบ

Yes Yes YesNo

No

No

คำถาม เอาต์พุตทางจอภาพจะเป็นอย่างไร ถ้าหากคอมพิวเตอร์ทำโปรแกรมต่อไปนี้

```
#include “stdio.h”
int         nickels,dimes,TotCenter;
main()
{
            nickels  =  3;
            dimes    =  7;
            TotCenter  =  (nickels * 5) + (dimes * 10);
            printf(“%d  nickels  and  %d  dimes \n”,nickels,dimes);
            printf(“=  %d  cents \n”,TotCenter);
}
```

คำตอบ

3 nickels and 7 dimes

= 85 cents

ชนิดของข้อมูล

ในการเขียนโปรแกรมเมื่อมีการประกาศตัวแปรคอมพิวเตอร์จะจองหน่วยความจำให้กับตัวแปรนั้น ถ้าหากตัวแปรเก็บข้อมูลต่างประเภทกันหน่วยความจำที่ใช้สำหรับเก็บข้อมูลก็จะไม่เท่ากันด้วย สำหรับข้อมูลในภาษาซีอาจแบ่งออกได้เป็น 4 กลุ่มดังต่อไปนี้

- ข้อมูลชนิดซิมเปิล (simple type)
- ข้อมูลประเภทสตริง (string type)
- ข้อมูลประเภทโครงสร้าง (structure type)
- ข้อมูลประเภทพอยต์เตอร์ (pointer type)

ในหัวข้อนี้จะกล่าวถึงข้อมูลชนิดซิมเปิลและข้อมูลชนิดสตริงก่อน ส่วนข้อมูลประเภทอื่น ๆ จะกล่าวต่อไปในภายหลัง

ข้อมูลชนิดซิมเปิล

ข้อมูลชนิดซิมเปิลแบ่งได้เป็นข้อมูลประเภทลำดับ (ordinal type) และข้อมูลประเภทจำนวนจริง (Real Data Type) โดยข้อมูลแบบลำดับเป็นข้อมูลที่มีค่าเป็นลำดับแน่นอน เช่นตัวเลขที่ใช้ในการนับ ลำดับตัวอักษรเป็นต้น ในภาษาซียังแบ่งข้อมูลชนิดลำดับออกได้หลายประเภท ในที่นี้จะกล่าวถึงข้อมูลประเภทจำนวนเต็ม ข้อมูลอักขระ และข้อมูลตรรกะ

- **ข้อมูลชนิดจำนวนเต็ม (Integer Data Type)**

ข้อมูลประเภทนี้จะใช้เก็บตัวเลขที่เป็นจำนวนเต็ม ในคอมพิวเตอร์จะใช้หน่วยความจำในการเก็บข้อมูล ถ้าหากคอมพิวเตอร์ใช้หน่วยความจำ 8 บิตหรือ 1 ไบต์ในการเก็บข้อมูลจะทำให้เก็บข้อมูลที่เป็นเลขฐานสิบได้ในช่วง 0 ถึง 255 ข้อมูลชนิดจำนวนเต็มนี้ยังแบ่งได้หลายประเภทขึ้นกับขนาดของหน่วยความจำที่คอมพิวเตอร์ใช้เก็บ โดยข้อมูลประเภทต่าง ๆ แสดงได้ดังตารางต่อไปนี้

ประเภท	ช่วงของข้อมูลที่เก็บได้	ขนาดหน่วยความจำ
char	-128 – 127	1 ไบต์
unsigned char	0 – 255	1 ไบต์
signed char	-128 .. 127	1 ไบต์
int	-2,147,483,648 .. 2,147,483,647	4 ไบต์
unsigned int	0 .. 4,294,967,296	4 ไบต์

ตารางแสดงประเภทของข้อมูลและช่วงของข้อมูลที่เก็บได้

จากตารางจะพบว่าการประกาศตัวแปรให้เก็บเลขจำนวนเต็มจะประกาศได้หลายประเภท แต่ละประเภทจะใช้หน่วยความจำไม่เท่ากัน ทำให้เก็บข้อมูลได้ในช่วงที่ต่างกันไปด้วย แต่ขนาดของหน่วยความจำที่ใช้กับข้อมูลแต่ละประเภทนั้นจะขึ้นกับชนิดของคอมไพเลอร์ที่ใช้ด้วย ตัวอย่างเช่น ถ้าหากประกาศตัวแปรเป็นข้อมูลชนิด int แล้วใช้คอมไพเลอร์ Turbo-C จะใช้หน่วยความจำเพียง 2 ไบต์ ทำให้เก็บข้อมูลได้ในช่วง -32768 ถึง +32767 แต่ถ้าหากเป็นคอมไพเลอร์ตัวอื่น เช่น DEV-C จะใช้หน่วยความจำ 4 ไบต์ ทำให้เก็บข้อมูลได้ในช่วงที่กว้างกว่า ในการประกาศตัวแปรนี้ถ้าหากประกาศตัวแปรที่เก็บข้อมูลได้มากเกินความจำเป็นจะทำให้โปรแกรมที่เขียนขึ้นใช้หน่วยความจำมากขึ้นตามไปด้วย ตัวอย่างเช่นถ้าหากแน่ใจว่าข้อมูลตัวเลขที่เกิดขึ้นมีค่าไม่เกิน 127 ก็ควรประการตัวแปรเป็นชนิด char ก็เพียงพอ

- **ข้อมูลประเภทตัวอักขระ (Character Data Type)**

ข้อมูลประเภทนี้จะเป็นตัวอักขระหนึ่งตัว ซึ่งเป็นไปตามตารางรหัส ASCII ประกอบด้วยข้อมูลที่เป็นตัวอักษร ตัวเลข และอักขระพิเศษ ข้อมูลประเภทนี้จะเป็นข้อมูลแบบลำดับได้ เนื่องจากเรียงตามลำดับรหัส ASCII ข้อมูลประเภทนี้จะใช้เนื้อที่ในการเก็บหนึ่งไบต์เช่น

'A','B','C' นอกจากนี้ยังมีอักขระพิเศษเช่น

'\n' รหัสขึ้นบรรทัดใหม่

'\t' รหัสเว้นวรรค 1 tab

'\a' เสียง Beep

- **ข้อมูลประเภทตรรก (Boolean Data Type)**

จะเป็นค่าทางลอจิก ได้แก่จริง(True) กับเท็จ (False) จะใช้ในคำสั่งควบคุมเพื่อตัดสินใจการทำงาน ในการเรียงลำดับจะให้ค่าที่เป็นเท็จมีลำดับก่อนค่าที่เป็นจริง ในการเขียนโปรแกรมบางครั้งจะแทนค่าจริงด้วยเลขจำนวนเต็ม 1หรือค่าที่มากกว่า 1 และแทนค่าเท็จด้วยเลข 0

- **ข้อมูลประเภทจำนวนจริง (Real Data Type)**

ข้อมูลประเภทนี้จะเป็นจำนวนจริงหรือเลขทศนิยม ข้อมูลประเภทนี้จะจัดลำดับก่อนหลังได้ยาก จึงไม่เป็นข้อมูลชนิดลำดับเนื่องจากทศนิยมมีได้หลายตำแหน่ง ข้อมูลจำนวนจริงนี้ยังแบ่งออกได้หลายประเภท โดยแต่ละประเภทจะใช้หน่วยความจำในการเก็บแตกต่างกัน ทำให้เก็บข้อมูลได้แตกต่างกัน ดังตารางต่อไปนี้

ประเภท	ช่วงของข้อมูลที่เก็บได้	ขนาดหน่วยความจำ
float	3.4×10^{-38} ถึง 3.4×10^{38}	4 ไบต์
double	1.7×10^{-308} ..ถึง $1.7 \times 10^{+308}$	8 ไบต์
long double	3.4×10^{-4032} .ถึง. 1.1×10^{4032}	10 ไบต์

ตารางแสดงประเภทของข้อมูลชนิดจำนวนจริง

- **ข้อมูลประเภทสตริง (string type)**

นอกจากข้อมูลแบบตัวเลขและตัวอักษรแล้วในภาษาซียังมีข้อมูลอีกประเภทหนึ่งเรียกว่าสตริง ข้อมูลประเภทนี้จะเป็นการนำตัวอักขระมาต่อเรียงกับเป็นข้อความตั้งแต่หนึ่งตัวขึ้นไป โดยสามารถเก็บตัวอักขระได้ 255 ตัว โดยตัวอักขระจะต้องอยู่ในเครื่องหมาย " " ในการเขียนโปรแกรมด้วยภาษาซีจะมีการเติมตัวอักษรว่าง NULL (\0) เป็นตัวสุดท้าย อย่างเช่นการเก็บสตริงคำว่า "COMPUTER" จะใช้เนื้อที่ในการเก็บ 9 ไบต์ โดยแต่ละไบต์เป็นดังนี้

"C"	"O"	"M"	"P"	"U"	"T"	"E"	"R"	"\0"

การเก็บข้อมูลของตัวแปร

ในการประกาศตัวแปรให้กับโปรแกรมนั้น เมื่อโปรแกรมถูกรันตัวแปรต่าง ๆ จะเป็นตำแหน่งหน่วยความจำที่ใช้เก็บข้อมูล ตัวอย่างเช่นในโปรแกรมที่ผ่านมา จะใช้หน่วยความจำสามตำแหน่ง หลังจากโปรแกรมทำงานค่าในหน่วยความจำตำแหน่งต่าง ๆ จะเป็น

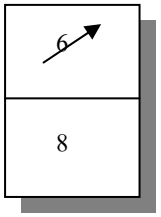
nickels	Dimes	TotCentr
3	7	85

เมื่อโปรแกรมมีการทำงานใด ๆ ค่าในหน่วยความจำหรือตัวแปรอาจมีการเปลี่ยนแปลงได้ โดยค่าเก่าจะหายไปและถูกแทนด้วยค่าใหม่ พิจารณาโปรแกรมต่อไปนี้

```
#include “stdio.h”
int       x;
main()
{
          x = 6;
          x = 8;
          printf(“X is %d”,x);
}
```

หน่วยความจำ

x

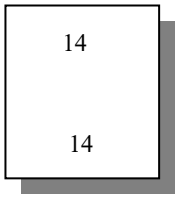


เมื่อโปรแกรมมีการรัน เริ่มแรกค่าในตัวแปร x จะมีค่าเป็น 6 ต่อมามีการใส่ค่า 8 ลงในตัวแปร x ทำให้ค่าใน x เป็น 8 เมื่อโปรแกรมรันไปถึงคำสั่ง printf จะทำให้พิมพ์เลข 8 ออกทางหน้าจอ

ในการเขียนโปรแกรมภาษาซี เราสามารถนำค่าในตัวแปรใส่ให้กับตัวแปรได้ โดยใช้เครื่องหมายเท่ากับ (=) เช่นเดียวกับการกำหนดค่าให้ตัวแปร พิจารณาโปรแกรมตัวอย่างต่อไปนี้

```
num1    =    5;
num2    =    14;
num1    =    num2;
printf(“%d”,num1);
printf(“%d”,num2);
```

เอาต์พุต



เมื่อโปรแกรมทำงาน เริ่มแรกค่าในหน่วยความจำ num1 จะเป็น 5 และค่าใน num2 เป็น 14 ต่อมามีการใส่ค่าในตัวแปร num1 ด้วยค่า num2 ซึ่งจะทำให้ค่าในตัวแปร num1 เปลี่ยนไป เอาต์พุตจากการรันโปรแกรมจะได้เลข 14 จำนวนสองตัวทางจอภาพ

ในการกำหนดค่าให้กับตัวแปรนั้นเราสามารถนำค่าตัวแปรเดิมมากระทำทางคณิตศาสตร์ได้ โดยที่ตัวแปรทางซ้ายเปลี่ยนไปตามการกระทำทางขวามือ ดังตัวอย่างต่อไปนี้

ก. count = count +1 {เพิ่มค่าในตัวแปร count ขึ้นหนึ่ง}

ข. sum = sum + x {นำค่าใน sum บวกกับ x โดยค่าใน x จะไม่เปลี่ยน}

ค. num = 3 * num {นำค่าใน num คูณกับ 3 และเก็บไว้ที่เดิม}

6.4 นิพจน์และตัวดำเนินการทางคณิตศาสตร์

ในการเขียนโปรแกรมให้คอมพิวเตอร์ประมวลผลนั้นสามารถนั้นเอาค่าคงที่ ตัวแปร ตัวดำเนินการมาประกอบกันเป็นนิพจน์ได้ โดยในนิพจน์หนึ่ง ๆ สามารถนำตัวดำเนินมากกว่าหนึ่งตัวมาใช้ได้ ตัวอย่างเช่น

```
int  y;                    /*ประกาศตัวแปร y สำหรับเก็บเลขจำนวนเต็ม */
y  =  5 * 3 + 2*2;         /*สร้างนิพจน์แล้วกำหนดค่าให้กับตัวแปร y */
printf("Output  =  %d ",y);
```

จากตัวอย่างผลลัพธ์ที่ได้จะเป็น Output = 19 โดยการประมวลผลลักษณะนี้ผู้เขียนโปรแกรมจะต้องเข้าใจลำดับก่อนหลังของการกระทำของตัวดำเนินการด้วยว่าถ้าหากมีตัวดำเนินการหลาย ๆ ตัวประกอบอยู่ คอมพิวเตอร์จะประมวลผลอย่างไร จากตัวอย่างโปรแกรมจะนำ 5 คูณกับ 3 ก่อน จากนั้นนำค่า 2 คูณกับ 2 แล้วนำผลลัพธ์ทั้งสองมารวมกันแล้วเก็บไว้ในตัวแปร y สำหรับตัวดำเนินการทางคณิตศาสตร์แสดงได้ดังตารางต่อไปนี้

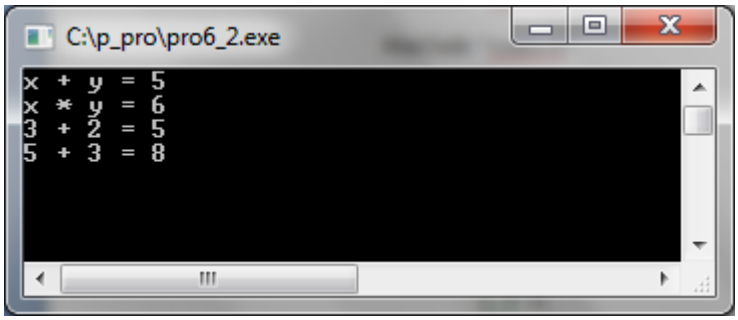
ตัวดำเนินการ	ความหมาย	ตัวอย่าง
+	บวก (Addition)	y = x + 5;
-	ลบ (Subtraction)	y = x – 2;
*	คูณ (Multiplication)	y = x * 3;
/	หาร (Division)	y = x / 2;
%	การหารแบบเอาเศษ (Modulus)	y = 5 % 3;
++	เพิ่มค่าขึ้นหนึ่งค่า (Increment)	y++;
--	ลดค่าลงหนึ่งค่า (Decrement)	y--;

ตัวดำเนินการทางคณิตศาสตร์

โปรแกรมที่ 6.2 โปรแกรมแสดงการใช้ตัวดำเนินการ

```
#include "stdio.h"
#include "conio.h"
main()
{
        int x,y;
        x  =  3;
        y  =  2;
        printf("x + y = %d\n",x+y);
        printf("x * y = %d\n",x*y);
        printf("%d + %d = %d\n",x,y,x+y);
        printf("5 + 3 = %d\n",5 + 3);
        getch();
}
```

เมื่อโปรแกรมทำงานเริ่มต้นจะประกาศตัวแปรชื่อ x และ y สำหรับเก็บเลขจำนวนเต็มจากนั้นกำหนดให้ x มีค่าเป็น 3 และ y มีค่าเป็น 2 เมื่อมาถึงฟังก์ชัน printf() โปรแกรมจะนำค่าอาร์กิวเมนต์มาประมวลผลและนำตัวดำเนินการ บวก มาใช้ ผลลัพธ์ที่ได้จะถูกแสดงผลทางรหัสกำหนดรูปแบบ %d ส่วนฟังก์ชัน printf() ฟังก์ชันที่สามจะมีรหัสกำหนดรูปแบบสามตัว ดังนั้นจะต้องมีอาร์กิวเมนต์สามตัวด้วย เมื่อคีย์โปรแกรมและรันด้วย DEV-C ผลลัพธ์ที่ได้จะเป็นดังนี้



ในการเขียนโปรแกรมที่มีการใช้ตัวดำเนินการหลาย ๆ ตัวนั้นถ้าหากป้องกันผลลัพธ์ที่คลาดเคลื่อนจากการกระทำก่อนหลังของตัวดำเนินการมักจะนำวงเล็บมาใช้ สำหรับลำดับการประมวลผลของนิพจน์จะเป็นดังตารางต่อไปนี้

ลำดับที่	ตัวดำเนินการ	ทิศทางการประมวลผล
1	( )	ทำในวงเล็บมีลำดับความสำคัญสูงสุด
2	++ , --	เรียงจากขวาไปซ้าย
3	*, / , %	เรียงจากซ้ายไปขวา
4	+ , -	เรียงจากซ้ายไปขวา
5	=	นำข้อมูลทางขวาไปใส่ทางซ้าย

ตารางแสดงลำดับการทำงานของตัวดำเนินการ

ตัวอย่าง จงหาค่าจากการประมวลผลต่อไปนี้

y = 7 + 8 * 2

วิธีคิด นำ 8 คูณ 2 ได้ 16 แล้วบวกกับ 7 ทำให้ y มีค่าเป็น 23

ตัวอย่าง จงหาค่าจากการประมวลผลต่อไปนี้

y = 5%2 + 14/3 – 6

วิธีคิด

1. 5 % 2 = 1
2. 14/3 = 4.66
3. 1 + 4.66 = 5.66
4. 5.66 – 6 = -0.33

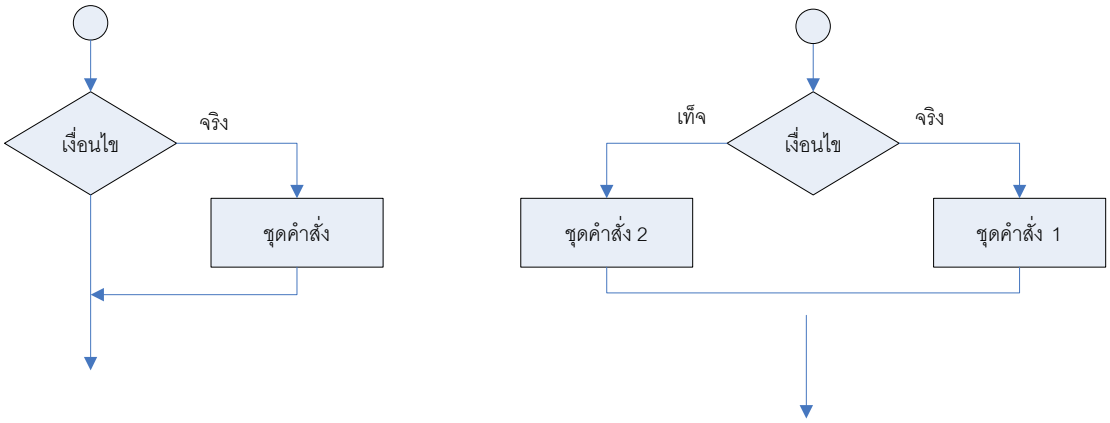
ดังนั้น y มีค่าเท่ากับ -0.33

6.5 คำสั่งกำหนดเงื่อนไข

การเขียนโปรแกรมคอมพิวเตอร์นั้นเป็นการนำคำสั่งต่าง ๆ มาต่อเรียงกันให้เครื่องทำงานต่อเนื่องกันไป สำหรับโปรแกรมที่มีความซับซ้อนมากขึ้นจะต้องมีการเปลี่ยนทิศทางการทำงานของโปรแกรมบ้าง ซึ่งจะต้องให้โปรแกรมตัดสินใจด้วยว่าจะเลือกทำชุดคำสั่งที่ตามมาหรือไม่ หรืออาจต้องเลือกทำชุดคำสั่งใดชุดคำสั่งหนึ่ง

การทำงานแบบเลือกทำนั้นสามารถแบ่งทิศทางการทำงานของโปรแกรมได้ดังนี้

1. การทำงานแบบมีทางเลือกเดียว (Single Selection)
2. การทำงานแบบสองทางเลือก (Double Selection)



ก. การเลือกทำแบบทางเลือกเดียว ข. การเลือกทำแบบสองทางเลือก

รูปแสดงการทำงานแบบมีทางเลือก

สำหรับการทำงานแบบมีทางเลือกเดียวและแบบสองทางเลือกจะต้องมีการตรวจสอบเงื่อนไขเพื่อเป็นตัวพิจารณาว่าจะทำโปรแกรมในทิศทางใดต่อไป การตรวจสอบเงื่อนไขนั้นผลลัพธ์ที่ได้จะเป็นค่าทางบูลีน คือมีค่าเป็นไปได้ 2 กรณีคือ เป็นจริง (true) หรือเป็นเท็จ (false)

กรณีการเลือกทำแบบทางเลือกเดียว ถ้าหากเงื่อนไขเป็นจริง โปรแกรมจะทำชุดคำสั่งที่กำหนด แต่ถ้าหากเงื่อนไขเป็นเท็จโปรแกรมจะไม่ทำชุดคำสั่ง ส่วนการเลือกทำแบบสองทางเลือก(รูป ข) ถ้าหากเงื่อนไขเป็นจริง โปรแกรมจะทำชุดคำสั่งที่ 1 แต่ถ้าหากเงื่อนไขเป็นเท็จโปรแกรมจะทำชุดคำสั่งที่ 2

ในส่วนของการตรวจสอบเงื่อนไขนั้นจะเป็นการเขียนนิพจน์ที่แสดงความสัมพันธ์ระหว่างตัวแปรกับตัวแปร หรือระหว่างตัวแปรกับค่าคงที่ โดยจะนำตัวดำเนินการเปรียบเทียบและตัวดำเนินการทางตรรกะมาใช้

ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการเปรียบเทียบ (Relation Operators) จะนำข้อมูลสองค่ามาเปรียบเทียบกัน โดยข้อมูลทั้งสองค่าจะต้องเป็นข้อมูลประเภทเดียวกัน ผลลัพธ์ที่ได้จะเป็นค่าทางลอจิกคือจริงหรือเท็จ ตัวดำเนินการเปรียบเทียบในภาษาซีเป็นดังตารางต่อไปนี้

ตัวดำเนินการ	ความหมาย	ตัวอย่าง
==	เท่ากับ (Equal)	6 == 3 เป็นเท็จ
!=	ไม่เท่ากับ (Not Equal)	6 != 3 เป็นจริง
<=	น้อยกว่าหรือเท่ากับ (Less Than or Equal)	6 <= 3 เป็นเท็จ
>=	มากกว่าหรือเท่ากับ (Greater Than or Equal)	6 >= 3 เป็นจริง
>	มากกว่า (Greater Than)	6 > 3 เป็นจริง
<	น้อยกว่า (Less Than)	6 < 3 เป็นเท็จ

ตารางแสดงตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการทางตรรกะ

ตัวดำเนินการทางตรรกะ(Logical Operator) เป็นตัวดำเนินการที่เชื่อมระหว่างเงื่อนไขสองเงื่อนไขหรือมากกว่า ผลลัพธ์ที่ออกมาจะเป็นจริงหรือเท็จ ตัวดำเนินการทางตรรกะแสดงได้ดังตารางต่อไปนี้

ตัวดำเนินการ	ความหมาย	การกระทำ
&&	และ (AND)	ผลลัพธ์จะเป็นจริงเมื่อเงื่อนไขสองเงื่อนไขเป็นจริง
||	หรือ (OR)	ผลลัพธ์จะเป็นเท็จเมื่อเงื่อนไขสองเงื่อนไขเป็นเท็จ
!	ไม่ใช่ (NOT)	เปลี่ยนค่าจากจริงเป็นเท็จ จากเท็จเป็นจริง

ตารางแสดงตัวดำเนินการทางตรรกะ

ตัวอย่าง

การกระทำต่อไปนี้ให้ผลลัพธ์เป็นจริงหรือเท็จ

(5 == 4 + 1) && (18 <= 6 * 4)

เนื่องจากประโยคในวงเล็บทั้งสองวงเล็บเป็นจริง ดังนั้นผลลัพธ์ที่ได้จะเป็นจริง

การเขียนโปรแกรมด้วยภาษาซีนั้นการตรวจสอบเงื่อนไขจะใช้คำสั่ง if แล้วตามด้วยเงื่อนไข ถ้าหากเงื่อนไขเป็นจริงโปรแกรมจะทำสเตตเมนต์ที่ตามมา แต่ถ้าหากเป็นเท็จจะไม่ทำ ถ้าหากสเตตเมนต์มีคำสั่งหลายคำสั่งจะเขียนอยู่ในเครื่องหมายปีกกา รูปแบบของคำสั่งเป็นดังนี้

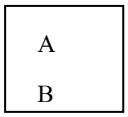
```
if(การตรวจสอบเงื่อนไข)
{
        คำสั่งหรือสเตตเมนต์
}
```

ตัวอย่าง ถ้าหากคอมพิวเตอร์ทำคำสั่งต่อไปนี้

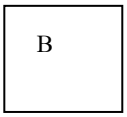
```
if (x >= 10)    printf("A\n");
        printf("B");
```

ถ้าหากค่าในตัวแปร x เป็นค่าต่าง ๆ จะทำให้คอมพิวเตอร์แสดงผลลัพธ์ดังต่อไปนี้

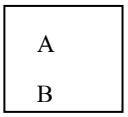
x = 12



x = 8



x = 15



ในการตรวจสอบเงื่อนไขถ้าหากมีเงื่อนไขมากกว่าหนึ่งเงื่อนไขเราสามารถนำตัวดำเนินการทางตรรกะมาใช้ได้ ตัวอย่างเช่นถ้าหากตัวแปร sc เก็บค่าคะแนน และต้องการตรวจสอบว่าถ้าคะแนนที่ได้มีค่ามากกว่า 80 และน้อยกว่าหรือเท่ากับ 100 ให้ได้เกรด A จะเขียนคำสั่ง if ได้ดังนี้

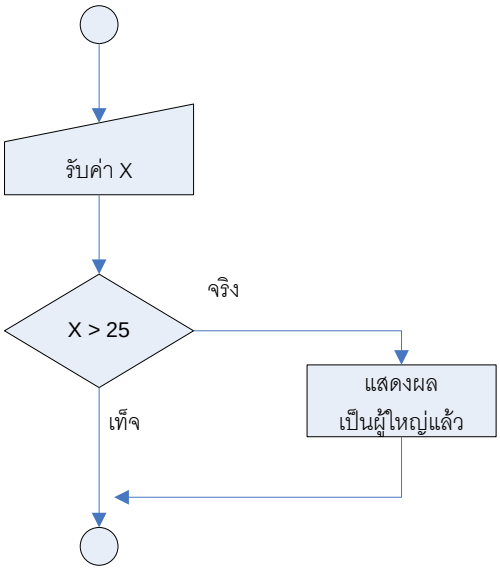
```
if((sc > 80) && (sc <= 100))
        printf("A");
```

สำหรับการเลือกทำแบบสองทางเลือกจะใช้คำสั่ง if-else ถ้าหากเงื่อนไขเป็นจริงจะทำนิพจน์หลัง if แต่ถ้าเงื่อนไขเป็นเท็จจะทำนิพจน์หลัง else โดยรูปแบบของคำสั่งเป็นดังนี้

```
if(ตรวจสอบเงื่อนไข)
{
        สเตตเมนต์ที่ 1;
}
else
{
        สเตตเมนต์ที่ 2;
}
```

ตัวอย่างที่ จงเขียนผังงานและรหัสเทียมสำหรับโปรแกรมตรวจสอบอายุ ถ้าหากอายุมากกว่า 25 ปี ให้แสดงคำว่า "เป็นผู้ใหญ่แล้ว" แต่ถ้าอายุยังไม่เกิน 25 ให้ทำงานต่อไป

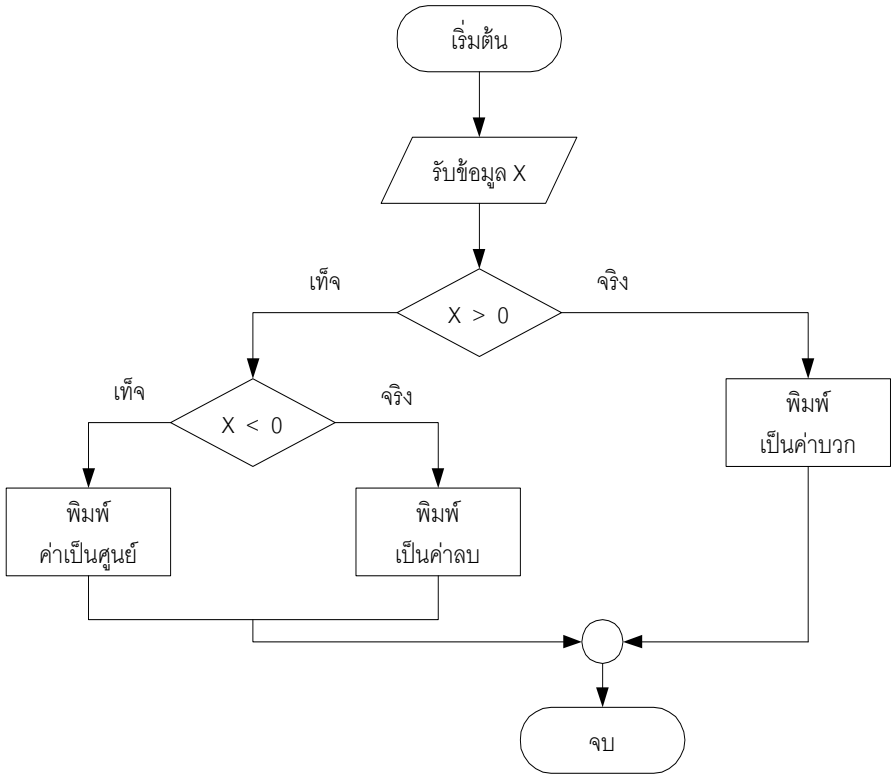
วิธีทำ กำหนดให้ตัวแปร x เป็นอายุที่รับเข้าไป และแยกการทำงานของโปรแกรมออกเป็นสองทิศทาง คือ ทิศทางที่เงื่อนไขเป็นจริง และทิศทางที่เงื่อนไขเป็นเท็จ จะเขียนผังงานและคำอธิบายโปรแกรมได้ดังนี้



1. รับค่าอายุมาเก็บในตัวแปร x
2. ถ้า x มากกว่า 25 แล้ว
   2.1 แสดงคำว่า “เป็นผู้ใหญ่แล้ว”
3. ชุดคำสั่งต่อไป

รูปแสดงผังงานและคำอธิบายโปรแกรม

ตัวอย่าง จงเขียนรหัสเทียมการทำงานจากผังงานในรูปต่อไปนี้



รูปแสดงผังงานที่มีการเลือกทำแบบสองทางเลือกซ้อนกัน

วิธีทำ จากผังงานที่โจทย์กำหนด จะเห็นว่ามีทั้งหมด 4 ขั้นตอนการทำงาน โดยขั้นตอนการทำงานที่ 3 จะเป็นขั้นตอนการทำงานแบบมีทางเลือก และถ้าเงื่อนไขเป็นเท็จก็จะมีการเลือกทำซ้อนขึ้นมาอีกทางหนึ่ง ผังงานนี้สามารถเขียนคำอธิบายได้ดังนี้

1. เริ่มต้นทำงาน
2. รับข้อมูลค่า X
3. ถ้า X > 0 แล้ว

   3.1 แสดงผลว่าเป็นค่าบวก

   มิฉะนั้น

   3.2 ถ้า X < 0 แล้ว

   3.2.1 แสดงผลเป็นค่าลบ

   มิฉะนั้น

   3.2.1 แสดงผลว่าเป็นค่าศูนย์
4. จบการทำงาน

แบบฝึกหัดท้ายบท

ตอนที่ 1 จงเลือกคำตอบที่ถูกต้องที่สุดเพียงหนึ่งข้อ

1. จงหาคำตอบของนิพจน์ (7%2) + (8%5) % 10

   ก. 3 ข. 4

   ค. 5 ง. 6
2. ตัวดำเนินการใดไม่ใช่ตัวดำเนินการทางตรรก

   ก. > ข. =

   ค. >= ง. !=
3. การเขียนประโยคใดต่อไปนี้ไม่สามารถใช้กำหนดเงื่อนไขให้กับ if ได้

   ก. m = 4.5 ข. x = = 8

   ค. ‘A’ > ‘B’ ง. C > D
4. ประโยคในข้อใด เป็นการตรวจสอบว่าตัวแปร x อยู่ในช่วงตั้งแต่ 20 ถึง 30

   ก. if((x >= 20) && (x <= 30)) ข. if((x >= 20) || (x <= 30))

   ค. if(20 < x < 30) ง. if((x <= 10) && (x >=20))
5. ในการทำซ้ำทุกรูปแบบจะต้องมีการประมวลผลแบบใด

   ก. การตรวจสอบเงื่อนไข ข. การคำนวณค่ากับตัวแปร

ค. การเพิ่มค่าให้ตัวแปร ง. การลดค่าตัวแปรลงหนึ่งค่า

6. คำสั่ง if – else ใช้กับเงื่อนไขแบบใด

ก. เงื่อนไขแบบทางเดียว ข. เงื่อนไขเลือกทำสองทาง

ค. เงื่อนไขเลือกทำหลายทาง ง. ไม่มีข้อถูก

7. ลูปประเภทใดที่ตรวจสอบเงื่อนไขก่อนการทำในลูป

ก. do..while ข. do..while และ for

ค. for ง. for และ while

ตอนที่ 2 จงตอบคำถามต่อไปนี้

1. เหตุใดจึงต้องมีการใช้คำสั่งควบคุม

2. รูปแบบของคำสั่ง if มีกี่แบบ จงอธิบายลักษณะการทำงานแต่ละแบบ

3. ถ้าหากผู้เรียนต้องการเขียนโปรแกรมเครื่องคิดเลขอย่างง่าย โดยให้ผู้ใช้กดคีย์ ‘+’, ‘-‘, ‘*’ และ ‘%’ จากนั้นให้โปรแกรมไปทำงานตามต้องการจะมีวิธีออกแบบโปรแกรมอย่างไร

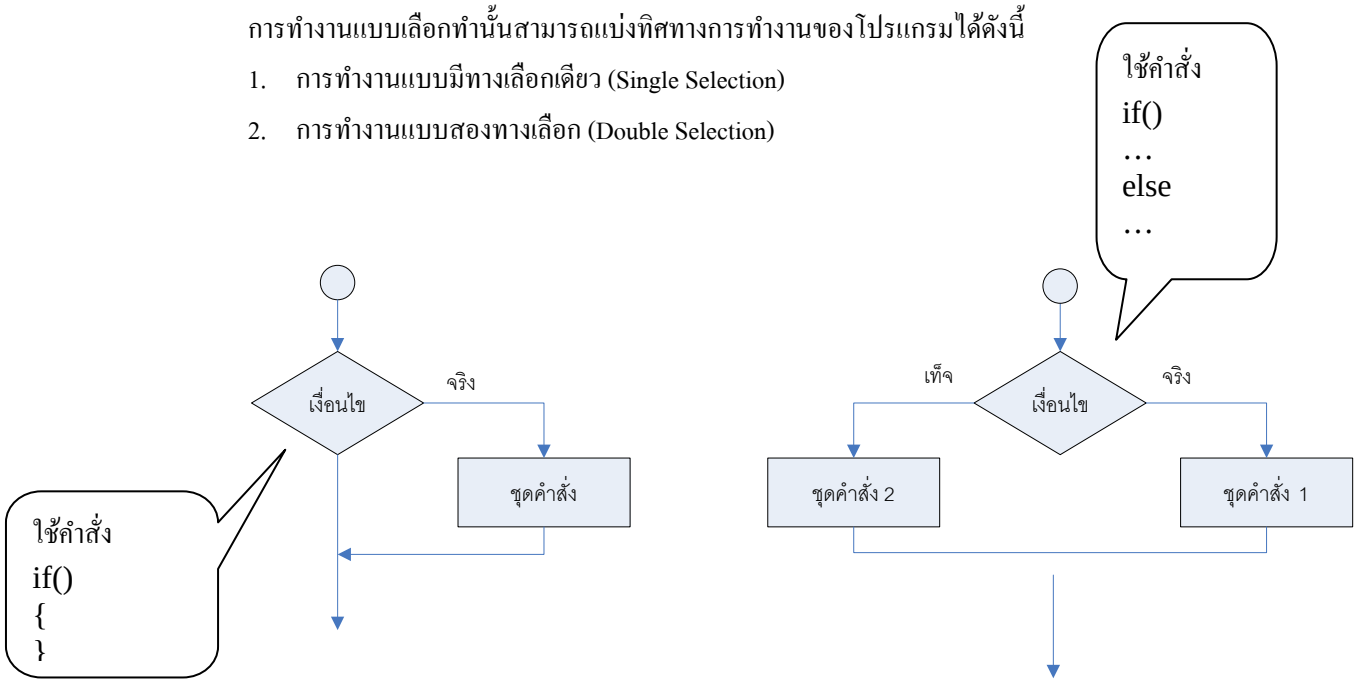
บทที่ 8
เขียนโปรแกรมแบบมีทางเลือก

การทำงานแบบลำดับในบทที่ 7 นั้นถือเป็นวิธีการทำงานที่ต้องมีในทุกโปรแกรม เมื่อนำมาพัฒนาเป็นโปรแกรมก็จะเสมือนว่าเป็นการนำคำสั่งต่าง ๆ ของภาษาโปรแกรมที่ต้องการเขียนมาต่อเรียงกันไปทำให้คอมพิวเตอร์ทำคำสั่งแต่ละคำสั่งต่อเนื่องกันไป(Sequential Connection) สำหรับโปรแกรมที่มีการทำงานที่ซับซ้อนขึ้นจะต้องมีการเปลี่ยนทิศทางการทำงานของโปรแกรมบ้าง เพื่อให้คอมพิวเตอร์สามารถเลือกได้ว่าคำสั่งหรือชุดคำสั่งใด ๆ คอมพิวเตอร์จะต้องทำหรือไม่หรือต้องทำในเวลาใด การทำงานของโปรแกรมในลักษณะนี้เรียกว่าการทำงานแบบมีทางเลือก หรือการทำงานแบบเลือกทำ

8.1 ประเภทของการทำงานแบบมีทางเลือก

การทำงานแบบเลือกทำนั้นสามารถแบ่งทิศทางการทำงานของโปรแกรมได้ดังนี้

1. การทำงานแบบมีทางเลือกเดียว (Single Selection)
2. การทำงานแบบสองทางเลือก (Double Selection)



ก. การเลือกทำแบบทางเลือกเดียว ข. การเลือกทำแบบสองทางเลือก

รูปที่ 8.1 การทำงานแบมีทางเลือก

สำหรับการทำงานแบบมีทางเลือกเดียวและแบบสองทางเลือกจะต้องมีการตรวจสอบเงื่อนไขเพื่อเป็นตัวพิจารณาว่าจะทำโปรแกรมในทิศทางใดต่อไป การตรวจสอบเงื่อนไขนั้นผลลัพธ์ที่ได้จะเป็นค่าทางบูลีน คือมีค่าเป็นไปได้ 2 กรณีคือ เป็นจริง (true) หรือเป็นเท็จ (false)

กรณีการเลือกทำแบบทางเลือกเดียว ถ้าหากเงื่อนไขเป็นจริง โปรแกรมจะทำชุดคำสั่งที่กำหนด แต่ถ้าหากเงื่อนไขเป็นเท็จโปรแกรมจะไม่ทำชุดคำสั่ง ส่วนการเลือกทำแบบสองทางเลือก(รูป ข) ถ้าหากเงื่อนไขเป็นจริง โปรแกรมจะทำชุดคำสั่งที่ 1 แต่ถ้าหากเงื่อนไขเป็นเท็จโปรแกรมจะทำชุดคำสั่งที่ 2

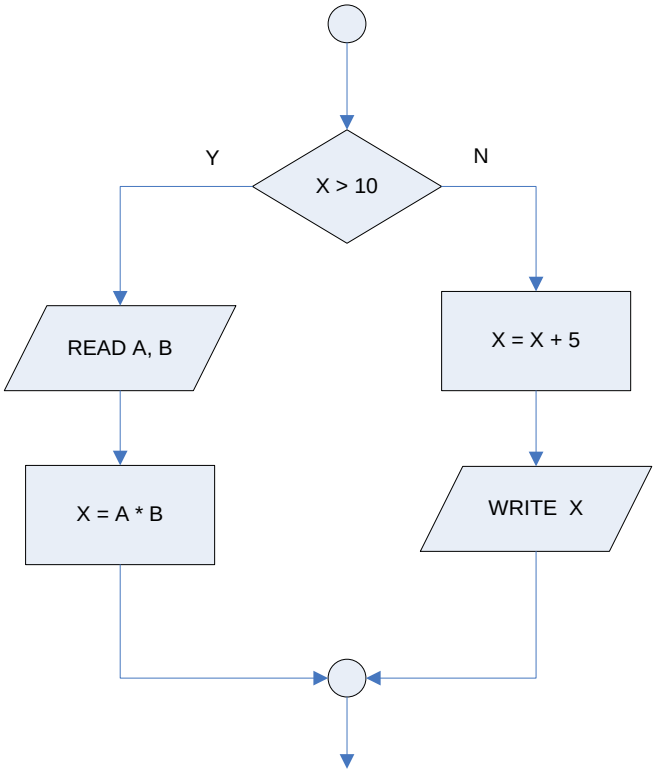
ในส่วนของการตรวจสอบเงื่อนไขนั้นจะเป็นการเขียนนิพจน์ที่แสดงความสัมพันธ์ระหว่างตัวแปรกับตัวแปร หรือระหว่างตัวแปรกับค่าคงที่ โดยจะนำตัวดำเนินการมาใช้ซึ่งส่วนใหญ่แล้วจะเป็นตัวดำเนินการเปรียบเทียบและตัวดำเนินการทางลอจิกดังที่ได้ศึกษามาในบทที่ 3

สำหรับคำอธิบายที่ใช้ในการเลือกทำจะใช้คำว่า “ถ้า”,”แล้ว” และ “มิฉะนั้นแล้ว” เป็นคำเริ่มต้นจากนั้นจะตามด้วยเงื่อนไขที่ใช้ตัดสินใจ

การเขียนคำอธิบายโปรแกรมจากผังงานแบบมีทางเลือก

ถ้าหากมีผังงานที่มีการเลือกทำไม่ว่าจะเป็นแบบทางเดียวหรือแบบสองทางแล้วต้องการเขียนคำอธิบายการทำงาน สิ่งแรกที่ต้องกระทำคือพยายามแยกกระบวนการทำงานทั้งหมดออกจากผังงานเป็นส่วน ๆ ก่อนจากนั้นให้พิจารณาทิศทางการทำงานของโปรแกรมทีละขั้นตอน แล้วจึงเขียนคำอธิบายการทำงานออกมา โดยคำอธิบายการทำงานควรมีหมายเลขบรรทัดด้วยและหมายเลขนี้ควรมีหมายเลขตามลำดับของสัญลักษณ์ของผังงาน

ตัวอย่างที่ 8.1 จงเปลี่ยนผังงานต่อไปนี้ให้เป็นคำอธิบายการทำงาน



รูปที่ 8.1 ผังงานโครงสร้างที่มีสองทางเลือก

วิธีทำ จากรูปเป็นการเลือกทำแบบสองทางเลือก ให้แยกการทำงานออกเป็นสองทิศทาง จะได้ว่าทิศทางการทำงานเมื่อเงื่อนไขเป็นจริงจะมีสองขั้นตอนการทำงาน และขั้นตอนการทำงานเมื่อเงื่อนไขเป็นเท็จจะมีสองขั้นตอนการทำงานเช่นกัน และใช้คำอธิบาย “ถ้า……แล้ว……มิฉะนั้น” แทนการเลือกทำ จะเขียนได้ดังนี้

1. ถ้า X > 10 แล้ว
    1.1 รับค่า A และรับค่า B
    1.2 คำนวณค่า X เท่ากับ A คูณกับ B
   มิฉะนั้น
    1.3 คำนวณค่า X เท่ากับ X บวก 5
    1.4 แสดงผลค่า X
2. ทำชุดคำสั่งต่อไป

8.2 การเขียนโปรแกรมสำหรับงานแบบมีทางเลือก

จากที่ได้ศึกษาการเขียนซูโดโค้ดมาแล้วจะพบว่าซูโดโค้ดที่นำมาใช้การเขียนโปรแกรมแบบเลือกทำจะใช้คำว่า IF หรือ IF-THEN-ELSE และ ENDIF ถ้าหากต้องการนำมาเขียนแทนคำอธิบายการทำงานของโปรแกรมจะใช้คำว่า “IF” แทนคำอธิบายว่า “ถ้า” ใช้คำว่า “THEN” แทนคำอธิบายว่า “แล้ว” และใช้คำว่า “ELST” แทนคำอธิบายว่า “มิฉะนั้น” และจบประโยคการเลือกทำด้วย ENDIF

จากปัญหาในตัวอย่างที่ 8.1 ถ้าหากนำคำอธิบายโปรแกรมในลักษณะของข้อความมาเขียนเป็นซูโดโค้ดจะเขียนได้ดังนี้

```
1. ถ้า X > 10 แล้ว
        1.1 รับค่า A และรับค่า B
        1.2 คำนวณค่า X เท่ากับ A คูณกับ B
   มิฉะนั้น
        1.3 คำนวณค่า X เท่ากับ X บวก 5
        1.4 แสดงผลค่า X
2. ทำชุดคำสั่งต่อไป
```

```
IF  X > 10  THEN
        READ A, B
        X = A * B
ELSE
        X = X + 5;
        WRITE X
ENDIF
```

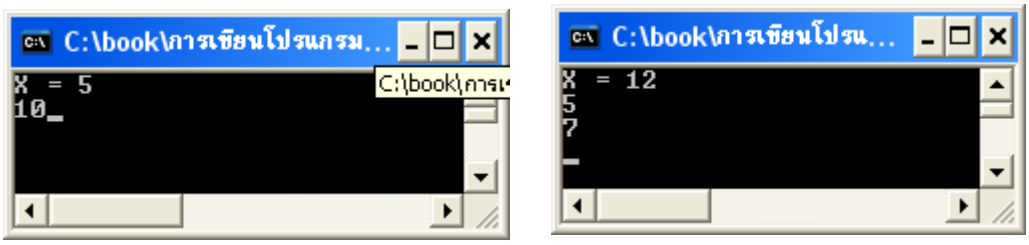
ถ้าหากนำซูโดโค้ดที่เขียนขึ้นไปพัฒนาเป็นโปรแกรมก็สามารถได้ง่าย แต่จะต้องทราบว่าในภาษาที่ต้องการเขียนโปรแกรมมีข้อกำหนดและรูปแบบในการเขียนโปรแกรมอย่างไร ถ้าหากนำซูโดโค้ดในตัวอย่างที่ 8.1 มาเขียนเป็นโปรแกรมภาษาซีจะเขียนได้ดังนี้

ซูโดโค้ด

```
IF  X > 10  THEN
        READ A, B
        X = A * B
ELSE
        X = X + 5;
        WRITE X
ENDIF
```

```
#include “stdio.h”
#include “conio.h”
main()
{
        int  A, B, X;
        printf(“X = “);
        scanf(“%d”, &X);
        if (X > 10)
        {
                scanf(“%d”,&A);
                scanf*”%d”,&B);
                X  = A+B;
        }
        else
        {
                X = X+5;
                printf(“%d”,X);
        }
        getch();
}
```

จากโปรแกรมที่เขียนขึ้นผลลัพธ์จากการรันโปรแกรมจะได้ดังรูปโดยครั้งแรกป้อน X เท่ากับ 5 ก็จะทำให้เงื่อนไขเป็นเท็จและจะไปทำการคำนวณ X บวกกับ 5 แล้วไปเก็บใน X ซึ่งทำให้ X มีค่าเป็น 10 แล้วแสดงผลออกมาทางหน้าจอ เมื่อรันโปรแกรมครั้งที่สองโดยป้อน X เท่ากับ 12 ก็จะทำให้เงื่อนไขของ IF เป็นจริง คอมพิวเตอร์ก็จะรับข้อมูลสองค่าไปเก็บในตัวแปร A และตัวแปร B

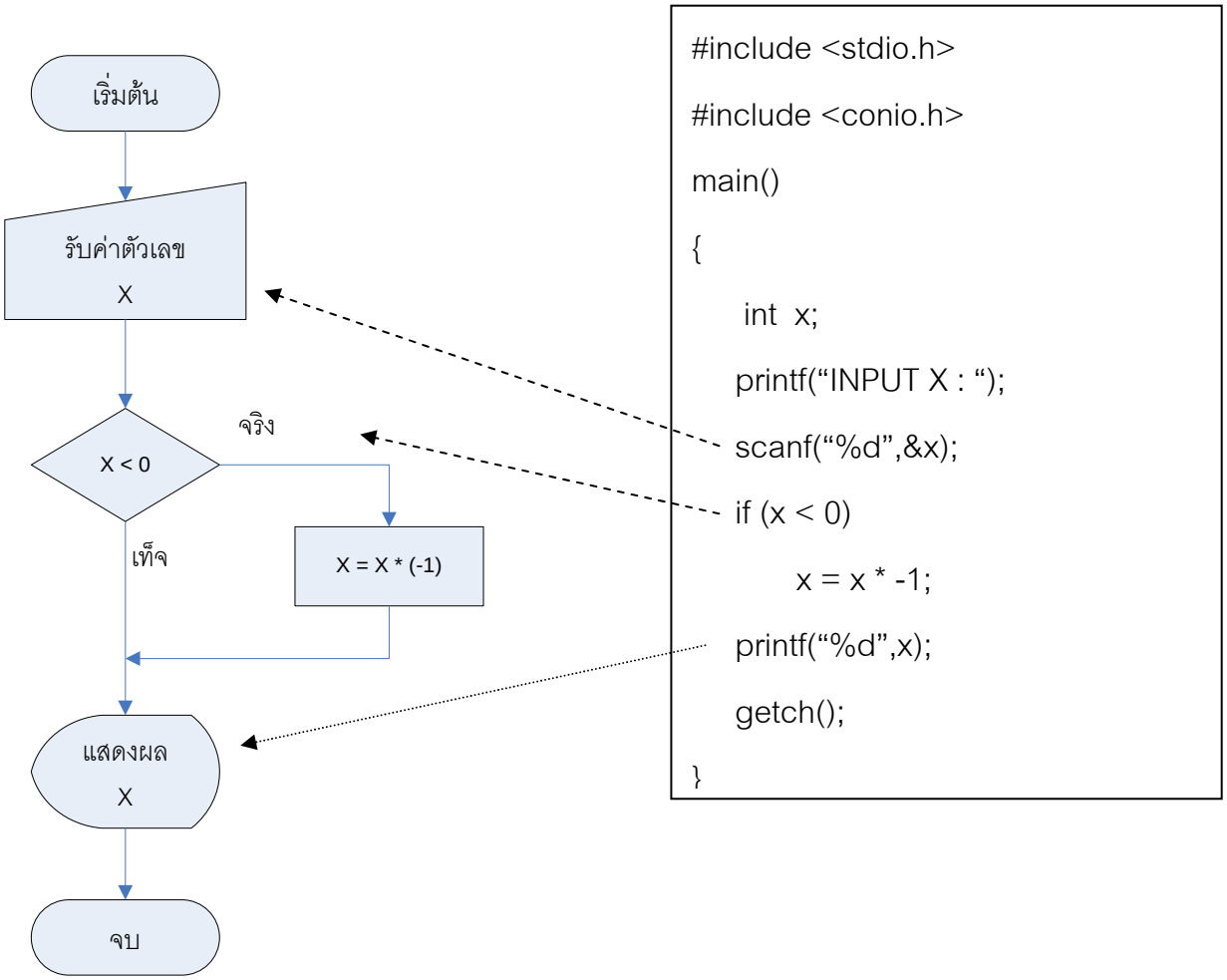


ผลลัพธ์จากการรันโปรแกรม

โปรแกรมที่ 8.1 จงเขียนโปรแกรมรับเลขจำนวนเต็มทางแล้วให้คอมพิวเตอร์แสดงค่าสัมบูรณ์ของเลขนั้นออกมา

วิธีทำ จากการวิเคราะห์ปัญหาของโจทย์จะพบว่าข้อมูลอินพุตคือตัวเลขจำนวนเต็ม เอาต์พุตก็เป็นเลขจำนวนเต็มที่เป็นค่าสัมบูรณ์ของเลขนั้น สำหรับวิธีการประมวลผลทำโดยตรวจสอบว่าเลขที่รับเข้ามานั้นมีค่ามากกว่า 0 หรือไม่ ถ้ามากกว่าให้แสดงเลขนั้น ถ้าน้อยกว่าให้ทำเป็นค่าบวกโดยนำค่า -1 ไปคูณ

จากปัญหานี้จะเป็นการเลือกทำแบบทางเดียว ซึ่งสามารถเขียนผังงานและโปรแกรมภาษาซีได้ดังรูปที่ 8.2



รูปที่ 8.2 ผังงานและโปรแกรมหาค่าสัมบูรณ์ของตัวเลข

โปรแกรมที่ 8.2 จงเขียนโปรแกรมรับตัวเลขสองค่า จากนั้นให้แสดงค่าที่มากกว่าทางจอภาพ

วิธีทำ อินพุต รับค่าตัวเลขสองค่า A และ B

เอาต์พุต แสดงค่าตัวเลขที่มากกว่าทางจอภาพ

วิธีการประมวลผล นำตัวเลข A และ B มาเปรียบเทียบ

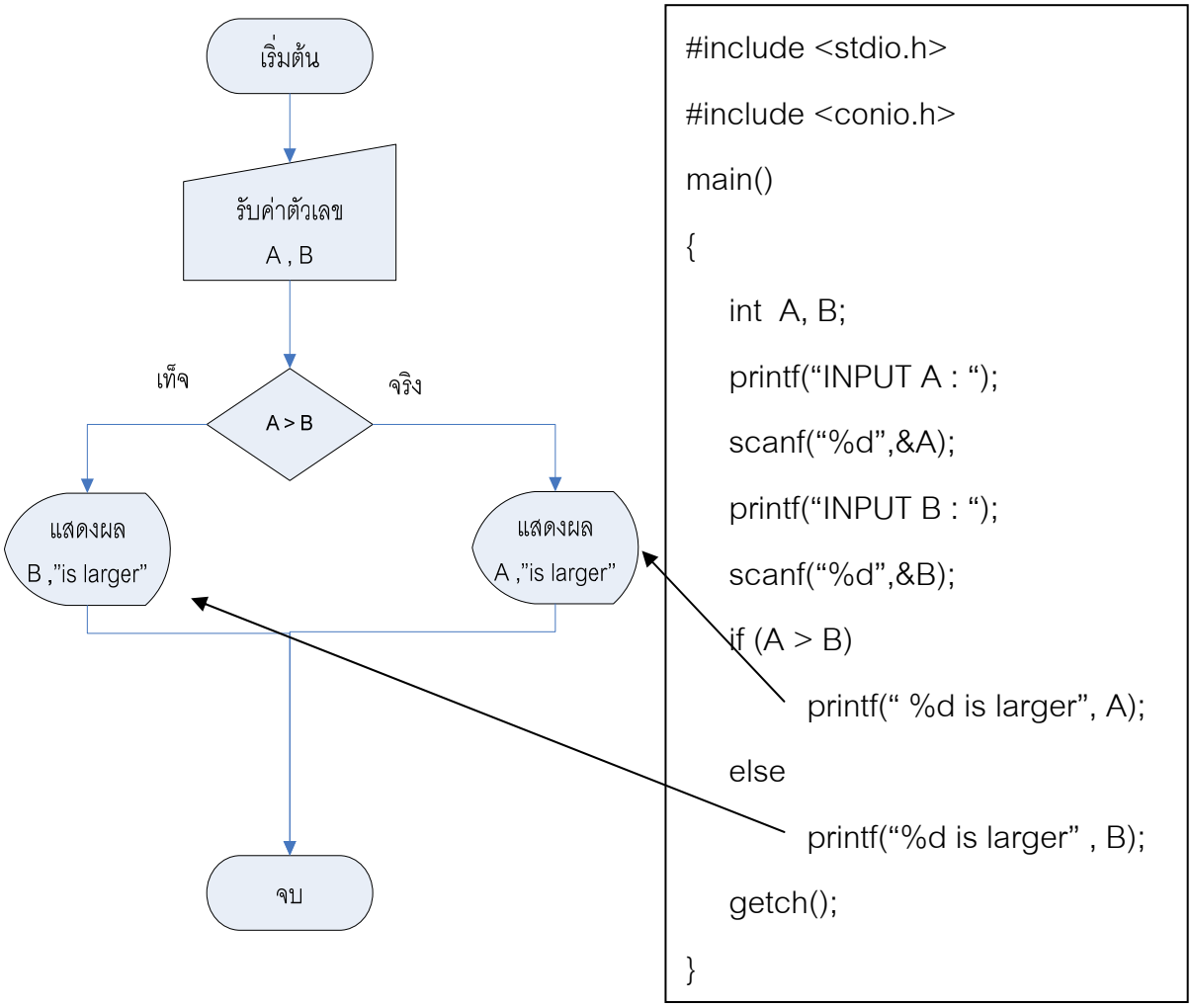
ถ้า A > B แล้ว

แสดงผลค่า A

มิฉะนั้น

แสดงผลค่า B

จากปัญหานี้จะเป็นการเลือกทำแบบสองทิศทาง การตรวจสอบเงื่อนไขทำโดยนำตัวเลขสองตัวมาเปรียบเทียบกัน สามารถเขียนเป็นผังงานและโปรแกรมได้ดังรูปที่ 8.3

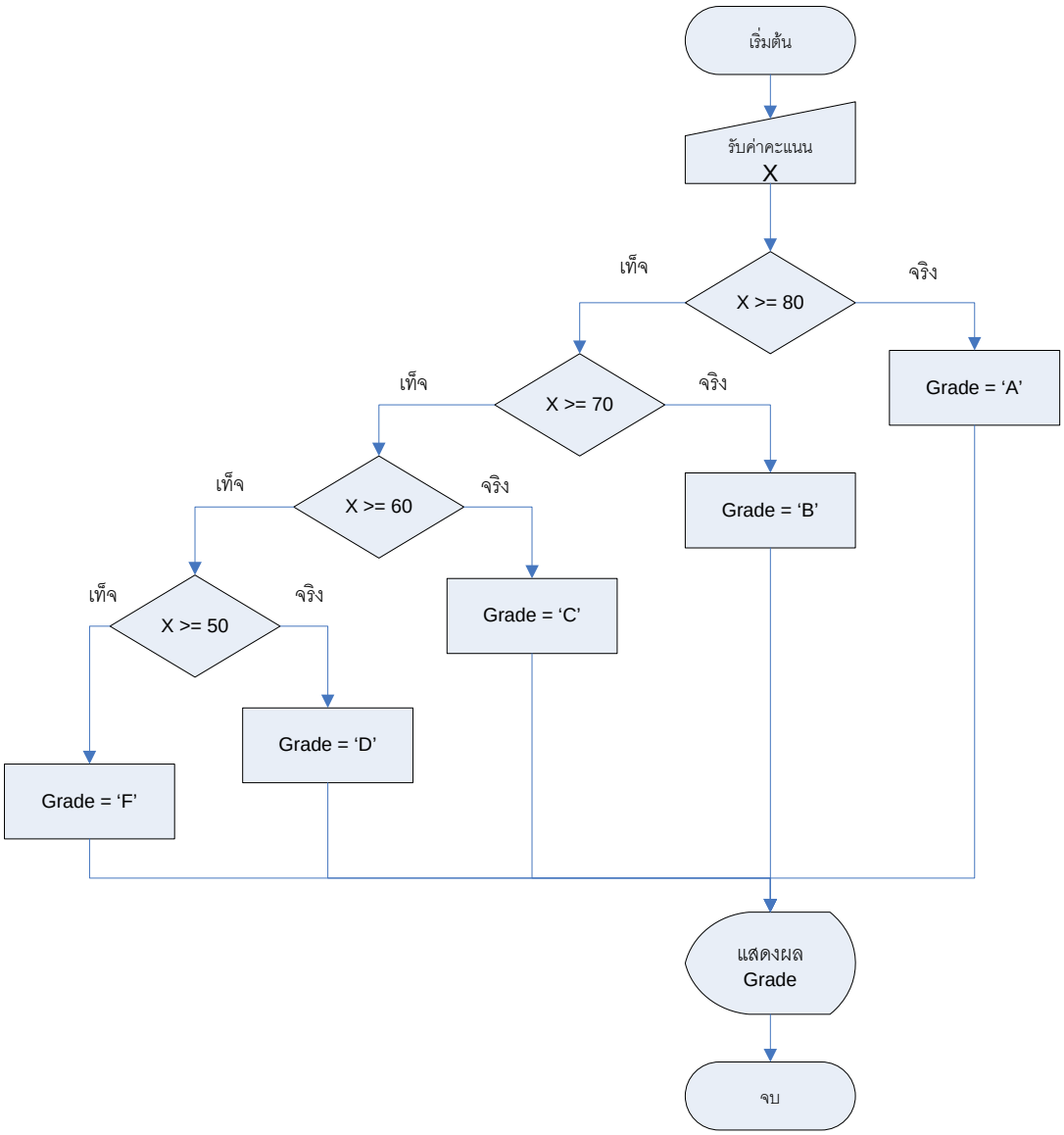


รูปที่ 8.3 ผังงานและโปรแกรมหาเลขสูงสุด

โปรแกรมที่ 8.3 จงเขียนผังงานและโปรแกรมตัดเกรดของนักเรียนโดยให้ป้อนคะแนนสอบเข้าไปจากนั้นให้แสดงเกรดออกมา โดยเงื่อนไขของเกณฑ์การตัดเกรดเป็นดังนี้

คะแนนตั้งแต่ 80 ขึ้นไป ได้เกรด "A"
คะแนนตั้งแต่ 70 ขึ้นไป ได้เกรด "B"
คะแนนตั้งแต่ 60 ขึ้นไป ได้เกรด "C"
คะแนนตั้งแต่ 50 ขึ้นไป ได้เกรด "D"
คะแนนต่ำกว่า 50 ได้เกรด "F"

วิธีทำ จากปัญหานี้สามารถนำการตรวจสอบเงื่อนไขแบบสองทางมาซ้อนกันให้เป็นการเลือกทำแบบหลายทางได้ ซึ่งเขียนเป็นผังงานได้ดังรูปที่ 8.4



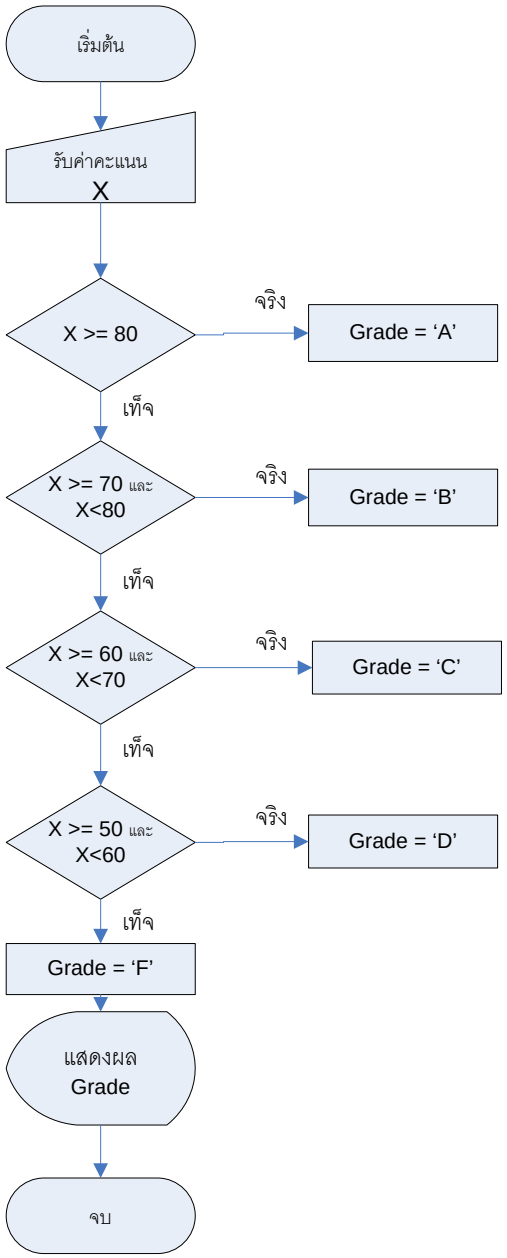
รูปที่ 8.4 ผังงานของโปรแกรมตัดเกรดโดยการนำการเลือกทำแบบสองทางมาซ้อนกัน

การเลือกทำในลักษณะนี้สามารถนำการเลือกทำแบบทางเดียวมาใช้ได้ แต่ต้องเปลี่ยนเงื่อนไขของการตัดสินใจเป็นการประมวลผลในลักษณะต่อไปนี้

ถ้าคะแนนตั้งแต่ 80 ขึ้นไป	ได้เกรด “A”
ถ้าคะแนนตั้งแต่ 70 ขึ้นไป แต่น้อยกว่า 80	ได้เกรด “B”
ถ้าคะแนนตั้งแต่ 60 ขึ้นไป แต่น้อยกว่า 70	ได้เกรด “C”
ถ้าคะแนนตั้งแต่ 50 ขึ้นไป แต่น้อยกว่า 60	ได้เกรด “D”
ถ้าคะแนนไม่ถึง 50	ได้เกรด “F”

ดังนั้นสามารถนำการเลือกทำแบบทางเดียวมาเขียนผังงานและโปรแกรมได้ดังรูปที่ 8.5

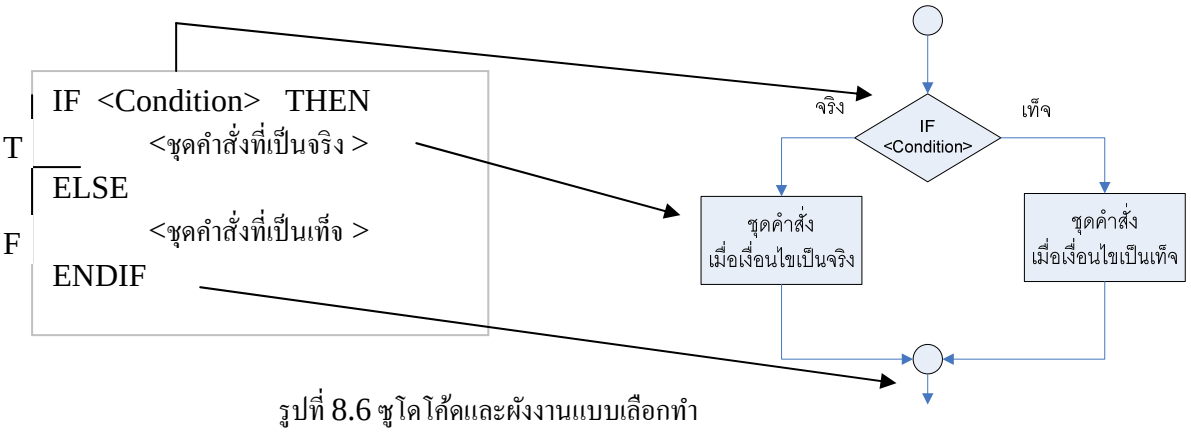
```
#include <stdio.h>
#include <conio.h>
main()
 {
      int  x;
      char  Grade;
      printf(“Input Score “);
      scanf(“%d”,&x);
      if (x >= 80)
          Grade = 'A';
     if ((x >= 70)&&(X < 80))
          Grade = 'B';
    if ((x >= 60)&&(x < 70))
          Grade = 'C';
    if ((x >= 50)&&(X < 60))
          Grade = 'D';
    if (X < 50)
          Grade = 'F';
    printf(“Grade = %c”,Grade);
    getch();
}
```



รูปที่ 8.5 การตัดเกรดโดยใช้การเลือกทำแบบทางเดียว

8.3 การเขียนผังงานจากซูโดโค้ดแบบมีทางเลือก

ผังงานนอกจากจะเขียนขึ้นมาจากการวิเคราะห์ปัญหาแล้วยังเขียนขึ้นมาจากซูโดโค้ดได้เช่นกัน สำหรับกรณีที่โปรแกรมเป็นแบบมีทางเลือก ให้พยายามแยกประโยค IF ออกมาเป็นกลุ่ม ๆ แล้วพิจารณาว่าถ้าเงื่อนไขเป็นจริงจะทำทิศทางใด เงื่อนไขเป็นเท็จจะทำทิศทางใด



รูปที่ 8.6 ซูโดโค้ดและผังงานแบบเลือกทำ

จากรูปที่ 8.6 จะเห็นว่าการตรวจสอบเงื่อนไขหลังคำว่า IF จะนำไปเขียนในสี่เหลี่ยมขนมเปียกปูน และนำชุดคำสั่งที่เป็นจริงมาเขียนในสัญลักษณ์การประมวลผลเมื่อเป็นจริง นำชุดคำสั่งหลัง ELSE มาเขียนในสัญลักษณ์การประมวลผลเมื่อเป็นเท็จ สำหรับโปรแกรมที่มีเงื่อนไขการเลือกทำซ้อนกันก็ใช้วิธีการแยกประโยค IF ออกมาเป็นกลุ่มในลักษณะเดียวกัน

ตัวอย่างที่ 8.2 ร้านค้าแห่งหนึ่งต้องการพัฒนาโปรแกรมคอมพิวเตอร์เพื่อจ่ายค่าตอบแทนกับพนักงานขาย โดยถ้าหากเป็นพนักงานประจำจะจ่ายเป็นเงินเดือนปกติ ถ้าหากเป็นพนักงานที่ทำงานเป็นชั่วโมงจะจ่ายเป็นรายชั่วโมง แต่ถ้าหากทำงานเกิน 40 ชั่วโมง จำนวนชั่วโมงที่เกินจะได้ค่าแรงเป็น 1.5 เท่าของอัตราปกติ จงเขียนซูโดโค้ดและผังงานของโปรแกรมนี้

วิธีทำ

สิ่งที่ต้องการ ค่าตอบแทนพนักงาน

ข้อมูลเข้า ประเภทของพนักงาน (PayType), จำนวนชั่วโมง (Hours)

วิธีประมวลผล

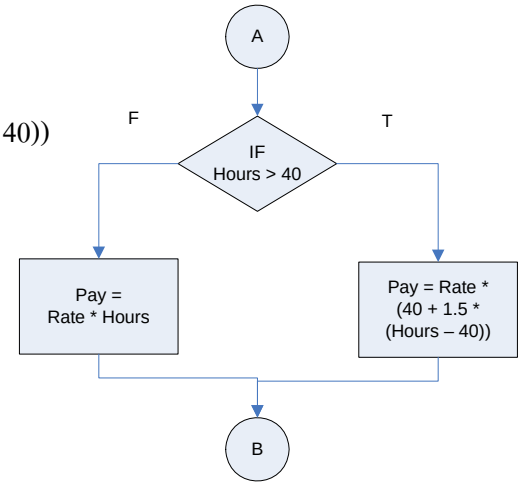
วิธีการประมวลผลในส่วนของการคำนวณค่าตอบแทนเป็นดังนี้

1. ตรวจสอบว่าเป็นพนักงานประเภททำเป็นชั่วโมงหรือไม่
2. ถ้าเป็นประเภททำงานเป็นรายชั่วโมง ตรวจสอบว่าทำเกิน 40 ชั่วโมงหรือไม่
3. ถ้าทำงานเกิน 40 ชั่วโมง

   ค่าตอบแทน = ค่าตอบแทน 40 ชั่วโมง + 1.5 เท่าของชั่วโมงที่เกิน
4. ถ้าทำงานไม่เกิน 40 ชั่วโมง ได้ค่าตอบแทนเป็นรายชั่วโมง
5. ถ้าไม่ใช่พนักงานที่ทำงานเป็นชั่วโมงให้ได้เงินเดือนปกติ

การประมวลผลจะเป็นแบบมีการเลือกทำซ้อนกัน ดังนั้นควรเขียนเงื่อนในภายในก่อน

```
T   IF  Hours > 40 THEN
            Pay = Rate * (40 +1.5 *(Hours - 40))
    ELSE
F           Pay = Rate * Hours
```

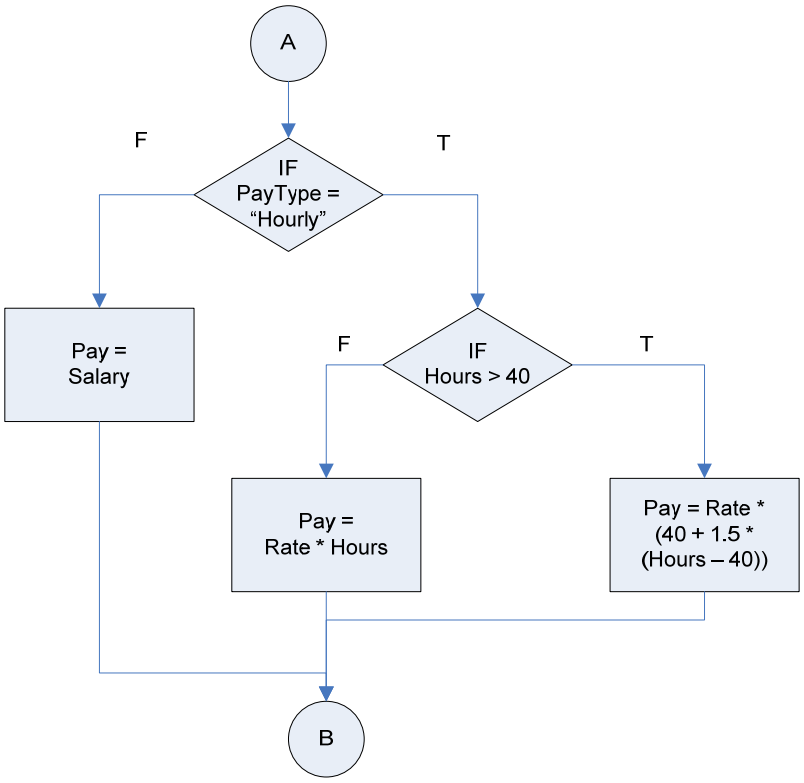


รูปที่ 8.7 การเขียนซูโดโค้ดและผังงานเงื่อนไขภายใน

ถ้าหากเขียนเป็นซูโดโค้ดของการประมวลผลทั้งหมด ก็เขียนเงื่อนไขภายนอกเพิ่มเข้าไปจะได้ดังนี้

```
T   IF      PayType = “Hourly”   THEN
        T   IF      Hours  >  40   THEN
                        Pay  =  Rate * (40 + 1.5 * (Hours – 40))
            ELSE
        F               Pay  =  Rate * Hours
            ENDIF
F   ELSE
            Pay  =  Salary
    ENDIF
```

จากซูโดโค้ดที่เขียนขึ้น และแบ่งส่วนที่เป็นจริงและเป็นเท็จของประโยค IF..THEN..ELSE เมื่อเขียนเป็นผังงานให้แทนสัญลักษณ์ของการตัดสินใจในตำแหน่งของคำว่า “IF” และเขียนประโยคการตัดสินใจไว้ภายในสัญลักษณ์ จะได้ผังงานดังรูปที่ 8.8



รูปที่ 8.8 ผังงานในส่วนของการคำนวณค่าตอบแทน

งานแบบมีทางเลือกบางประเภทสามารถเขียนซูโดโค้ดหรือเขียนผังงานได้ทั้งแบบยาวโดยตรวจสอบเงื่อนไขไปเรื่อย ๆ และแบบมีเงื่อนไขซ้อนกันดังตัวอย่างที่ 9.6

ตัวอย่างที่ 8.3 การแสดงคอนเสิร์ตรายการหนึ่งจะขายบัตรชมคอนเสิร์ตที่มีราคาต่างกัน โดยมีราคาชาร์ตเพิ่มตามอายุผู้ชมดังนี้

อายุ	**Charge (%)**
Age < 16	7
Age >= 16 และ Age <65	10
Age >= 65	5

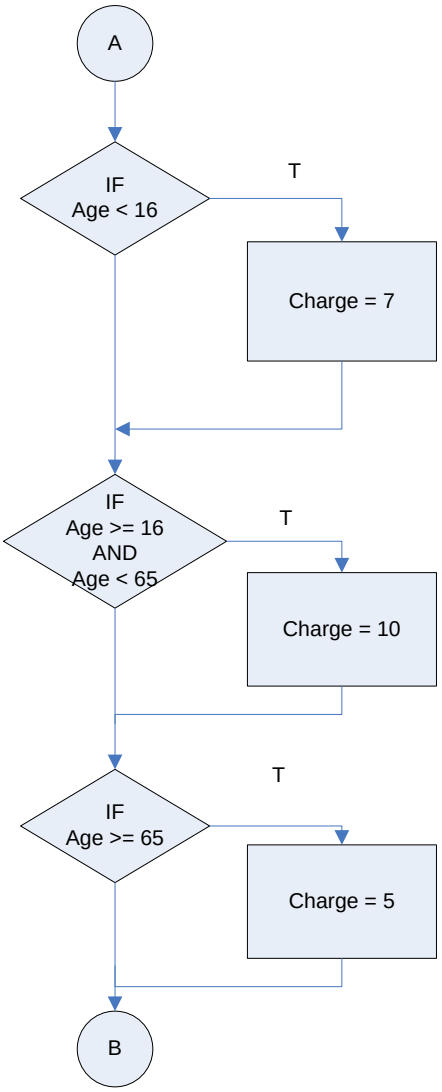
จงเขียนซูโดโค้ดและเขียนผังงานจากซูโดโค้ดของปัญหานี้

วิธีทำ สามารถนำการเลือกทำแบบทางเดียวมาใช้ได้ และให้แยกการเลือกทำเป็นส่วน ๆ จะได้ส่วนของโปรแกรมดังนี้

```
T   IF  Age < 16  THEN
        Charge = 7
    ENDIF

T   IF  Age >= 16 AND Age < 65  THEN
        Charge = 10
    ENDIF

T   IF Age >= 65  THEN
        Charge = 5
    ENDIF
```

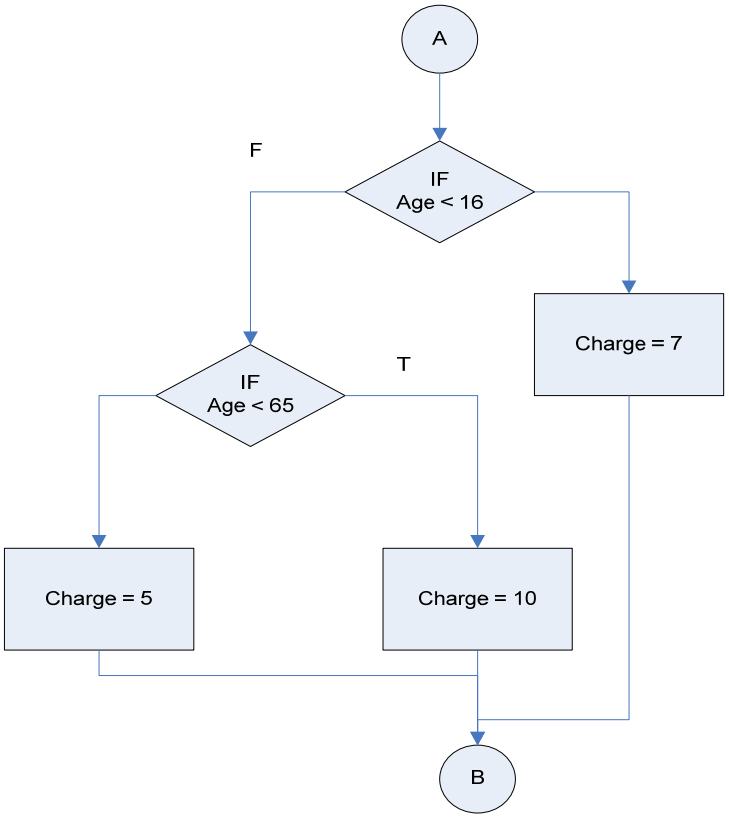


รูปที่ 8.9 ซูโดโค้ดและผังงานของส่วนของโปรแกรมปัญหาการคิดราคาตั๋ว

นอกจากนี้ปัญหาในข้อนี้ สามารถเขียนซูโดโค้ดและผังงานแบบมีการเลือกทำซ้อนกันได้ดังนี้

```
T   IF      Age < 16   THEN
        Charge = 7
    ELSE
        IF    Age < 65  THEN
T           Charge = 10
        ELSE
F           Charge = 5
F   ENDIF
    ENDIF
```

หากต้องการเขียนผังงานก็ทำโดยนำสัญลักษณ์การเลือกทำมาแทนคำว่า “IF” และเขียนเงื่อนไขการเลือกทำลงไปในสัญลักษณ์ จะได้ดังรูปที่ 8.10

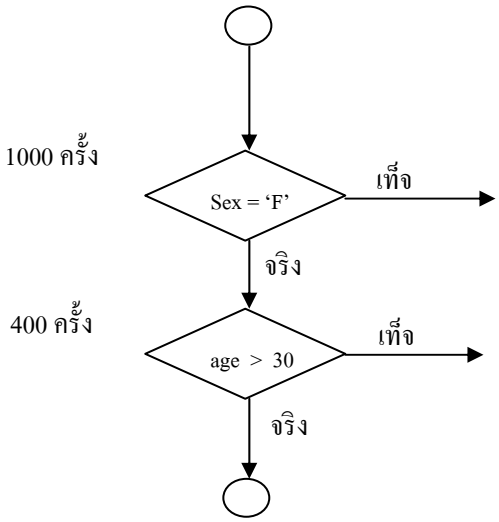


รูปที่ 8.10 ผังงานที่เขียนจากซูโดโค้ดข้างบน

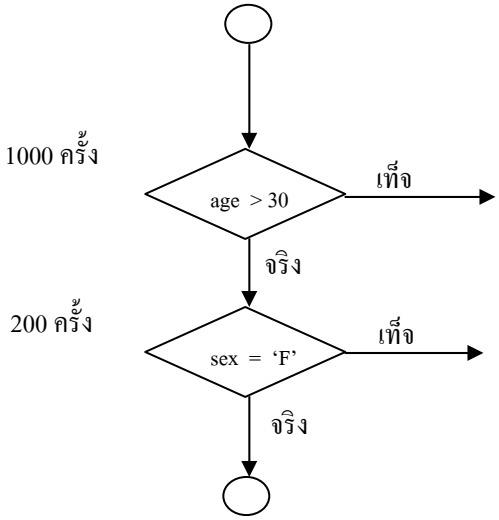
จากตัวอย่างที่ 8.3 ถ้าหากพิจารณาการทำงานของโปรแกรมจะพบว่า ถ้าหากผู้ซื้อบัตรมีอายุมากกว่า 65 ปี การเขียนโปรแกรมในแบบแรกนั้นโปรแกรมจะต้องตรวจสอบเงื่อนไข 3 ครั้ง หรือตรวจสอบ 3 เงื่อนไข แต่ถ้าหากเขียนโปรแกรมที่การเลือกทำแบบซ้อน การตรวจสอบเงื่อนไขจะทำ 2 เงื่อนไขเท่านั้น ทำให้ในกรณีนี้การเขียนโปรแกรมที่เลือกทำแบบซ้อนโปรแกรมจะทำงานได้เร็วกว่า สำหรับปัญหาในตัวอย่างที่ 8.3 จะไม่มีผลต่อประสิทธิภาพของโปรแกรมมากนัก เนื่องจากในการขายตั๋วผู้ซื้อจะมาซื้อครั้งละไม่กี่ใบ แต่ถ้าหากต้องการประมวลผลกับข้อมูลจำนวนมากการออกแบบโปรแกรมควรจะพิจารณาถึงลำดับการตรวจสอบเงื่อนไขด้วย ตัวอย่างเช่น ถ้าหากมีแฟ้มข้อมูลพนักงานประกอบด้วยข้อมูลจำนวน 1000 เรคอร์ด โดยข้อมูลที่เก็บอยู่มีลักษณะดังต่อไปนี้

ลักษณะของข้อมูล	จำนวน	ร้อยละ
เพศชาย	600	60
เพศหญิง	400	40
อายุมากกว่า 30 ปี	200	20
อายุน้อยกว่า 30 ปี	800	80

จากปัญหาดังกล่าวถ้าหากต้องการเขียนโปรแกรมสำหรับค้นหาพนักงานที่เป็นผู้หญิง ที่มีอายุมากกว่า 30 ปี โดยให้โปรแกรมอ่านข้อมูลทั้งหมดขึ้นมา 1000 เรคอร์ด โดยมีการประมวลผลดังนี้



จากข้อมูลในตารางเบื้องต้นจะเห็นว่า ถ้าหากให้โปรแกรมตรวจสอบเงื่อนไขที่เป็นเพศก่อนว่าเป็นผู้หญิงหรือไม่แล้วจึงเปรียบเทียบอายุ จะพบว่าโปรแกรมจะตรวจสอบเงื่อนไขทั้งหมด 1400 ครั้ง แต่ถ้าหากให้โปรแกรมทำงานตรวจสอบอายุก่อน ดังวิธีการต่อไปนี้



จะเห็นว่าถ้าหากเปลี่ยนเงื่อนไขการตรวจสอบ จะทำให้จำนวนครั้งในการตรวจสอบเงื่อนไขลดลงเหลือ 1200 ครั้ง ซึ่งทำให้ประสิทธิภาพของโปรแกรมดีกว่ากรณีแรก ดังนั้นในการออกแบบโปรแกรมที่มีการตรวจสอบเงื่อนไขหลาย ๆ อย่าง ผู้เขียนโปรแกรมควรพิจารณาถึงเงื่อนไขก่อนหลังในการใช้คำสั่งตัดสินใจเลือกทำในแต่ละครั้งด้วย

ปัญหาการสลับค่า

ตัวอย่างปัญหาหนึ่งที่นำการเขียนโปรแกรมแบบตัดสินใจมาใช้ เช่น ปัญหาการเรียงลำดับข้อมูล ซึ่งจะต้องนำข้อมูลมาเปรียบเทียบและสลับค่าข้อมูลตามเงื่อนไขที่กำหนด ตัวอย่างเช่นถ้ามีตัวแปร X และ Y เก็บค่าตัวเลขสองค่า และต้องการออกแบบโปรแกรมให้ X เก็บค่ามาก และตัวแปร Y เก็บค่าน้อย อย่างเช่นถ้ามีค่าดังต่อไปนี้

	X	Y		X	Y
เริ่มต้น	4	6	เปลี่ยนเป็น	6	4

	X	Y	
เริ่มต้น	8	0	ไม่ต้องเปลี่ยนเนื่องจาก X เก็บค่ามากอยู่แล้ว

	X	Y	
เริ่มต้น	6	6	ไม่ต้องเปลี่ยนเนื่องจากตัวแปรทั้งสองมีค่าเท่ากัน

จากปัญหานี้สามารถนำการเลือกทำแบบ IF-THEN-ELSE-ENDIF มาใช้ได้ แต่จะต้องวางแผนการเขียนโปรแกรมให้ดีในการสลับค่า ตัวอย่างเช่น ถ้า X เก็บค่า 14 และ Y เก็บค่า 20 แล้วใช้วิธีการประมวลผลดังต่อไปนี้

	X	Y
ค่าเริ่มต้น	14	20
สลับค่า Y ไปเก็บใน X	20	20
สลับค่า X ไปเก็บใน Y	20	20

จะพบว่าหลังการสลับค่าผลลัพธ์ที่ได้จะไม่ถูกต้อง เนื่องจากข้อมูลทั้งสองตัวทับกันมีค่าเป็น 20 ทั้งคู่ แต่ถ้าหากสลับวิธีการสลับค่าดังต่อไปนี้

	X	Y
ค่าเริ่มต้น	14	20
สลับค่า X ไปเก็บใน Y	14	14
สลับค่า Y ไปเก็บใน X	14	14

ค่าผลลัพธ์ก็จะไม่ถูกต้องเช่นกัน แม้ว่าเปลี่ยนวิธีการสลับค่าแล้ว ผลลัพธ์จะไม่เหมือนกัน แต่ค่าที่ได้ก็ไม่ถูกต้อง ปัญหานี้จะต้องสร้างตัวแปรพิเศษขึ้นมาตัวหนึ่ง เพื่อเป็นที่พักข้อมูลในการสลับค่า สมมติว่าให้ตัวแปรนั้นชื่อว่า TEMP ปัญหานี้จะออกแบบวิธีการประมวลผลได้ดังนี้

	X	Y	TEMP
ค่าเริ่มต้น	14	20	-
ย้าย X ไปเก็บใน TEMP	14	20	14
ย้าย Y ไปเก็บใน X	20	20	14
ย้าย TEMP ไปเก็บใน Y	20	14	14

จะทำให้ตัวแปร X เก็บค่าที่มากกว่าได้ถูกต้อง สำหรับวิธีการสลับข้อมูลระหว่างตัวแปรสองตัวนี้เรียกว่า swap วิธีการแก้ปัญหานี้สามารถเขียนเป็นซูโดโค้ดโดยใช้การเลือกทำแบบทางเดียวได้ดังนี้

```
IF  X < Y  THEN
        MOVE  X  TO  TEMP
        MOVE  Y  TO  X
        MOVE  TEMP  TO  Y
ENDIF
```

ตัวอย่างที่ 8.4 ถ้าหากต้องการรับตัวอักขระจำนวนสามตัว จากนั้นให้แสดงตัวอักขระเรียงตามรหัสแอสกีแล้วแสดงออกทางจอภาพ จงเขียนซูโดโค้ดของการแก้ปัญหานี้ พร้อมทั้งแสดงวิธีการตรวจสอบคำตอบ

วิธีทำ **ต้องการอะไร** เรียงลำดับตัวอักขระจำนวนสามตัวตามรหัสแอสกี
ต้องการเอาต์พุตอย่างไร แสดงตัวอักขระทั้งสามตัวเรียงกันทางจอภาพ
ข้อมูลเข้า ตัวอักขระจำนวนสามตัว (char_1, char_2, char_3)
วิธีการประมวลผล จับคู่อักขระทีละสองตัวแล้วนำมาเปรียบเทียบกัน
ถ้าตัวแรกมีรหัสแอสกีมากกว่าตัวที่สองจริง
ให้สลับระหว่างตัวแรกกับตัวที่สอง

การสลับค่าข้อมูลนี้เรียกว่า swap สามารถเขียนขั้นตอนได้ดังนี้ โดยประกาศตัวแปรที่พักข้อมูล (temp) ขึ้นมา

```
temp     char_1
char_1 = char_2
char_2 = temp
```

ดังนั้นจะเขียนซูโดโค้ดในส่วนของการรับข้อมูลแล้วเรียงลำดับของปัญหานี้ได้ดังนี้

```
1.  READ char_1, char_2, char_3
2.  IF char_1 > char_2 THEN
        temp  =  char_1
        char_1  =  char_2
        char_2  =  temp
    ENDIF
3.  IF char_2 > char_3 THEN
        temp  =  char_2
        char_2  =  char_3
        char_3  =  temp
    ENDIF
```

4. IF char_1 > char_2 THEN

temp = char_1

char_1 = char_2

char_2 = temp

ENDIF

5. PRINT To The screen char_1, char_2, char_3

สำหรับการตรวจสอบความถูกต้องของขั้นตอนการทำงานสามารถทำโดยการสมมุติค่าตัวแปรขึ้นมาแล้วทดลองแทนค่าดังขั้นตอนต่อไปนี้

1. สร้างตัวเลขทดสอบทางอินพุต

ตัวแปร	ข้อมูลชุดแรก	ข้อมูลชุดที่สอง
char_1	K	z
char_2	B	s
char_3	G	a

2. ลองคิดคำตอบการประมวลผลด้วยตนเอง

ตัวแปร	ข้อมูลชุดแรก	ข้อมูลชุดที่สอง
char_1	B	a
char_2	G	s
char_3	K	z

3. สร้างตารางขึ้นมาแสดงการทำงานและดูค่าตัวแปรทีละขั้นตอน

ขั้นตอน	char_1	char_2	char_3	Temp
ข้อมูลชุดแรก				
1	k	B	g	
2	b	K		k
3		G	k	k
4				
5	แสดงผล	แสดงผล	แสดงผล	
ข้อมูลชุดที่สอง				
1	z	S	a	
2	s	Z		z
3		A	z	z
4	a	S		
5	แสดงผล	แสดงผล	แสดงผล	

4. จะพบว่าการแสดงผลในตารางในข้อ 3 จะตรงกับตารางในข้อ 2 ดังนั้นวิธีการนี้ทำได้

ตัวอย่างที่ 8.5 ร้านค้าแห่งหนึ่งจำหน่ายอุปกรณ์ไฟฟ้า สามารถใช้ลูกค้าเลือกวิธีการชำระเงินได้ 2 วิธี โดยแต่ละวิธีจะทำให้ราคาสินค้าไม่เท่ากันดังนี้

1. ซื้อด้วยเงินสด มีส่วนลด 5% จากราคาสินค้า
2. ซื้อแบบผ่อนชำระ จะคิดค่าดอกเบี้ย 2% ต่อเดือน โดยที่ลูกค้าสามารถเลือกจำนวนเดือนสำหรับการผ่อนชำระได้

จงเขียนขั้นตอนการทำงานในลักษณะข้อความ และผังงานสำหรับการแก้ปัญหานี้

วิธีทำ จากโจทย์สามารถวิเคราะห์ปัญหาได้ดังนี้

ต้องการอะไร คำนวณราคาสินค่าที่ต้องชำระหรือราคาที่ผ่อนเป็นรายเดือน

ต้องการเอาต์พุตอย่างไร ราคาสินค้าที่ต้องชำระทางจอภาพ

ข้อมูลเข้า ราคาสินค้า(Price), วิธีการชำระเงิน(choice), จำนวนเดือนที่ผ่อน

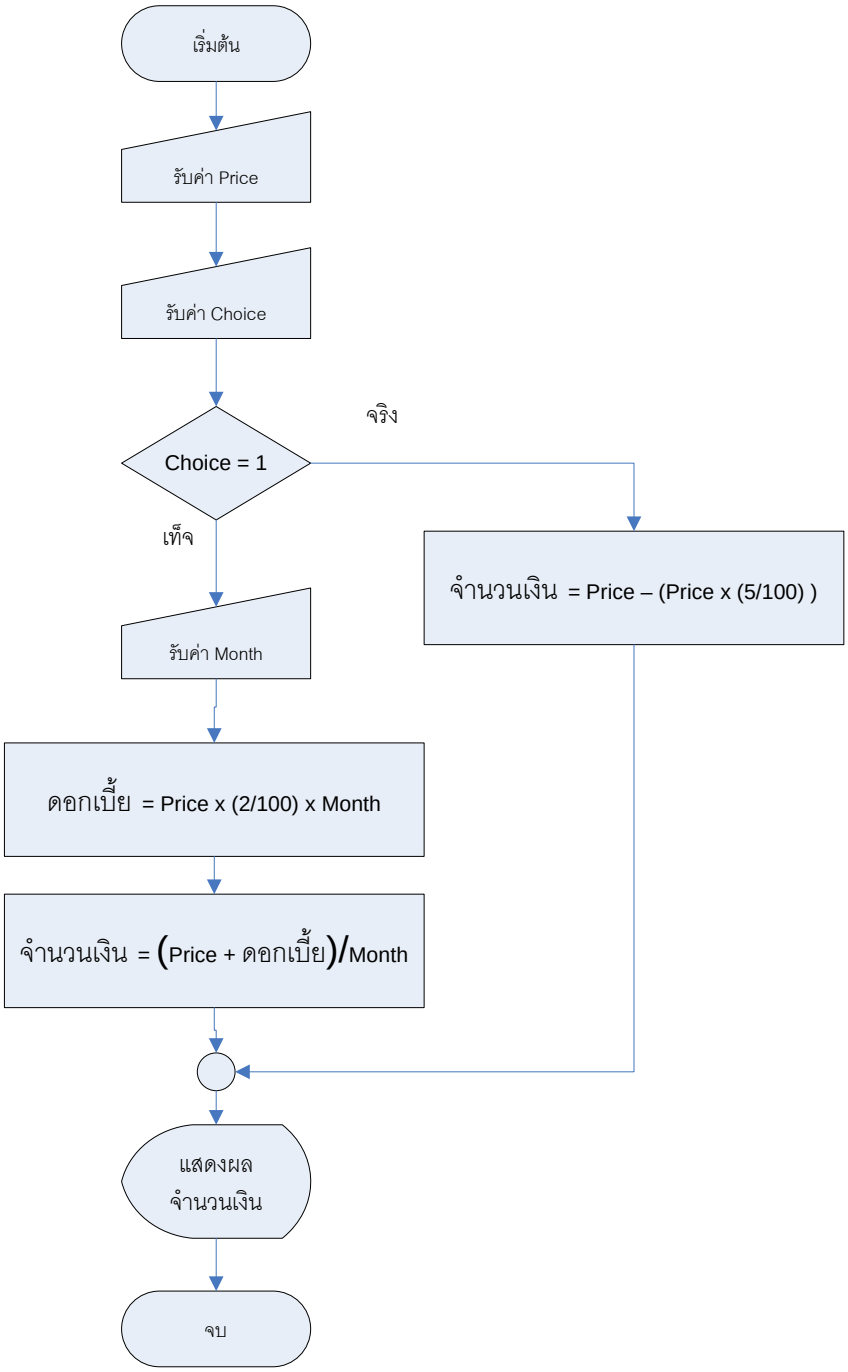
วิธีการประมวลผล

ถ้าเลือกวิธีที่ 1 จำนวนเงินที่ต้องชำระคือราคาสินค้าลบด้วยส่วนลด 5%
ถ้าเลือกวิธีที่ 2 จำนวนเงินที่ต้องชำระรายเดือนคือราคาสินค้ารวมดอกเบี้ยแล้วหารด้วยจำนวนเดือน โดยที่ดอกเบี้ยหาได้จาก
จำนวนเดือน(Month) x ราคาสินค้า(Price) x 2%

ดังนั้นขั้นตอนการทำงานในลักษณะข้อความสามารถเขียนได้ดังนี้

1. เริ่มต้น
2. รับค่าราคาสินค้า (Price)
3. รับวิธีการชำระเงิน (Choice)
4. ถ้า Choice = 1 แล้ว
   - 4.1 จำนวนเงินที่ต้องชำระ = Price – (Price *(5/100))
     มิฉะนั้น
   - 4.2 รับจำนวนเดือน (Month)
   - 4.3 ภาษี = Price x Month x (2/100)
   - 4.4 จำนวนเงินที่ต้องชำระ = (Price + ภาษี) / Month
5. แสดงผลจำนวนเงินที่ต้องชำระทางจอภาพ
6. จบ

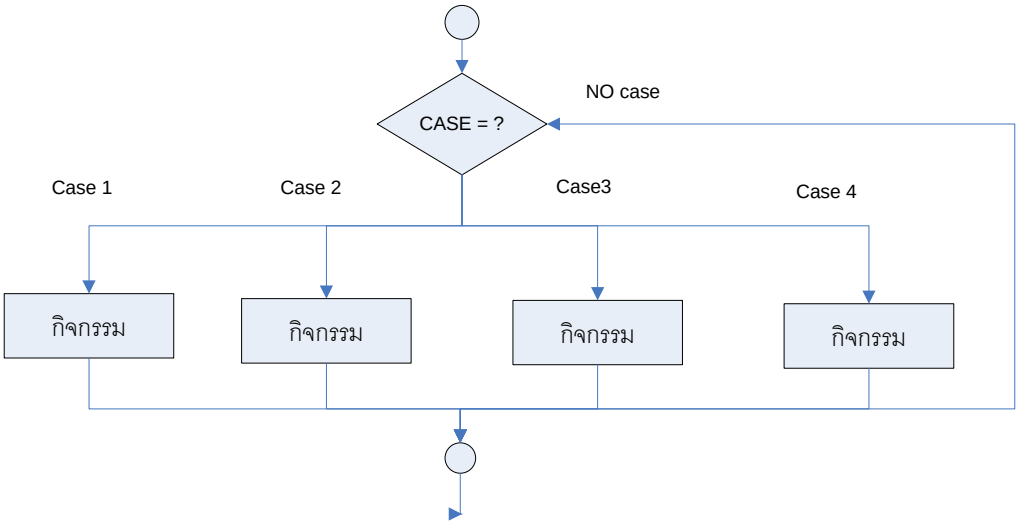
สำหรับผังงานของปัญหานี้เขียนได้ดังนี้



รูปที่ 8.11 ผังงานของปัญหาในตัวอย่างที่ 8.5

8.4 การทำงานแบบมีทางเลือกหลายทาง

สำหรับการทำงานที่มีทางเลือกหลายทางนั้นสามารถเขียนประโยค IF..THEN..ELSE มาซ้อนกันได้ แต่ถ้าหากมีทางเลือกมากขึ้นจะทำให้การเขียนแบบ IF ดูแล้วซับซ้อนและไม่ค่อยสะดวก ดังนั้นจึงมีการนำคำว่า “CASE”, “OF” และ “ENDCASE” มาใช้ พิจารณารูปที่ 8.12



รูปที่ 8.12 ตัวอย่างการเลือกทำแบบหลายทาง

เมื่อการทำงานของระบบเข้าสู่เงื่อนไขการเลือกทำ ระบบจะตรวจสอบว่าค่าคงที่ที่ได้จากการตรวจสอบเงื่อนไข (CASE)นั้น ถ้าหากทราบว่ามีค่าเท่ากับค่าคงที่หรือ CASE ใด ก็จะกระโดดไปทำกิจกรรมที่อยู่ใน CASE นั้น แต่ถ้าหากไม่เท่ากับ CASE ใดเลยก็จะตรวจสอบ CASE ใหม่ หรือก็ไม่ทำกิจกรรมใดเลย

ตัวอย่างที่ 8.6 การจ่ายค่าจ้างพนักงานของร้านค้าแห่งหนึ่ง เมื่อถึงวันจ่ายค่าแรงฝ่ายการเงินจะคีย์รหัสของพนักงานเข้าไป โดยรหัสและวิธีการจ่ายค่าแรงเป็นดังนี้

รหัส	**อัตราค่าแรง**
H	จ่ายเป็นอัตราชั่วโมง
P	จ่ายเป็นรายการที่ขายได้
C	จ่ายเป็นอัตราค่า Commission
S	จ่ายเป็นเงินเดือน

ถ้าให้ Rate เป็นค่าจ้างรายชั่วโมงและตามรายการที่ขายได้
Sales เป็นอัตราที่ขายตาม Commission
Salary เป็นอัตราเงินเดือน
จงเขียนซูโดโค้ดและผังงานของส่วนการคำนวณอัตราค่าจ้างพนักงาน

วิธีทำ กรณีนี้เป็นการเลือกทำแบบหลายทาง สามารถนำคำว่า CASE มาใช้ได้ โดยได้ดังนี้

```
1.  CASE   code    OF
           "H"     :     Pay = Rate * Hours
           "P"     :     Pay = Rate * Pieces
```

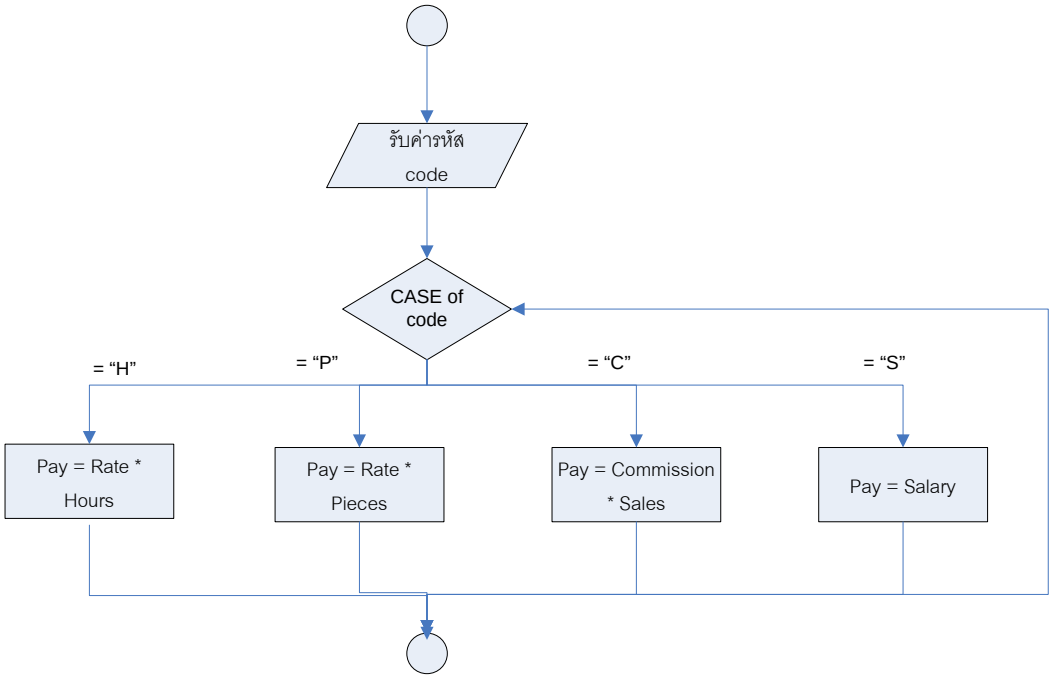
“C” : Pay = Commission

“S” : Pay = Salary

ENDCASE

2. EXIT

โดย code เป็นรหัสที่ป้อนเข้าไป ถ้าหากเท่ากับค่าคงที่ที่เป็นรหัสตัวใดก็จะไปคำนวณค่าจ้างตามที่กำหนดหลังค่าคงที่นั้น ถ้าหากเขียนเป็นผังงานจะได้ดังรูปที่ 8.13



รูปที่ 8.13 ผังงานส่วนของการคำนวณค่าจ้างพนักงาน

ขั้นตอนต่อไปถ้าหากต้องการพัฒนาเป็นโปรแกรมจะต้องทราบว่าโปรแกรมภาษาที่ใช้ จะใช้คำสั่งหรือฟังก์ชันใดมาใช้กับกรณีการเลือกทำแบบนี้ สำหรับการเขียนโปรแกรมด้วยภาษาซี การเลือกทำในลักษณะนี้จะใช้คำสั่ง switch .. case ซึ่งมีรูปแบบคำสั่งดังนี้

```
switch  (variable)
{
        case  constant_1   :        statement;
                                    break;
        case  constant_2   :        statement;
                                    break;
        case  constant_3   :        statement;
                                    break;
        …….                         …….
```

```
        case constant_n  : statement;
                                    break;
        default  :  statement;
}
```

คำสั่ง switch นี้จะนำค่าใน variable มาตรวจสอบว่าเท่ากับค่าคงที่ค่าใดหลัง case จากนั้นโปรแกรมจะไปทำ statement หลังค่าคงที่ตัวนั้น และออกจาก switch เมื่อถึงคำสั่ง break แต่ถ้าไม่เท่าค่าคงที่ค่าใดเลยโปรแกรมจะไปทำ statement หลัง default สำหรับค่าที่ใช้ตรวจสอบจะเป็นตัวแปร นิพจน์ หรือฟังก์ชั่นก็ได้ สำหรับในแต่ละ case สามารถมีคำสั่งได้มากกว่าหนึ่งคำสั่งหรืออาจไม่มีก็ได้ โดยถ้าไม่มีคำสั่งโปรแกรมจะไปทำงานใน case ถัดไป **<u>และค่าคงที่หลัง case จะต้องเป็น int หรือ char เท่านั้น</u>**

NOTE

การใช้คำสั่ง switch มีสิ่งสำคัญที่ควรรู้ 4 ประการคือ

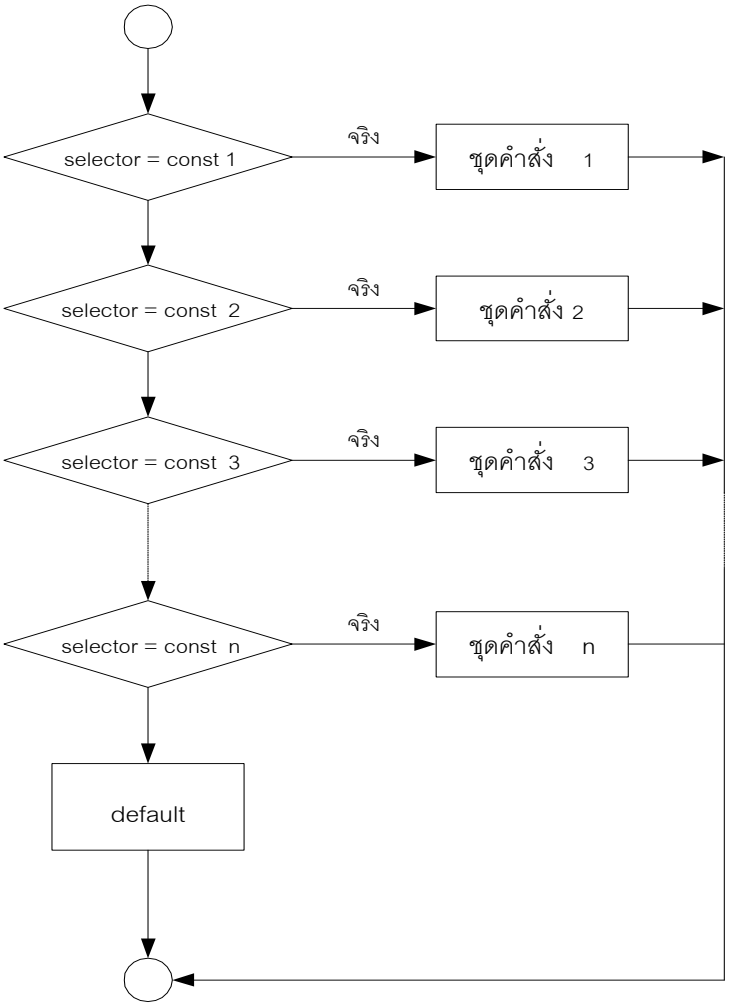
1. คำสั่ง switch ต่างจากคำสั่ง if ตรงที่ switch สามารถทดสอบเงื่อนไขได้หลายอย่าง แต่คำสั่ง if จะตรวจสอบเฉพาะความสัมพันธ์ หรือลอจิกเท่านั้น
2. ค่าคงที่ของคำสั่ง switch สามารถมีค่าซ้ำกันได้
3. ถ้าค่าคงที่เป็นตัวอักษร คำสั่ง switch จะมองเป็นเลขจำนวนเต็ม
4. ค่า default จะมีหรือไม่มีก็ได้

ตัวอย่างเช่นการเขียนคำสั่งต่อไปนี้

```
switch  (year)
{
        case    1 :     printf("Fershman\n");
                        break;
        case    2 :     printf("Sophomore\n");
                        break;
        case    3 :     printf("Junior \n");
                        break;
        case    4 :     printf("Senior\n");
                        break;
        default :       printf("Nonmatriculated\n");
}
```

จากชุดคำสั่งที่ผ่านมา ถ้าหากค่าใน year มีค่าเท่ากับ 1 คอมพิวเตอร์จะพิมพ์คำว่า Freshman ถ้าหากมีค่าเท่ากับ 3 จะพิมพ์คำว่า Junior แต่ถ้าหากไม่เท่ากับ 1,2,3,4 จะพิมพ์คำว่าNonmatriculated เราอาจสรุปได้ว่าคำสั่ง switch นี้จะนำค่าในตัวแปร selector ไปเปรียบเทียบกับค่าคงที่ค่าต่าง ๆ ถ้าเท่ากับค่าคงที่ค่าใดโปรแกรมจะไปทำคำสั่งหรือชุดคำสั่งที่อยู่หลังค่าคงที่นั้น แต่ถ้าไม่เท่ากับค่าคงที่ค่าใดเลย โปรแกรมจะทำคำสั่งที่อยู่ต่อจาก default

สำหรับตัวแปรที่ใช้เลือกทำที่อยู่ตามหลัง switch จะต้องเป็นตัวแปรประเภทลำดับ ซึ่งจะทำให้คอมพิวเตอร์สามารถเดาค่าได้ และค่าคงที่ต้องเป็นตัวแปรประเภทเดียวกับตัวแปรที่ตามหลัง case การทำงานของคำสั่ง case อาจจะเขียนเป็นผังงานได้ดังนี้



รูปที่ 8.14 การทำงานของคำสั่ง switch..case

ตัวอย่าง

ถ้าหากตัวแปร num เป็นเลขจำนวนเต็ม การกำหนดค่าหลัง case ในคำสั่ง switch เช่น

```
switch(num)
{
        case  4          : ชุดคำสั่ง;  break;        /* ถูกต้องเพราะเป็นตัวเลข */
        case  2.5        : ชุดคำสั่ง;  break;        /*ไม่ถูกต้องเพราะเป็นทศนิยม */
        case  m          : ชุดคำสั่ง;  break;        /*ไม่ถูกต้องเพราะเป็นตัวแปร */
        case  '2'        : ชุดคำสั่ง;  break;        /*ไม่ถูกต้องเพราะเป็นตัวอักขระ */
        default          : ชุดคำสั่ง
}
```

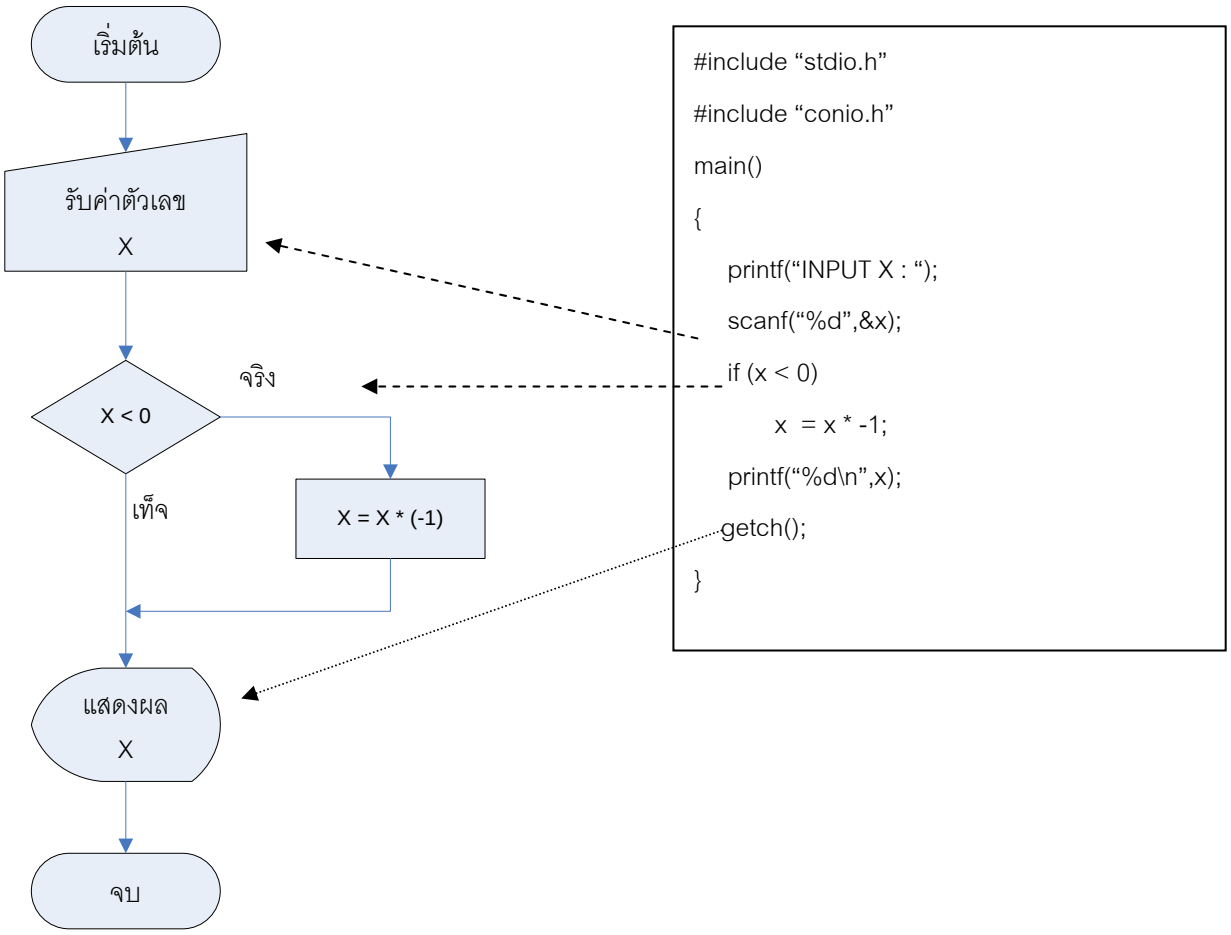
8.5 กรณีศึกษาการทำงานแบบมีแบบมีทางเลือก

สำหรับในหัวข้อนี้จะแสดงตัวอย่างงานที่มีการทำงานแบบเลือกทำเพื่อเป็นแนวทางในการเขียนโปรแกรมต่อไป

โปรแกรมที่ 8.4 จงเขียนโปรแกรมรับเลขจำนวนเต็มทางแล้วให้คอมพิวเตอร์แสดงค่าสัมบูรณ์ของเลขนั้นออกมา

วิธีทำ จากการวิเคราะห์ปัญหาของโจทย์จะพบว่าข้อมูลอินพุตคือตัวเลขจำนวนเต็ม เอาต์พุตก็เป็นเลขจำนวนเต็มที่เป็นค่าสัมบูรณ์ของเลขนั้น สำหรับวิธีการประมวลผลทำโดยตรวจสอบว่าเลขที่รับเข้ามานั้นมีค่ามากกว่า 0 หรือไม่ ถ้ามากกว่าให้แสดงเลขนั้น ถ้าน้อยกว่าให้ทำเป็นค่าบวกโดยนำค่า -1 ไปคูณ

จากปัญหานี้จะเป็นการเลือกทำแบบทางเดียว ซึ่งสามารถเขียนผังงานและโปรแกรมภาษาซีได้ดังรูปที่ 8.15



รูปที่ 8.15 ผังงานและโปรแกรมหาค่าสัมบูรณ์ของตัวเลข

โปรแกรมที่ 8.5 จงเขียนโปรแกรมรับตัวเลขสองค่า จากนั้นให้แสดงค่าที่มากกว่าทางจอภาพ

วิธีทำ อินพุต รับค่าตัวเลขสองค่า A และ B

เอาต์พุต แสดงค่าตัวเลขที่มากกว่าทางจอภาพ

วิธีการประมวลผล นำตัวเลข A และ B มาเปรียบเทียบ

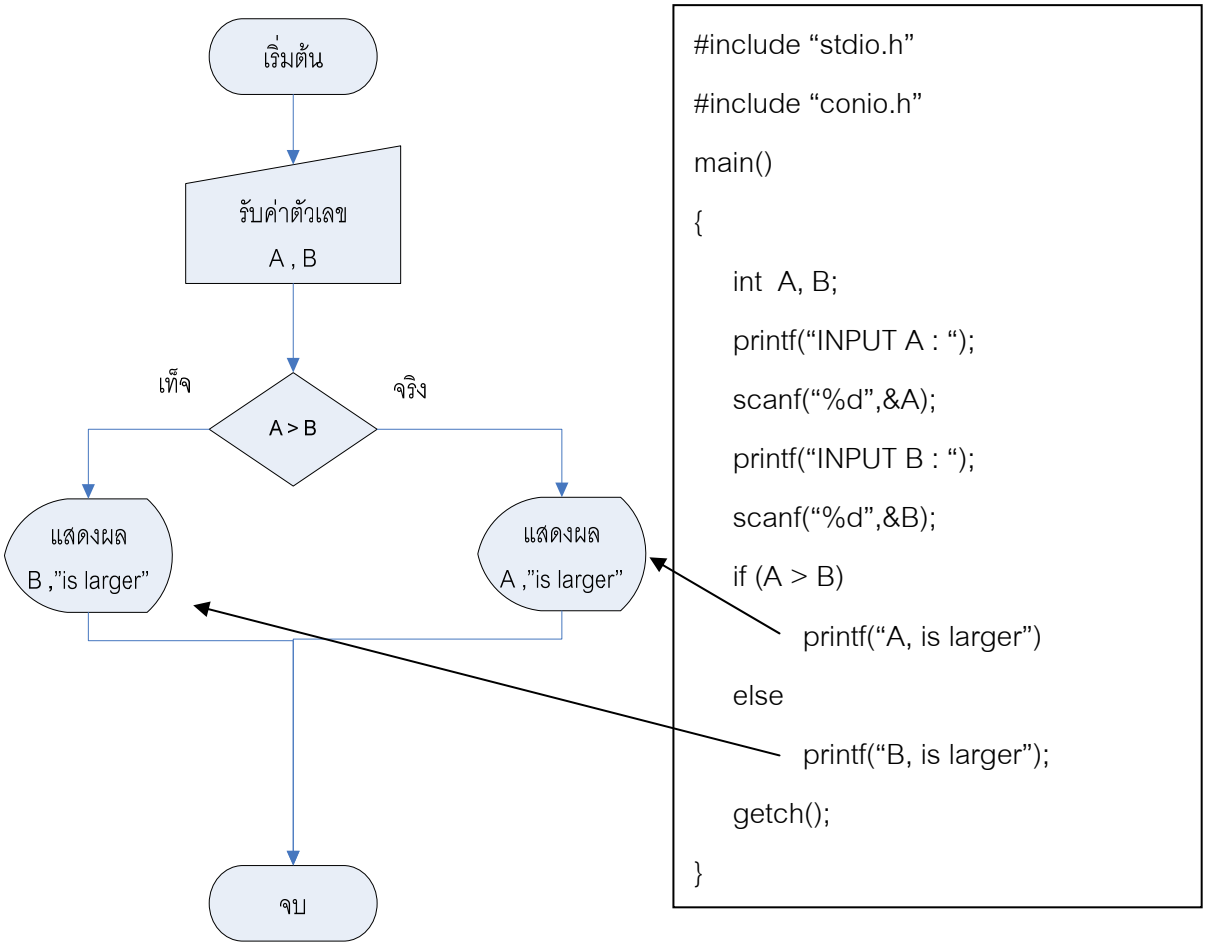
ถ้า A > B แล้ว

แสดงผลค่า A

มิฉะนั้น

แสดงผลค่า B

จากปัญหานี้จะเป็นการเลือกทำแบบสองทิศทาง การตรวจสอบเงื่อนไขทำโดยนำตัวเลขสองตัวมาเปรียบเทียบกัน สามารถเขียนเป็นผังงานและโปรแกรมได้ดังรูปที่ 8.16

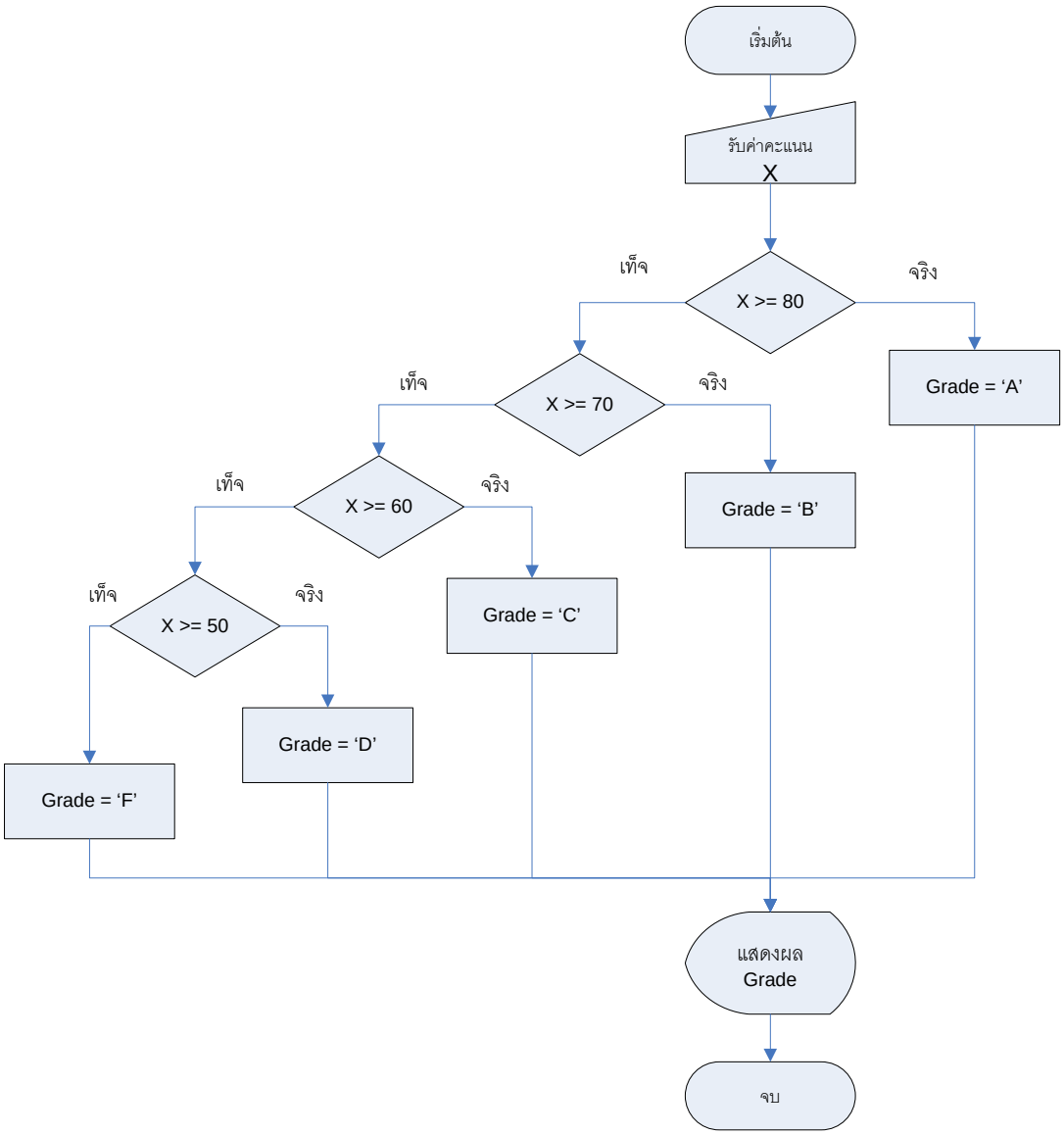


รูปที่ 8.16 ผังงานและโปรแกรมหาเลขสูงสุด

โปรแกรมที่ 8.6 จงเขียนผังงานและโปรแกรมตัดเกรดของนักเรียนโดยให้ป้อนคะแนนสอบเข้าไปจากนั้นให้แสดงเกรดออกมา โดยเงื่อนไขของเกณฑ์การตัดเกรดเป็นดังนี้

คะแนนตั้งแต่ 80 ขึ้นไป ได้เกรด “A”
คะแนนตั้งแต่ 70 ขึ้นไป ได้เกรด “B”
คะแนนตั้งแต่ 60 ขึ้นไป ได้เกรด “C”
คะแนนตั้งแต่ 50 ขึ้นไป ได้เกรด “D”
คะแนนต่ำกว่า 50 ได้เกรด “F”

วิธีทำ จากปัญหานี้สามารถนำการตรวจสอบเงื่อนไขแบบสองทางมาซ้อนกันให้เป็นการเลือกทำแบบหลายทางได้ ซึ่งเขียนเป็นผังงานได้ดังรูปที่ 8.17



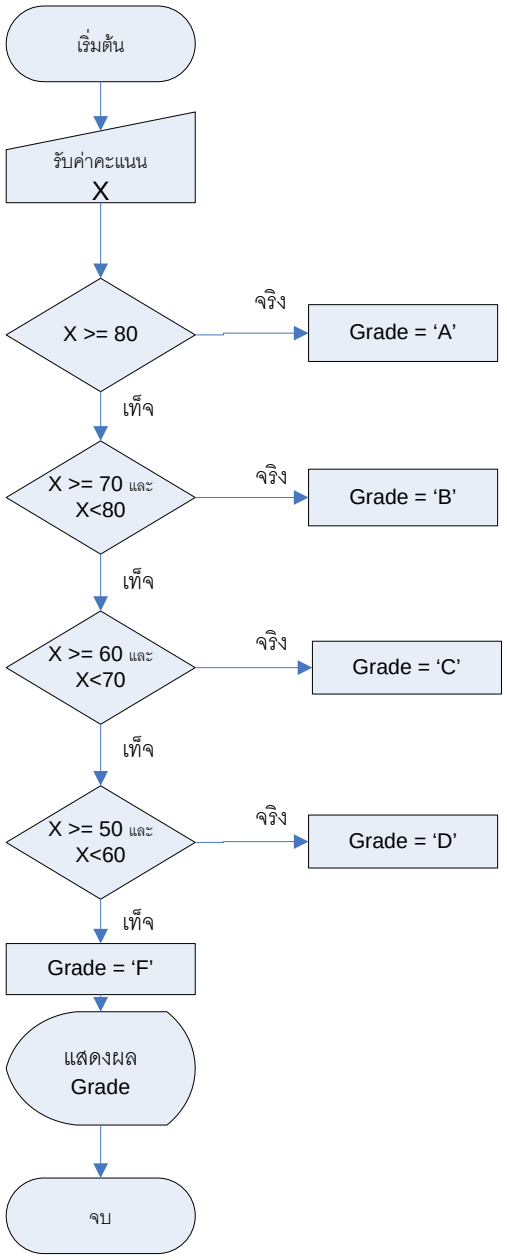
รูปที่ 8.17 ผังงานของโปรแกรมตัดเกรดโดยการนำการเลือกทำแบบสองทางมาซ้อนกัน

การเลือกทำในลักษณะนี้สามารถนำการเลือกทำแบบทางเดียวมาใช้ได้ แต่ต้องเปลี่ยนเงื่อนไขของการตัดสินใจเป็นการประมวลผลในลักษณะต่อไปนี้

ถ้าคะแนนตั้งแต่ 80 ขึ้นไป ได้เกรด “A”
ถ้าคะแนนตั้งแต่ 70 ขึ้นไป แต่น้อยกว่า 80 ได้เกรด “B”
ถ้าคะแนนตั้งแต่ 60 ขึ้นไป แต่น้อยกว่า 70 ได้เกรด “C”
ถ้าคะแนนตั้งแต่ 50 ขึ้นไป แต่น้อยกว่า 60 ได้เกรด “D”
ถ้าคะแนนไม่ถึง 50 ได้เกรด “F”

ดังนั้นสามารถนำการเลือกทำแบบทางเดียวมาเขียนผังงานและโปรแกรมได้ดังรูปที่ 8.18

```
#include <stdio.h>
#include <conio.h>
main()
 {
      int  x;
      char  Grade;
      printf(“Input Score “);
      scanf(“%d”,&x);
      if (x >= 80)
          Grade = 'A';
     if ((x >= 70)&&(X < 80))
          Grade = 'B';
    if ((x >= 60)&&(x < 70))
          Grade = 'C';
    if ((x >= 50)&&(X < 60))
          Grade = 'D';
    if (X < 50)
          Grade = 'F';
    printf(“Grade = %c”,Grade);
    getch();
}
```



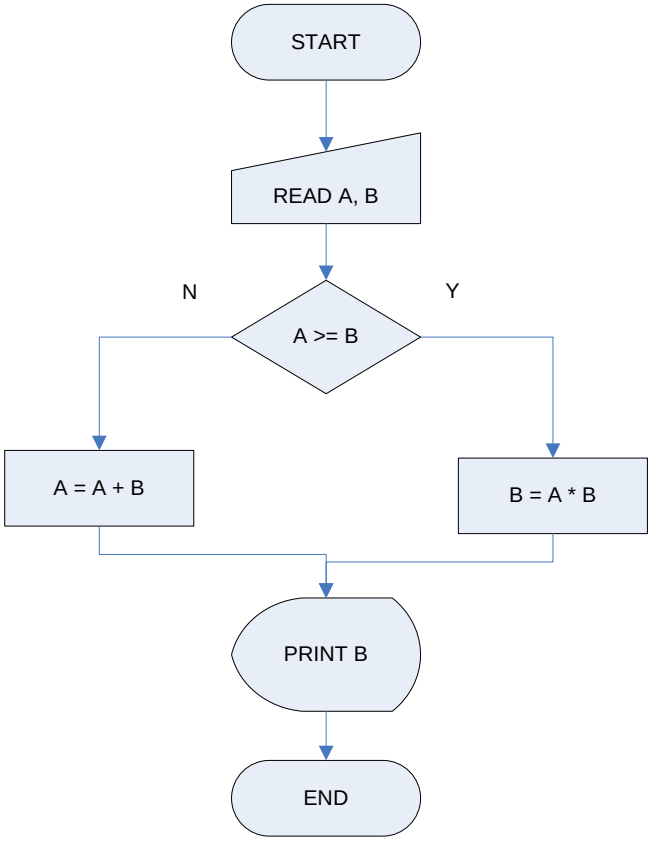
รูปที่ 8.18 การตัดเกรดโดยใช้การเลือกทำแบบทางเดียว

แบบฝึกหัดท้ายบท

ตอนที่ 1 จงเลือกคำตอบที่ถูกต้องที่สุดเพียงหนึ่งข้อ

1. สัญลักษณ์ของผังงานในข้อใด จำเป็นจะต้องมีอยู่ในโครงสร้างผังงานแบบมีทางเลือก
   ก. การรับข้อมูล ข. การตัดสินใจ
   ค. การแสดงผลข้อมูล ง. ถูกทุกข้อ
2. การอธิบายการทำงานของโปรแกรมที่มีทางเลือกจะใช้คำอธิบายว่าอะไร
   ก. ถ้า…แล้ว ข. เมื่อ…จะ
   ค. ถ้า…ดังนั้น ง. ทำ…ถ้า
3. ถ้าหากในซูโดโค้ดมีการใช้คำว่า IF แต่ไม่มีคำว่า ELSE ท่านคิดว่าเป็นการเลือกทำแบบใด
   ก. การเลือกว่าชุดคำสั่งที่ตามมาจะทำหรือไม่
   ข. การเลือกทำที่มีสองทางเลือก
   ค. การเลือกทำที่มีสามทางเลือก
   ง. ถูกทุกข้อ
4. สัญลักษณ์การตัดสินใจที่นำมาใช้กับการทำงานแบบมีทางเลือกจะมีลักษณะใด
   ก. มีลูกศรชี้เข้าสองทิศทาง ชี้ออกทางเดียว
   ข. มีลูกศรชี้เข้าสองทิศทาง ชี้ออกสองทิศทาง
   ค. มีลูกศรชี้เข้าทิศทางเดียว ชี้ออกทิศทางเดียว
   ง. มีลูกศรชี้เข้าทิศทางเดียว ชี้ออกสองทิศทาง
5. ในการเขียนซูโดโค้ดแบบการทำงานที่มีทางเลือก ขั้นตอนวิธีที่อยู่ต่อจากคำว่า IF คือขั้นตอนแบบใด
   ก. การตรวจสอบเงื่อนไข
   ข. การคำนวณ
   ค. การรับค่าอินพุต
   ง. การพิสูจน์
6. ข้อใดถูกต้องสำหรับการเขียนซูโดโค้ดสำหรับงานที่มีหลายทางเลือก
   ก. ใช้คำว่า “IF” ,”THEN”,”ELSE มาซ้อน ๆ กัน
   ข. คำนำว่า “CASE” มาใช้
   ค. เขียน “IF”,”THEN” หลาย ๆ ครั้ง
   ง. ถูกทุกข้อ

จากผังงานต่อไปนี้ใช้ตอบคำถามข้อ 7 ถึงข้อ 10



7. จากผังงานดังรูปในส่วนของการทำงานแบบมีทางเลือกจะเป็นการทำงานลักษณะใด
    ก. การเลือกว่าการประมวลผลที่ตามมาจะทำหรือไม่
    ข. การเลือกทำแบบสองทางเลือก
    ค. การเลือกทำถ้าผลลัพธ์เป็นเท็จ
    ง. การเลือกทำถ้าผลลัพธ์เป็นจริง
8. ถ้าหากเงื่อนไขในการตรวจสอบเงื่อนไขเป็นจริง การทำงานจะเป็นในลักษณะใด
    ก. คำนวณค่า B เท่ากับ A คูณกับ B
    ข. คำนวณค่า A เท่ากับ A บวกกับ B
    ค. แสดงผลค่าในตัวแปร B
    ง. มีข้อถูกมากกว่าหนึ่งข้อ
9. ถ้าหากรับค่า A เท่ากับ 3 และรับค่า B เท่ากับ 2 ข้อใดถูกต้อง
    ก. การตรวจสอบเงื่อนไขจะเป็นจริง
    ข. ตัวแปร A มีค่าเท่ากับ 5
    ค. ระบบจะแสดงผลค่า 2
    ง. ถูกทุกข้อ

10. จากข้อ 9 การแสดงผลจะเป็นอย่างไร
    ก. แสดงผลค่า 2 ออกทางจอภาพ
    ข. แสดงผลค่า 2 ออกทางเครื่องพิมพ์
    ค. แสดงผลค่า 6 ออกทางจอภาพ
    ง. แสดงผลคำตอบโดยไม่ระบุประเภทเอาต์พุต

ตอนที่ 2 จงทำเครื่องหมาย √ หน้าข้อที่ถูก และเครื่องหมาย × หน้าข้อที่ผิด

……. 1. โครงสร้างการทำงานแบบมีทางเลือกจะต้องมีการตรวจสอบเงื่อนไขเสมอ

……. 2. การทำงานแบบมีทางเลือก เงื่อนไขจะเป็นได้ทั้งจริง เท็จ หรือมีทั้งสองกรณี

……. 3. การทำงานแบบมีทางเลือกระบบจะทำเงื่อนไขที่เป็นจริงก่อนแล้วจึงทำเงื่อนไขที่เป็นเท็จ

……. 4. การเขียนผังงานแบบมีทางเลือก ไม่จำเป็นต้องมีขั้นตอนการทำงานครบทั้งสองกรณี

……. 5. การเขียนคำอธิบายการทำงานแบบมีทางเลือกจะต้องมีคำว่า “และ” ประกอบอยู่

ตอนที่ 3 จงตอบคำถามต่อไปนี้

1. การทำงานแบบมีทางเลือกมีลักษณะเป็นอย่างไร

……

……

…….

2. จงเขียนผังงานตามคำอธิบายโปรแกรมต่อไปนี้

เริ่มต้น

รับค่า X

รับค่า Y

ถ้า Y มากกว่า X แล้ว

คำนวณค่า M มีค่าเท่ากับ X ยกกำลังสอง บวกกับ Y ยกกำลังสอง

มิฉะนั้น

คำนวณค่า M มีค่าเท่ากับ X บวกกับ Y

แสดงค่าของ M ทางจอภาพ

จบ

3. จากข้อ 2 ถ้าหาก X เท่ากับ 7 และ Y เท่ากับ 3 จอภาพจะแสดงผลค่าใด

……

……

4. จงเขียนคำอธิบายการทำงาน และผังงานสำหรับคำนวณเกรดของนักศึกษา โดยให้ป้อนคะแนนการบ้าน คะแนนสอบกลางภาค และคะแนนสอบปลายภาคเข้าไป แล้วให้ระบบคำนวณคะแนนรวมและเกรดพร้อมทั้งแสดงออกทางจอภาพ โดยเงื่อนไขของเกรดเป็นดังนี้

ได้คะแนน 0 ถึง 49	ได้เกรด F
ได้คะแนน 50 ถึง 59	ได้เกรด D
ได้คะแนน 60 ถึง 69	ได้เกรด C
ได้คะแนน 70 ถึง 79	ได้เกรด B
ได้คะแนน 80 ถึง 100	ได้เกรด A

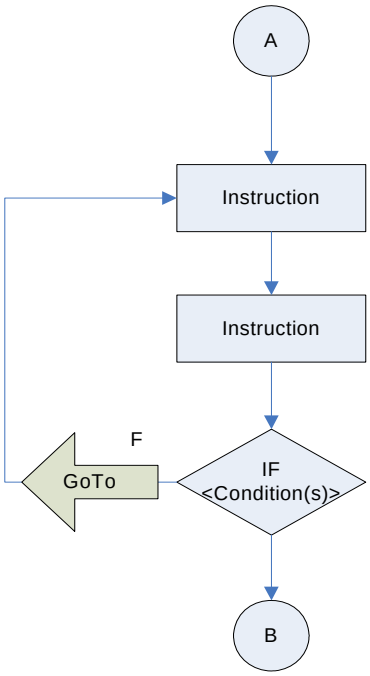
บทที่ 9
การทำงานแบบมีทำซ้ำ

การวนซ้ำหรือการทำเป็นวงรอบคือการทำงานอย่างเดียวกันหรือซ้ำ ๆ กันตามเงื่อนไขที่กำหนด ซึ่งการทำงานแบบนี้คอมพิวเตอร์สามารถทำได้เป็นอย่างดีและมีความถูกต้องมากกว่ามนุษย์ ในโปรแกรมหนึ่ง ๆ อาจมีชุดคำสั่งที่ต้องทำซ้ำบ่อย ๆ การออกแบบคอมพิวเตอร์ให้ทำงานอย่างนี้ถือว่ามีความจำเป็นสำหรับการเขียนโปรแกรม ในบทนี้จะกล่าวถึงลักษณะการทำซ้ำ การเขียนคำอธิบายการทำงาน การเขียนซูโดโค้ดและการเขียนผังงานของงานที่มีการทำซ้ำ

9.1 ประเภทของการทำซ้ำและคำสั่งภาษาซี

การทำงานแบบวนซ้ำนั้นจะต้องมีการตรวจสอบเงื่อนไขอยู่ภายในลูปของการทำซ้ำด้วย งานบางประเภทมีจำนวนครั้งในการทำซ้ำที่แน่นอน งานบางประเภทมีจำนวนครั้งในการทำซ้ำไม่แน่นอน ขึ้นอยู่กับเงื่อนไขที่เขียนในโปรแกรม ประเภทของการทำซ้ำจะแบ่งได้เป็น 3 ประเภทดังนี้

1. การทำซ้ำแบบที่ทราบจำนวนครั้งในการทำซ้ำ
2. การทำซ้ำจนระบบมีเงื่อนไขอย่างหนึ่งจึงหยุด
3. การทำซ้ำแบบถ้าเงื่อนไขเป็นจริงจะทำชุดคำสั่ง



รูปที่ 9.1 ลักษณะของการทำคำสั่งซ้ำ ๆ

การทำซ้ำแบบแรกนั้นมักจะใช้กับงานที่ทราบจำนวนครั้งในการทำซ้ำที่แน่นอน โดยในลูปจะมีตัวแปรสำหรับควบคุมการนับลูปอยู่ภายใน ส่วนการทำซ้ำแบบทำซ้ำจนระบบมีเงื่อนไขอย่างหนึ่งจึงหยุด เมื่อโปรแกรมเข้าสู่ลูปการทำซ้ำจะมีการตรวจสอบเงื่อนไขที่ด้านท้ายของลูปถ้าหากเงื่อนไขเป็นจริงก็จะออกจากลูปการทำซ้ำ ส่วนการทำซ้ำแบบถ้าเงื่อนไขเป็นจริงจะทำชุดคำสั่ง จะมีการตรวจสอบเงื่อนไขก่อนการทำซ้ำถ้าเงื่อนไขเป็นจริงจะทำชุดคำสั่งภายใน แต่ถ้าเงื่อนไขเป็นเท็จจะออกจากลูปการทำซ้ำ

คำสั่งการทำลูปในภาษาซี

การทำงานซ้ำหรือการทำลูปทั้ง 3 ประเภทที่กล่าวมานั้นในภาษาซีจะมีคำสั่งอยู่ 3 ประเภท ดังนี้

- **คำสั่ง for**

คำสั่งนี้จะทำซ้ำในลูปโดยมีการกำหนดค่าเริ่มต้น แล้วตรวจสอบเงื่อนไข ถ้าเงื่อนไขเป็นจริงจะทำในลูปซ้ำจนกว่าเงื่อนไขจะเป็นเท็จ โดยทั่วไปจะใช้คำสั่งนี้ในการทำงานซ้ำ ๆ ที่ทราบจำนวนครั้งในการทำซ้ำแน่นอน รูปแบบของคำสั่งเป็นดังนี้

```
for (initialization; condition; increment or decrement)
{
        คำสั่งต่าง ๆ ในลูป
}
```

โดยที่	initialization	เป็นการกำหนดค่าเริ่มต้นให้กับตัวแปรที่ใช้ตรวจสอบเงื่อนไข
	condition	เป็นนิพจน์ที่ใช้ตรวจสอบเงื่อนไข
	increment	เพิ่มค่าให้กับตัวแปร
	decrement	ลดค่าให้กับตัวแปร

ตัวอย่าง พิจารณาส่วนของการเขียนโปรแกรมต่อไปนี้

```
int  i;              /*ประกาศตัวแปร I สำหรับเก็บเลขจำนวนเต็ม */
for ( i = 1; i <= 5; i++)
        printf("%d  ",i);
```

เมื่อโปรแกรมทำงานจะกำหนดให้ i มีค่าเริ่มต้นเป็น 1 จากนั้นจะตรวจสอบว่า i มีค่าน้อยกว่าหรือเท่ากับ 5 จริงหรือไม่ ถ้าจริงจะทำงานในลูป คือพิมพ์ค่าตัวแปร i จากนั้นจะเพิ่มค่า I ขึ้นหนึ่งค่า แล้วกลับมาตรวจสอบเงื่อนไขอีกครั้งหนึ่ง ถ้าหากเงื่อนไขเป็นเท็จจะออกนอกลูป ทำให้การทำงานของคำสั่งเป็นการพิมพ์ตัวเลข 1 2 3 4 5 ออกมาทางจอภาพ

- **คำสั่ง do-while**

เป็นคำสั่งที่ใช้ในการวนซ้ำที่มีจำนวนครั้งไม่แน่นอน เมื่อโปรแกรมเข้าสู่ลูปจะทำคำสั่งต่าง ๆ ในลูปก่อนหนึ่งครั้ง จากนั้นจะมีการตรวจสอบเงื่อนไขอยู่ด้านท้ายลูป ถ้าหากตรวจสอบเงื่อนไขแล้วเป็นจริงโปรแกรมจะทำในลูปต่อไป แต่ถ้าเงื่อนไขเป็นเท็จจะออกนอกลูป รูปแบบของคำสั่งเป็นดังนี้

```
do {
        คำสั่งต่าง ๆ ในลูป
}while (condition)
```

โดยที่	condition	เป็นการตรวจสอบเงื่อนไขท้ายลูป ถ้าหากเป็นเท็จจะออกจากลูป

- **คำสั่ง while**

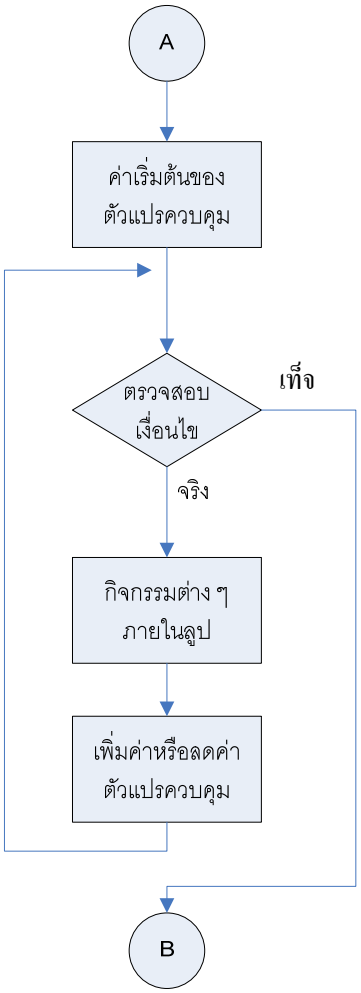
เป็นคำสั่งที่จะตรวจสอบเงื่อนไขก่อนที่จะทำซ้ำ ถ้าหากเงื่อนไขเป็นจริงจะทำซ้ำในลูป ถ้าหากเงื่อนไขเป็นเท็จจะออกนอกลูป การใช้คำสั่งนี้โปรแกรมอาจไม่ทำในลูปเลยก็ได้ถ้าหากเงื่อนไขเป็นเท็จ รูปแบบของคำสั่งเป็นดังนี้

```
while (condition)
{
        คำสั่งต่าง ๆ ในลูป
}
```

โดยที่ condition เป็นการตรวจสอบเงื่อนไขก่อนการทำงานในลูป

9.2 การทำซ้ำแบบทราบจำนวนครั้งในการทำซ้ำ

การทำซ้ำในแบบนี้จะเริ่มต้นด้วยการกำหนดค่าเริ่มต้นให้กับตัวแปรที่ทำหน้าที่เป็นตัวควบคุมลูป จากนั้นจะตรวจสอบเงื่อนไขว่าเป็นจริงหรือเท็จ ถ้าเป็นจริงจะทำกิจกรรมต่าง ๆ ที่อยู่ในลูป แต่ถ้าเป็นเท็จจะออกนอกลูป โดยภายในลูปนั้นจะมีการเพิ่มค่าหรือลดค่าตัวควบคุมลูปด้วย ดังตัวอย่างในรูปที่ 9.2



ตัวอย่างคำอธิบายการพิมพ์ค่าตัวเลข 1 ถึง 10

1. เริ่มต้น
2. กำหนดให้ X = 1
3. ในขณะที่ X น้อยกว่าหรือเท่ากับ 10 ทำ
   พิมพ์ค่า X ทางจอภาพ
   เพิ่มค่า X ขึ้นหนึ่งค่า
4. ทำคำสั่งต่อไป

รูปที่ 9.2 ผังงานและตัวอย่างลักษณะการทำซ้ำที่มีตัวควบคุม

ภายในลูปของการทำซ้ำจะประกอบด้วยชุดคำสั่งที่ต่อกันแบบลำดับก็ได้ หรือจะมีหลาย ๆ คำสั่งก็ได้

การทำซ้ำประเภทนี้การอธิบายการทำงานจะใช้คำว่า “ในขณะที่” และคำว่า “ทำ” เป็นคำที่ใช้อธิบายการทำงาน โดยมีการตรวจสอบเงื่อนไขอยู่ระหว่างคำว่า “ในขณะที่” กับคำว่า “ทำ” ดังตัวอย่างในรูปที่ 10.2

สำหรับการเขียนซูโดโค้ดจะใช้คำว่า “FOR”, “DO” และ “ENDFOR” โดยมีรูปแบบดังนี้

```
FOR     กำหนดรอบการทำซ้ำ
        ชุดคำสั่งที่ต้องการทำซ้ำ
ENDFOR
```

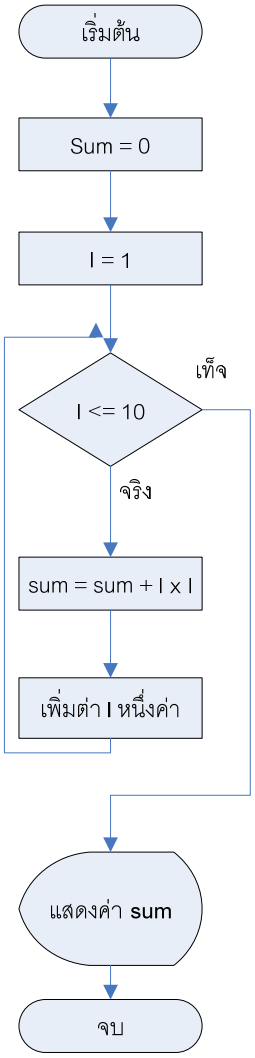
ตัวอย่างที่ 9.1 จงเขียนผังงานสำหรับหาผลบวกของเลขยกกำลังสอง ถ้าหากตัวเลขเป็นเลขจำนวนเต็มระหว่าง 1 ถึง 10 แล้วแสดงผลบวกออกทางจอภาพ

วิธีทำ งานในลักษณะนี้จะทำจำนวน 10 ครั้ง ดังนั้นจะกำหนดให้

sum เป็นตัวแปรสำหรับเก็บผลรวม

I เป็นตัวแปรที่ใช้นับลูป

การทำงานของโปรแกรมจะเขียนได้ดังนี้



1. เริ่มต้น
2. ให้ sum = 0
3. ให้ I = 1
4. ในขณะที่ I น้อยกว่าหรือเท่ากับ 10 ทำ
   4.1 sum เท่ากับ sum บวก I คูณ I
   4.2 เพิ่ม I ขึ้นหนึ่งค่า
5. แสดงผล sum
6. จบ

รูปที่ 9.3 ผังงานและคำอธิบายการทำงานของตัวอย่างที่ 9.1

จากตัวอย่างที่ 9.1 จะพบว่าการหาผลบวกนั้นจะต้องกำหนดค่าเริ่มต้นสำหรับตัวแปรเก็บผลบวก (Sum) ให้มีค่าเป็นศูนย์เสียก่อน จากนั้นในลูปจะนำค่าจำนวนนับ (I) มายกกำลังสองแล้วเอามาบวกกับตัวแปร sum นี้ โดยให้ตัวแปร I เป็นตัวแปรควบคุมลูปไปด้วย สำหรับการทำลูปแต่ละครั้งจะตรวจสอบดูว่าค่าตัวแปร I มีค่าเกิน 10 หรือยัง ถ้าหากเกิน 10 แล้วให้ออกจากลูปแล้วแสดงผล

โปรแกรมที่ 9.1 ถ้าหากต้องการรับตัวเลขจำนวนเต็มระหว่าง 0 ถึง 100 เข้าไปจำนวน 10 ค่า และให้ระบบแสดงค่าสูงสุดและค่าต่ำสุดออกมาทางจอภาพ จงวิเคราะห์งานของปัญหานี้ พร้อมทั้งเขียนผังงานและตัวอย่างโปรแกรมในภาษาซี

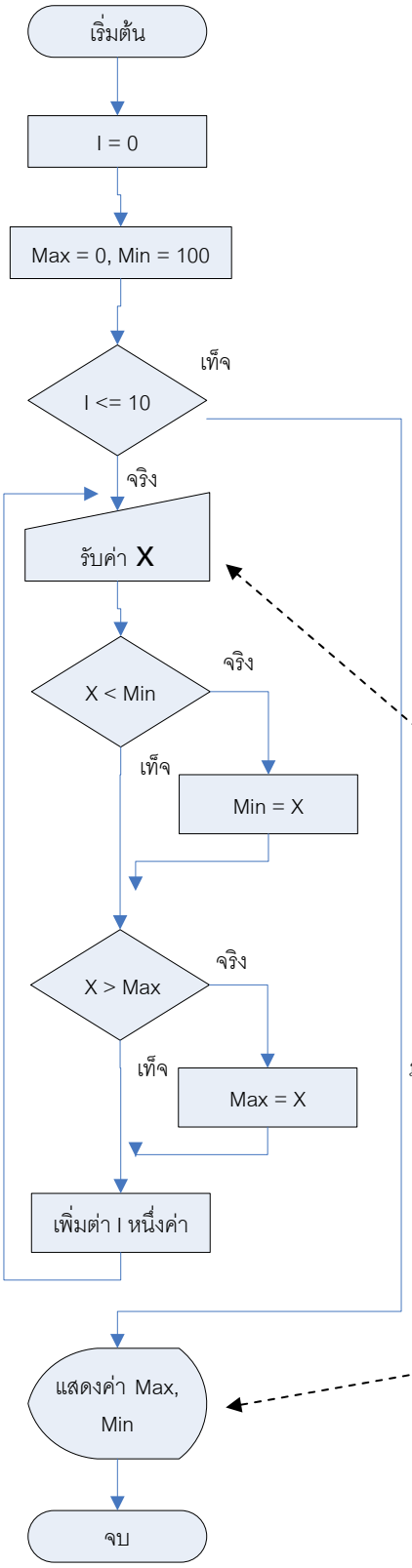
วิธีทำ

ต้องการอะไร	ค่าสูงสุด (Max), ค่าต่ำสุด (Min)
ต้องการเอาต์พุตอย่างไร	แสดงค่า Max และ Mix ทางจอภาพ
ข้อมูลเข้า	รับข้อมูลเลขจำนวนเต็ม 10 ค่าทางแป้นพิมพ์
วิธีการประมวลผล	ให้ตัวแปร I เป็นตัวนับจำนวนข้อมูล 10 ค่า กำหนดให้ Max = 0 และ Min = 100 ให้ตัวแปร x เป็นค่าข้อมูลที่รับเข้าทางแป้นพิมพ์ แล้วให้เปรียบเทียบว่าค่านี้ควรเป็น Max หรือ Min

จากลักษณะของงานจะต้องมีการใช้คำสั่งเลือกทำอยู่ในลูปด้วย เพื่อตรวจสอบว่าค่า x ที่รับเข้ามาเป็นค่าสูงสุดหรือต่ำสุด ซึ่งสามารถนำการเลือกทำแบบทางเดียวมาใช้ได้ ดังนั้นสามารถเขียนคำอธิบายการทำขั้นตอนการทำงานได้ดังนี้

1. เริ่มต้น
2. กำหนดให้ I = 1
3. กำหนดให้ Max = 0, Min = 100;
4. ในขณะที่ I น้อยกว่าหรือเท่ากับ 10 ทำ
   - 4.1 รับค่า x
   - 4.2 ถ้า x น้อยกว่า Min แล้วทำ
     - Min = x
   - 4.3 ถ้า x มากกว่า Max แล้วทำ
     - Max = x
   - 4.4 เพิ่มค่าตัวนับ I

   การทำงานภายในลูป
5. แสดงผลค่า Max และ Min
6. จบการทำงาน



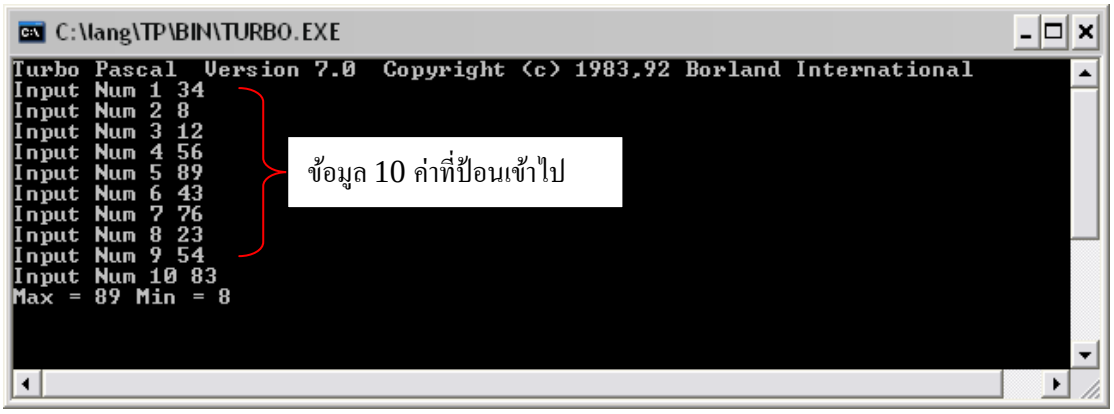
โปรแกรมภาษาซี

```
#include  “stdio.h”
#include  “conio.h”
main()
{
        int  I, X, Max, Min;
        Max = 0;
        Min = 100;
        for(I = 1; I <= 10; I++)
        {
                printf(“Input Num  %d “,I);
                scanf(“%d”,&X);
                if (X < Min)
                        Min = X;
                if (X > Max)
                        Max = X;
        }
        printf(“Max = %d, Min = %d\n”, Max, Min)
        getch();
}
```

ภายในลูป

รูปที่ 9.4 ผังงานและโปรแกรมของปัญหาการค่าค่าสูงสุดต่ำสุดจากข้อมูล 10 ค่า

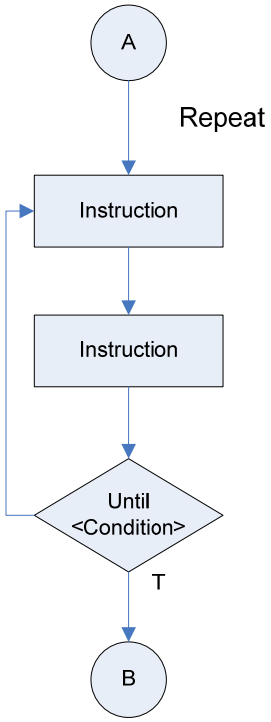
สำหรับตัวอย่างและผลลัพธ์ในการรันโปรแกรมจะได้ดังรูปต่อไปนี้



รูปแสดงตัวอย่างและผลลัพธ์ในการรันโปรแกรมในตัวอย่าง

9.3 การทำซ้ำจนระบบมีเงื่อนไขอย่างหนึ่งจึงหยุด

การทำซ้ำในลักษณะนี้เป็นการทำซ้ำที่มีจำนวนครั้งในการทำไม่แน่นอน เมื่อโปรแกรมเข้าสู่ลูปการทำซ้ำจะมีการตรวจสอบเงื่อนไขอยู่ด้านท้ายของลูป ถ้าหากตรวจสอบเงื่อนไขแล้วเป็นเท็จโปรแกรมจะกลับไปทำกิจกรรมในลูปอีกครั้งแต่ถ้าตรวจสอบเงื่อนไขแล้วเป็นจริงโปรแกรมจะออกนอกลูป ดังนั้นโปรแกรมจะต้องทำกิจกรรมต่าง ๆ ภายลูปหนึ่งครั้งเสมอ การทำงานในลักษณะนี้จะใช้คำอธิบายว่า “จนกระทั่ง” และตามด้วยเงื่อนไขของการตรวจสอบ ส่วนซูโดโค้ดที่ใช้ในการเขียนการทำซ้ำแบบนี้จะใช้คำว่า “REPEAT…UNTIL” หรือ “DO…UNTIL” ก็ได้ การทำซ้ำในลักษณะนี้เขียนเป็นผังงานได้ดังรูปที่ 9.5




```
REPEAT
        ……
        ……
    ชุดคำสั่งต่าง ๆ ภายในลูป
        ……
        ……
UNTIL (ตรวจสอบเงื่อนไข)
```

รูปที่ 9.5 ลักษณะการทำซ้ำที่ตรวจสอบเงื่อนไขหลังการทำลูป

การทำซ้ำลักษณะนี้ภาษาซีจะใช้คำสั่ง do..while ในการทำลูป โดยรูปแบบของคำสั่งเป็นดังนี้

```
do
{
        statement;
}while(condition);
```

ชุดคำสั่งต่าง ๆ ที่จะทำซ้ำจะอยู่ในเครื่องหมายปีกกาหลัง do ส่วน condition ที่อยู่หลัง while จะใช้สำหรับตรวจสอบเงื่อนไข ถ้าหากเงื่อนไขเป็นจริงจะทำในลูป ถ้าเป็นเท็จจะออกนอกลูป

ตัวอย่างที่ 9.2 พิจารณาการทำชุดคำสั่งต่อไปนี้

```
int  counter  =  1;              /*กำหนดตัวแปร counter เท่ากับ 1 */
do  {
        printf("%d  ",counter);
}while(++counter <= 10);    /*ทำในลูปถ้า counter น้อยกว่า 10 จริง */
```

จากชุดคำสั่งเริ่มแรกโปรแกรมจะพิมพ์ตัวเลข 1 ออกมา จากนั้นจะเพิ่มค่าตัวแปร counter ขึ้นหนึ่งค่าแล้วตรวจสอบว่าค่าตัวแปรนี้ยังน้อยกว่า 10 จริงอยู่หรือไม่ ถ้าจริงทำในลูปอีกครั้ง ถ้าเป็นเท็จจะออกนอกลูป ถ้าหากรันโปรแกรมผลลัพธ์ที่ได้จะเป็นดังนี้

```
1  2  3  4  5  6  7  8  9  10
```

ตัวอย่างที่ 9.3 ถ้าหากต้องการเขียนชุดคำสั่งวนลูปรับค่าตัวเลขที่มีค่าเกิน 100 ไปเรื่อย ๆ ถ้าหากตัวเลขที่ป้อนเข้าไปมาค่าน้อยกว่า 100 ให้ออกนอกลูปจะเขียนชุดคำสั่งได้ดังนี้

```
int  num;                          /* ประกาศตัวแปร num สำหรับเก็บข้อมูล */
do{
        printf("Input Number ");
        scanf("%d",&num);          /*รับค่าตัวเลขมาเก็บในตัวแปร num */
        printf("/n");              /*ขึ้นบรรทัดใหม่ */
}while(num > 100);                 /*ตรวจสอบว่า num มีค่ามากกว่า 100 หรือไม่ */
```

โปรแกรมที่ 9.2 ถ้าหากต้องการให้ระบบคอมพิวเตอร์รับค่าตัวเลขจำนวนเต็มเข้าไปที่ละค่า แล้วนำตัวเลขมารวมกันจนกระทั่งป้อนตัวเลขเป็น 0 ให้หยุดรับค่าและแสดงผลรวมออกมาทางจอภาพ จงวิเคราะห์ปัญหานี้ พร้อมทั้งเขียนผังงาน ซูโดโค้ดและโปรแกรมตัวอย่างภาษาซี

วิธีทำ

ต้องการอะไร	หาค่าผลรวมของตัวเลข
ต้องการเอาต์พุตอย่างไร	แสดงผลรวมทางจอภาพ
ข้อมูลเข้า	รับตัวเลขที่ละค่า ถ้าตัวเลขเป็น 0 ให้จบโปรแกรม
วิธีการประมวลผล	ต้องประกาศตัวแปรหนึ่งตัว (sum) สำหรับเก็บผลรวม โดยเริ่มต้นให้ตัวแปรนี้มีค่าเป็น 0 รับข้อมูล(x) เข้ามาทีละค่าแล้วเอาไปรวมกับ sum ถ้าหาก x เป็น 0 ให้แสดงผลรวมแล้วออกจากโปรแกรม

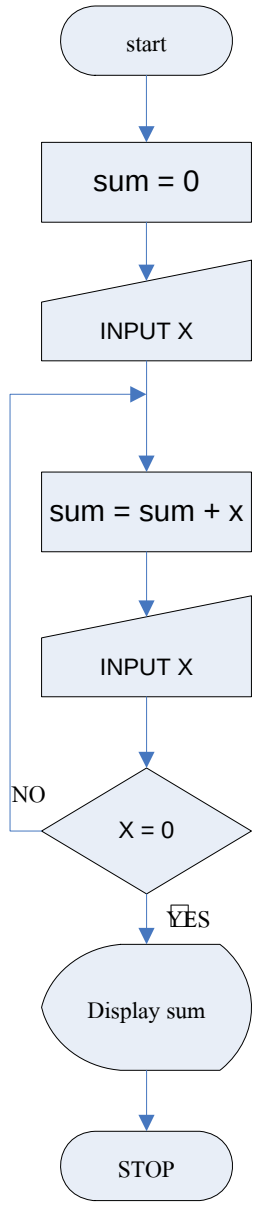
การทำซ้ำแบบนี้จะไม่ทราบจำนวนครั้งในการทำซ้ำที่แน่นอน โดยจะทำไปเรื่อย ๆ จนกว่าข้อมูลที่รับเข้าไปจะเป็น 0 ดังนั้นสามารถเขียนคำอธิบายวิธีการทำงานได้ดังนี้

1. เริ่มต้นทำงาน
2. กำหนดค่า sum ให้เท่ากับ 0
3. รับค่า X ทางแป้นพิมพ์
4. ทำในลูปถ้า X มีค่าไม่เท่ากับ 0
   4.1 หาค่าผลรวม sum = sum + X
   4.2 รับค่า X ทางแป้นพิมพ์
5. แสดงค่าผลรวม sum ทางจอภาพ
6. จบการทำงาน

สำหรับซูโดโค้ดสามารถเขียนได้ดังนี้

```
START
        INIT  SUM = 0
        READ  X
        do
                SUM  =  SUM + X
                READ  X
        while (X !=0)
        PRINT X
END
```

ภายในลูป



รูปที่ 9.6 ซูโดโค้ดและผังงานของโปรแกรมที่ 9.2

ถ้าหากเขียนเป็นโปรแกรมภาษาซีจะได้ดังโปรแกรมต่อไปนี้

```
#include “stdio.h”
#include “conio.h”
main()
{
      int x, sum;
      sum = 0;
      printf(“INPUT DATA “);
      scanf(“%d”,&x);
      do {
            sum = sum + x;
            printf(“INPUT DATA “);
            scanf(“%d”,&x);
      }while(x != 0);
      printf(“Sum = %d\n”,sum);
      getch();
}
```

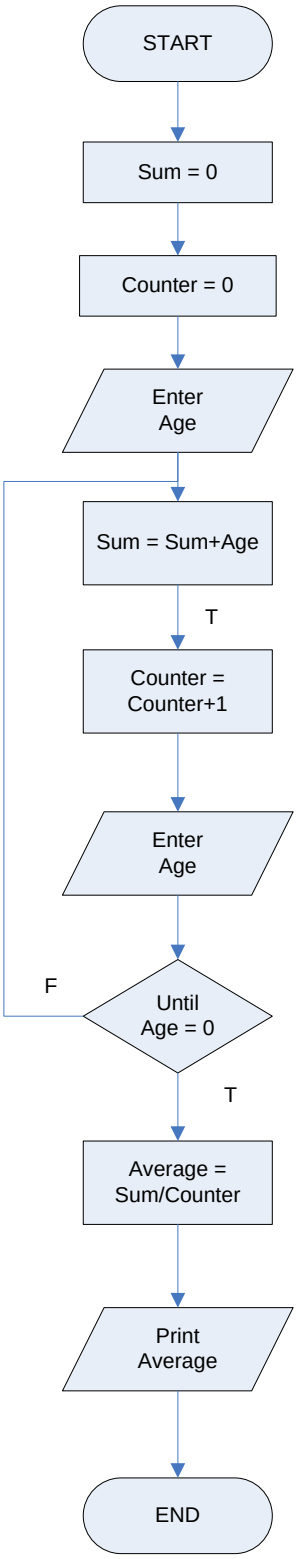
สำหรับผลลัพธ์จากการรันโปรแกรมจะได้ดังรูป โดยโปรแกรมจะให้ป้อนค่าตัวเลขต่าง ๆ เข้าไป เมื่อป้อนเลข 0 จะแสดงผลรวมและออกจากโปรแกรม

รูปแสดงผลลัพธ์จากการรันโปรแกรมในตัวอย่าง

ตัวอย่างที่ 9.4 จงเขียนรหัสลำลองและผังงาน ถ้าหากต้องการออกแบบโปรแกรมหาค่าเฉลี่ยของอายุ โดยให้โปรแกรมรับอายุเข้าไปที่ละค่า ถ้าหากป้อนอายุเป็น 0 ให้หยุดรับค่า จากนั้นให้พิมพ์ค่าเฉลี่ยของอายุออกมา

วิธีทำ การหาค่าเฉลี่ยจะต้องทราบผลรวมและจำนวนข้อมูลทั้งหมด ดังนั้นจะให้ตัวแปร Sum เป็นตัวแปรที่หาค่าผลรวม ตัวแปร Counter เป็นตัวแปรที่นับค่าข้อมูลที่ป้อนเข้าไป โดยอายุเฉลี่ยหาได้จาก ผมรวม (Sum) หารด้วยจำนวนข้อมูล (Counter) ดังนั้นรหัสลำลองและผังงานเขียนได้ดังรูปที่ 9.7

```
1.START
2.Sum = 0
3.Counter = 0
4.READ Age
5 DO
      Sum = Sum + Age
      Counter = Counter+1
      READ Age
  WHILE Age != 0
6.Average = Sum/Counter
7.Print Average
8. STOP
```

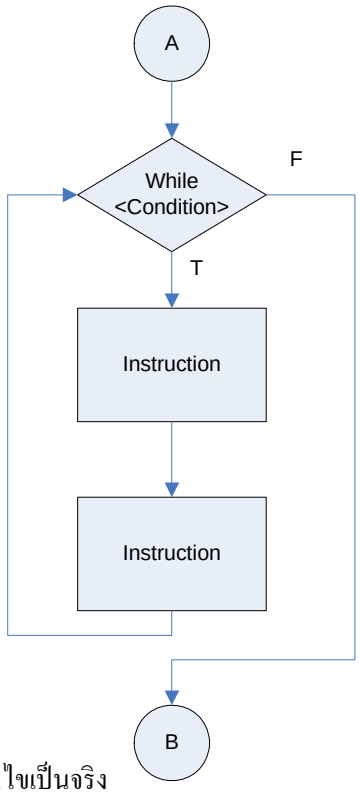


รูปที่ 9.7 ซูโดโค้ดและผังงาน

9.4 การทำซ้ำแบบถ้าเงื่อนไขเป็นจริงจะทำชุดคำสั่ง

การทำซ้ำแบบนี้จะมีการตรวจสอบเงื่อนไขก่อนการเข้าสู่ลูป ถ้าหากเงื่อนไขเป็นจริงจะทำในลูป แต่ถ้าหากเงื่อนไขเป็นเท็จจะออกจากการทำลูป ดังนั้นลูปแบบนี้อาจไม่มีการทำงานภายในลูปเลยก็ได้ถ้าหากเงื่อนไขเป็นเท็จ สำหรับคำอธิบายการทำงานจะใช้คำว่า “ในขณะที่...ทำ” โดยระหว่างคำว่าขณะที่กับทำจะเป็นนิพจน์ที่ตรวจสอบเงื่อนไป สำหรับการเขียนซูโดโค้ดสำหรับการทำซ้ำแบบนี้จะใช้คำว่า “WHILE...ENDWHILE” ผังงานของการทำซ้ำแบบนี้แสดงได้ดังรูปที่ 9.8 ตัวอย่างเช่นถ้าหากต้องการให้แสดงค่าเลขจำนวนเต็มที่เก็บอยู่ในตัวแปร x ตั้งแต่ 1 ถึง 10 จะเขียนคำอธิบายได้ดังนี้

1. เริ่มต้น
2. ให้ x เท่ากับ 0
3. ในขณะที่ x < 10 ทำ
   เพิ่มค่า x ขึ้นหนึ่งค่า
   แสดงค่า x
4. จบการทำงาน



รูปที่ 9.8 ผังงานของการทำซ้ำถ้าเงื่อนไขเป็นจริง

ตัวอย่างที่ 9.5 ถ้าหากต้องการพัฒนาโปรแกรมคอมพิวเตอร์สำหรับแปลงหน่วยการวัดอุณหภูมิจากหน่วยฟาเรนไฮซ์(fahrenheit) เป็นหน่วยเซลเซียส์(Celsius) โดยสามารถรับได้สูงสุด 15 ค่า จงเขียนรหัสลำลองของปัญหานี้พร้อมทั้งแสดงวิธีการตรวจสอบคำตอบ

วิธีทำ	**ต้องการอะไร**	แปลงหน่วยวัดอุณหภูมิจากฟาเรนไฮซ์เป็นเซลเซียส์
	ต้องการเอาต์พุตอย่างไร	แสดงค่าในหน่วยเซลเซียส์
	ข้อมูลเข้า	รับค่าเป็นฟาเรนไฮซ์ทีละค่าทางแป้นพิมพ์
	วิธีการประมวลผล	เนื่องจากรับได้สูงสุด 15 ค่า ดังนั้นจะต้องมีตัวแปรสำหรับนับ การแปลงจากฟาเรนไฮซ์เป็นเซลเซียส์คำนวณได้จาก

c_temp = (f_temp – 32)*(5/9)

ให้ตัวแปรเป็นดังนี้

c_temp	เป็นตัวแปรอุณหภูมิในหน่วยเซลเซียส์
f_temp	เป็นตัวแปรอุณหภูมิในหน่วยฟาเรนไฮซี
temp_count	เป็นตัวแปรสำหรับนับค่าที่รับเข้าไป

วิธีการประมวลผลสามารถเขียนเป็นคำอธิบายได้ดังนี้

1. เริ่มต้น
2. กำหนดให้ temp_count มีค่าเป็นศูนย์
3. ในขณะที่ temp_count น้อยกว่า 15
   3.1 รับค่า f_temp
   3.2 หาค่า c_temp จาก c_temp = (f_temp - 32) * 5/9
   3.3 แสดงผล c_temp
   3.4 เพิ่มค่า temp_count ขึ้นหนึ่งค่า
4. จบ

สำหรับรหัสลำลองเขียนได้ดังนี้

```
1.  START
2.  INIT  temp_count = 0
3.  WHILE  temp_count < 15
4.      READ  f_temp
5.      c_temp  =  (f_temp – 32) * 5/9
6.      PRINT  c_temp
7.      temp_count = temp_count + 1
    ENDWHILE
8.  STOP
```

สำหรับการตรวจสอบคำตอบนั้นอาจทำโดยสมมุติข้อมูลขึ้นมาเพียงสองค่า แล้วพิจารณาว่าการทำงานต่าง ๆ ในลูปถูกต้องหรือไม่ ดังขั้นตอนต่อไปนี้

1. กำหนดตัวแปรอุณหภูมิขึ้นมาสองค่าโดยให้เป็น 32 กับ 50

ตัวแปร	ข้อมูลตัวแรก	ข้อมูลตัวที่สอง
f_temp	32	50

2. ลองคิดคำตอบการประมวลผลเปลี่ยนเป็นอุณหภูมิหน่วยเซลเซียสเองจะได้

ตัวแปร	ข้อมูลตัวแรก	ข้อมูลตัวที่สอง
c_temp	0	10

3. สร้างตารางขึ้นมาแสดงการทำงานและดูค่าตัวแปรทีละขั้นตอนจากซูโดโค้ด โดยพิจารณาเงื่อนไขของลูปด้วย

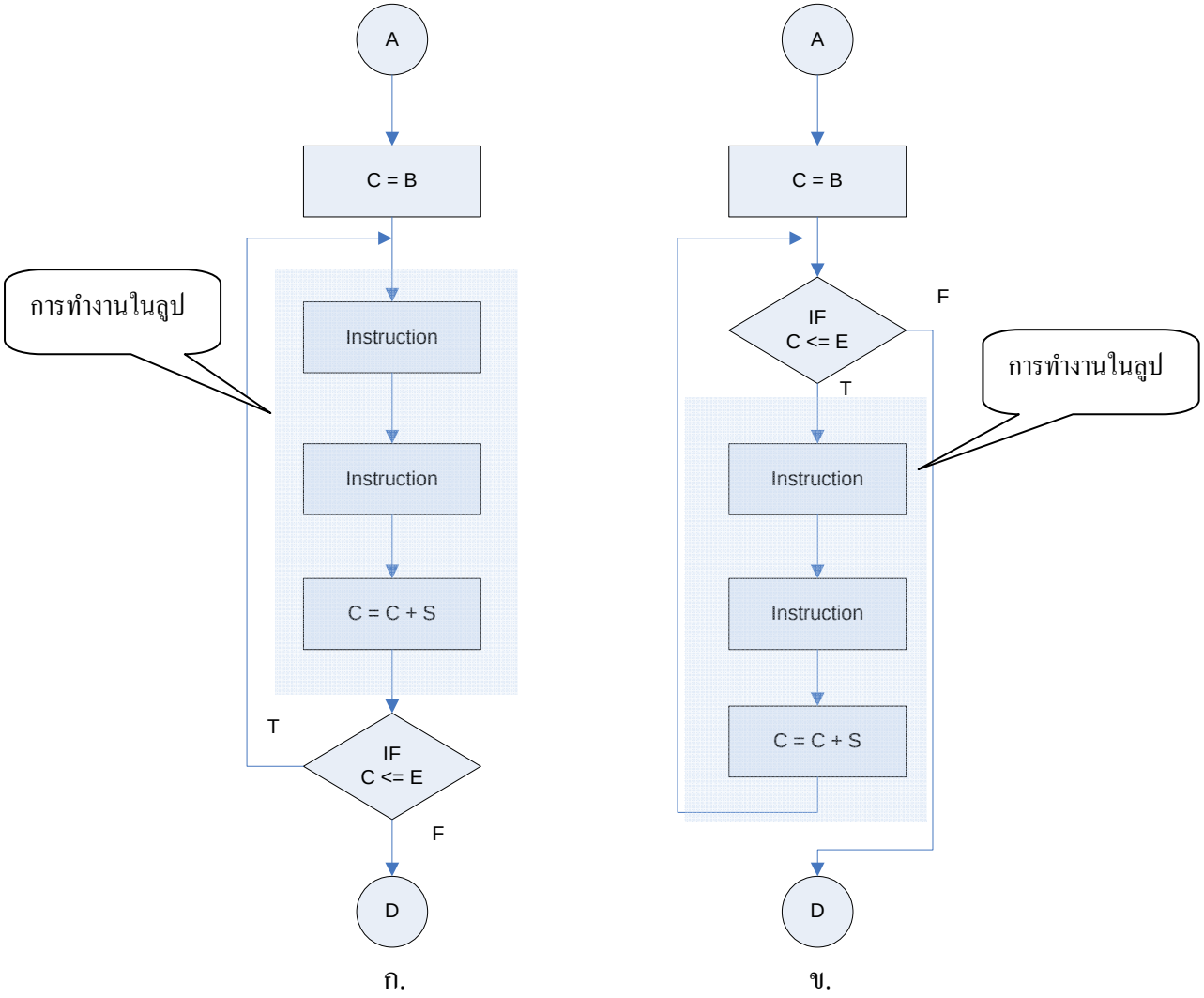
ขั้นตอน	temp_count	WHILE	f_temp	c_temp
1,2	0			
3		จริง		
4			32	
5				0
6				แสดงผล
7	1			
3		จริง		
4			50	
5				10
6				แสดงผล
7	2			

4. จะพบว่าการแสดงผลของตัวแปร c_temp ในตารางในข้อ 3 จะตรงกับตารางในข้อ 2 ดังนั้นวิธีการนี้ทำได้

9.5 แนวทางการประยุกต์การทำซ้ำ

จากที่ทราบมาแล้วว่าการทำซ้ำของคอมพิวเตอร์นั้นจะเป็นการทำชุดคำสั่งซ้ำตามเงื่อนไขที่กำหนด และไม่ว่าจะออกแบบการทำซ้ำให้ออกมาในรูปแบบใด ขั้นตอนวิธีการทำโปรแกรมจะต้องมีจุดจบของการทำซ้ำ

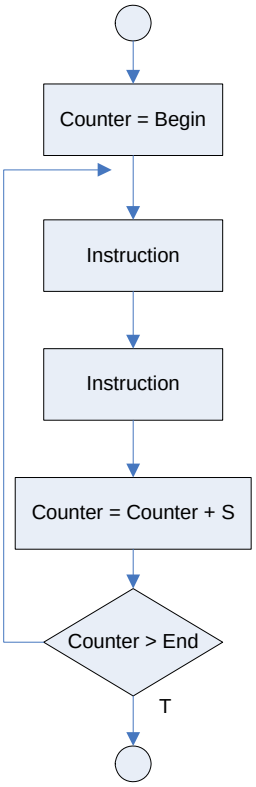
เนื่องจากการทำซ้ำจะต้องมีการตรวจสอบเงื่อนไข โดยที่เงื่อนไขนี้อาจตรวจสอบหลังการทำชุดคำสั่งที่ต้องทำซ้ำไปแล้วหนึ่งรอบ หรือตรวจสอบก่อนที่จะทำชุดคำสั่งที่ต้องการทำซ้ำก็ได้ ดังแสดงในแผนภาพต่อไปนี้



รูปที่ 9.9 ลักษณะผังงานการทำซ้ำ

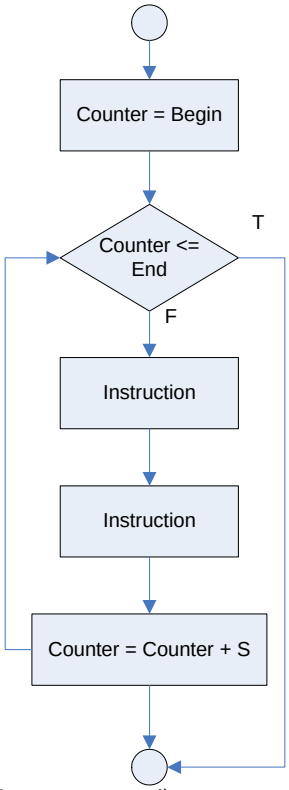
จากตัวอย่างผังงานจะให้ C เป็นตัวนับ, B แทนค่าตัวเลขเริ่มต้น, E แทนค่าตัวเลขตัวสุดท้าย และ S แทนค่าการนับที่จะมีขึ้น การทำซ้ำในของผังงานในรูปแบบ ก. เป็นการทำซ้ำที่ตรวจสอบเงื่อนไขหลังการทำในลูป ส่วนรูปแบบ ข. จะตรวจสอบเงื่อนไขก่อนเข้าสู่ลูป สำหรับการเขียนเป็นซูโดโค้ดทำได้โดยใช้คำว่า WHILE และ REPEAT-UNTIL ถ้าหากนำการทำซ้ำดังผังงานในรูป ก. มาเขียนเป็นซูโดโค้ด โดยใส่ค่าเริ่มต้นให้กับตัวนับและตรวจสอบเงื่อนไขของการนับว่านับมาถึงค่าสุดท้ายหรือยังจะเขียนได้ดังนี้

```
Counter  =  Begin
REPEAT
        Instruction
        Instruction
        Counter = Counter + S
UNTIL Counter > End
```



สำหรับการทำงานในลักษณะของผังงานรูป ข. ถ้าหากกำหนดค่าเริ่มต้น และเงื่อนไขการตรวจสอบการทำซ้ำ จะเขียนซูโดโค้ดและผังงานได้ดังนี้

```
Counter  =  Begin
WHILE  Counter <= End
        Instruction
        Instruction
ENDWHILE
```



จากการทำซ้ำทั้งสองรูปแบบที่กล่าวมา จะเห็นว่าเป็นการทำซ้ำที่มีตัวนับการทำซ้ำอยู่ภายในลูป และค่า S คือค่าที่จะเพิ่มขึ้นในการนับแต่ละครั้ง ลูปทั้งสองประเภทดังกล่าวสามารถใช้กับโปรแกรมที่ทราบจำนวนครั้งในการทำซ้ำแน่นอน หรือไม่ทราบจำนวนครั้งในการทำซ้ำก็ได้ แต่ในการเขียนโปรแกรมจะต้องมีจุดจบของการทำซ้ำ

ถ้าหากคอมพิวเตอร์ต้องการรับข้อมูลเข้ามาหรืออ่านข้อมูลใด ๆ เข้ามา แล้วมีการเขียนโปรแกรมทำซ้ำเพื่ออ่านข้อมูลค่าต่อ ๆ ไป เราสามารถเขียนโปรแกรมตรวจสอบเงื่อนไขการทำซ้ำกับข้อมูลเหล่านั้นได้ แต่อาจใช้ไม่ได้กับงานบางประเภท เทคนิคที่ใช้ในการตรวจสอบหาจุดจบของข้อมูลนิยมใช้สองเทคนิคดังนี้

1. **ใช้วิธีตรวจสอบหาข้อมูลตัวสุดท้าย** วิธีนี้จะใช้โปรแกรมทำซ้ำไปเรื่อย ๆ จะกว่าจะพบข้อมูลตัวสุดท้าย โดยจะมีการสมมุติค่าข้อมูลตัวสุดท้ายเป็นค่าที่รู้แน่ ๆ ว่าข้อมูลที่ใช้จะไม่เป็นค่านั้น เช่น การเขียนโปรแกรมรับค่าตัวเลขจำนวนเต็มบวก โดยให้โปรแกรมวนลูปรับไปเรื่อย ๆ ถ้าหากตัวเลขที่รับมาเป็นเลขลบก็หมายความว่าข้อมูลนั้น ๆ เป็นข้อมูลตัวสุดท้ายแล้ว การเขียนโปรแกรมทำซ้ำลักษณะนี้จะเป็นการทำซ้ำแบบ WHILE ดังตัวอย่างซูโดโค้ดต่อไปนี้

```
READ  DATA
WHILE   DATA != ค่าที่กำหนดตัวสุดท้าย
        …….
        …….
        READ  DATA
ENDWHILE
```

2. **การใช้ตัวนับ** วิธีการนี้จะต้องใช้เมื่อทราบจำนวนครั้งในการทำซ้ำที่แน่นอน โดยโปรแกรมจะต้องมีการตรวจสอบว่าตัวนับนั้นได้นับมาถึงค่าที่กำหนดหรือยัง ถ้าถึงแล้วก็ให้หยุดการประมวลผล การทำซ้ำแบบนี้สามารถนำการทำซ้ำในลักษณะ WHILE หรือ FOR มาใช้ได้ ดังปัญหาในตัวอย่างที่ 9.5 ที่ผ่านมา

ตัวอย่างที่ 9.6 ถ้าหากต้องการพัฒนาโปรแกรมหาค่าผลรวม ผลคูณและค่าเฉลี่ยของตัวเลขสองค่า โดยให้โปรแกรมรับค่าตัวเลขเข้าไปสองค่าทางแป้นพิมพ์ จากนั้นคำนวณค่าผลรวม ผลคูณ และค่าเฉลี่ยแล้วแสดงออกทางจอภาพ ถ้าหากผลรวมของตัวเลขมีค่ามากกว่า 200 ให้แสดงเครื่องหมาย asterisk (*) ออกมาด้วย โดยให้โปรแกรมวนลูปรับตัวเลขครั้งละสองค่าไปเรื่อย ๆ แต่ถ้าหากตัวเลขทั้งสองค่าเป็นศูนย์ทั้งสองตัว ให้หยุดการทำซ้ำ จงออกแบบแนวทางการเขียนโปรแกรมและเขียนซูโดโค้ดของโปรแกรมนี้

วิธีทำ จากตัวอย่างนี้จะเป็นการทำซ้ำ โดยสามารถออกจากการทำซ้ำโดยใช้เทคนิคการตรวจสอบหาข้อมูลตัวสุดท้าย จากตัวอย่างที่โจทย์กำหนดสามารถสรุปได้ดังนี้

ต้องการอะไร	คำนวณหาค่าผลรวม ผลคูณและค่าเฉลี่ย
ต้องการเอาต์พุตอย่างไร	แสดงเอาต์พุตทางจอภาพ ถ้าผลรวมเกิน 200 ให้แสดง *
ข้อมูลเข้า	ตัวเลขสองค่า ถ้าเป็นศูนย์ทั้งสองค่าให้หยุดรับค่า
วิธีการประมวลผล	ผลรวม = ตัวเลขแรก + ตัวเลขที่สอง
	ผลคูณ = ตัวเลขแรก * ตัวเลขที่สอง
	ค่าเฉลี่ย = ผลรวม หารด้วยสอง
	ถ้า ผลรวม > 200 แล้วแสดงเครื่องหมาย *

กำหนดให้ตัวแปรเป็นดังนี้

number1	เป็นตัวแปรเก็บตัวเลขตัวแรก
number2	เป็นตัวแปรเก็บตัวเลขตัวที่สอง
sum, product, average	เป็นตัวแปรสำหรับเก็บค่าผลรวม ผลคูณ และค่าเฉลี่ย

- การประมวลผลแบบทำซ้ำจะใช้รูปแบบ WHILE โดยตรวจสอบเงื่อนไขก่อนการทำซ้ำ ว่าตัวเลขที่เข้ามาเป็นศูนย์หรือไม่
- การตรวจสอบเงื่อนไขจะนำตัวดำเนินการ AND มาใช้
- ใช้การเลือกทำแบบ IF ว่าจะพิมพ์เครื่องหมาย * หรือไม่

วิธีการประมวลผลสามารถเขียนเป็นคำอธิบายได้ดังนี้

1. เริ่มต้น
2. กำหนดให้ sum มีค่าเป็นศูนย์
3. แสดงเครื่องหมาย Prompt สำหรับรับตัวเลขสองตัว
4. รับค่า number1 และ number2
5. ในขณะที่ number1 และ number2 ไม่เท่ากับศูนย์

   5.1 หาค่า sum = number1 + number2

   5.2 หาค่า produce = number1 * number2

   5.3 หาค่า average = sum / 2

   5.4 ถ้า sum มากกว่า 200 แล้ว

      5.4.1 แสดงผล sum, '*' ,product, average

   มิฉะนั้น

      5.4.2 แสดงผล sum, product, average

   5.5 แสดงเครื่องหมาย Prompt สำหรับรับตัวเลขสองตัว

   5.6 รับค่า number1 และ number2
6. จบ

สำหรับซูโดโค้ดเขียนได้ดังนี้

```
START
INIT  sum = 0
Prompt for number1, number2
READ number1, number2
WHILE  NOT(number1 = 0 AND number2 = 0)
     sum = number1 + number2
     product = number1*number2
     average  =  sum/2
     IF  sum > 200 THEN
          Display sum, '*', product, average
     ELSE
          Display sum, product, average
```

```
        ENDIF
        Prompt for number1, number2
        READ number1, number2
    ENDWHILE
    STOP
```

ตัวอย่างที่ 9.7 ถ้าหากมีแฟ้มข้อมูลของนักเรียนอยู่แฟ้มหนึ่ง โดยที่ข้อมูลของนักเรียนแต่ละคนจะถูกเก็บเป็นเรคอร์ด นักเรียนหนึ่งคนมีสองเรคอร์ด ใช้ชื่อว่าเรคอร์ด ‘S’ และเรคอร์ด ‘U’ โดยเรคอร์ด ‘S’ เก็บรหัสประจำตัวนักเรียน, ชื่อ, อายุ, เพศ, ที่อยู่ และสถานภาพว่าเป็นนักเรียนแบบ เต็มเวลา (full-time :F/T) หรือเป็นนักเรียนแบบนอกเวลา (part-time : P/T) สำหรับเรคอร์ด ‘U’ จะเก็บชื่อ และหน่วยกิตที่นักเรียนคนนั้นเรียนอยู่ จงออกแบบอัลกอริธึมสำหรับอ่านไฟล์เรคอร์ดข้อมูลนักเรียนแล้วพิมพ์รหัสประจำตัวนักเรียน ชื่อ และที่อยู่ของนักเรียนแต่ละคำออกมา

วิธีทำ

กำหนดปัญหา

ข้อมูลและตัวแปรทางอินพุต มีเรคอร์ดข้อมูลอยู่สองเรคอร์ด

‘S’ Records

- number แทนรหัสประจำตัวนักเรียน
- name แทนชื่อนักเรียน
- address แทนที่อยู่นักเรียน
- age แทนอายุ
- gender แทนเพศ
- attendance_pattern แทนสถานภาพนักเรียน

‘U’ Record

วิธีการประมวลผล

- พิมพ์ส่วนหัวของรายการ
- อ่านเรคอร์ดข้อมูลนักเรียนแต่ละคน
- เลือกเฉพาะเรคอร์ด ‘S’
- พิมพ์เฉพาะเรคอร์ดที่เลือก

เอาต์พุต

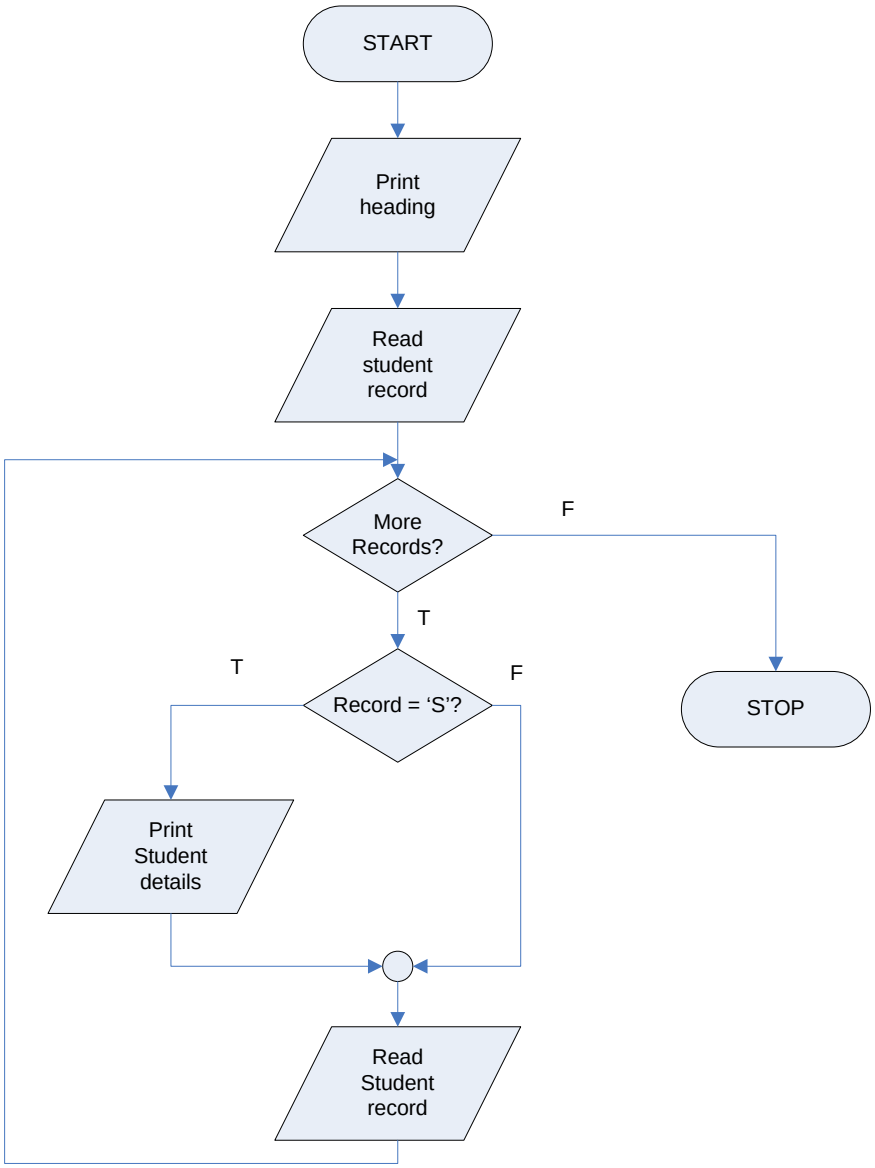
- พิมพ์ส่วนหัวของรายการ
- เลือกเรคอร์ด แล้วพิมพ์ number, name และ address

การประมวลผลจะต้องใช้วิธีการทำซ้ำ โดยวนลูปอ่านข้อมูลขึ้นมาจนกว่าจะอ่านข้อมูลนักเรียนครบทุกคน โดยจะให้จุดสุดท้ายของไฟล์เป็นตัวบอกว่าจบการทำซ้ำ

ในการอ่านข้อมูลของนักเรียนแต่ละคน จะต้องใช้การเลือกทำ โดยเลือกเฉพาะเรคอร์ด ‘S’ ขึ้นมาแสดงผลเท่านั้น

จากการวิเคราะห์ปัญหาที่โจทย์กำหนดสามารถเขียนเป็นอัลกอริธึมและผังงานได้ดังนี้

```
Print_student_records
     Print ‘STUDENT LIST’ heading
     READ student record
     WHILE  more records exist
          IF student record = ‘S’ record THEN
               Print student_number, name, address
          ENDIF
          READ student record
     ENDWHILE
END
```



รูปที่ 9.10 แสดงผังงานของตัวอย่างที่ 9.7

ตัวอย่างที่ 9.8 จากปัญหาในตัวอย่างที่ 9.7 จงออกแบบอัลกอริธึมสำหรับสร้างโมดูลย่อยเพื่ออ่านไฟล์นักเรียน และแสดงรหัสประจำตัว, ชื่อ, ที่อยู่ และอายุ ของนักเรียนที่เป็นผู้หญิง และเรียนแบบนอกเวลา

วิธีทำ จากปัญหานี้ จะต้องใช้วิธีการวนลูปอ่านข้อมูลในลักษณะเดียวกับตัวอย่างที่ 9.7 แต่ในลูปนั้นจะต้องเพิ่มเงื่อนไขเพื่อเลือกเฉพาะนักเรียนที่เป็นผู้หญิง และเรียนแบบนอกเวลา การเขียนอัลกอริธึมเพื่อแก้ปัญหานี้ทำได้หลายวิธี โดยอาจนำการเลือกทำมาซ้อนกันเพื่อเลือกเงื่อนไขทีละเงื่อนไข หรือกำหนดเงื่อนไขการเลือกทำให้ละเอียดขึ้นก็ได้ สำหรับในตัวอย่างนี้จะเสนออัลกอริธึมของโปรแกรมย่อยสามแบบคือ

รูปแบบที่ 1 ใช้ IF ในการเลือกทำแต่ละเงื่อนไข

```
Produce_part_time_female_list
     Print 'PART TIME FEMALE STUDENTS' heading
     READ student record
     WHILE  more records
          IF student record = 'S' record THEN
               IF attendance_pattern = P/T THEN
                    IF gender = female THEN
                         Print student_number, name,
                               address, age
                    ENDIF
               ENDIF
          ENDIF
          READ student record
     ENDWHILE
END
```

รูปแบบที่ 2 ใช้ IF ซ้อนกันสองตัว แล้วกำหนดเงื่อนไขให้ละเอียดขึ้น

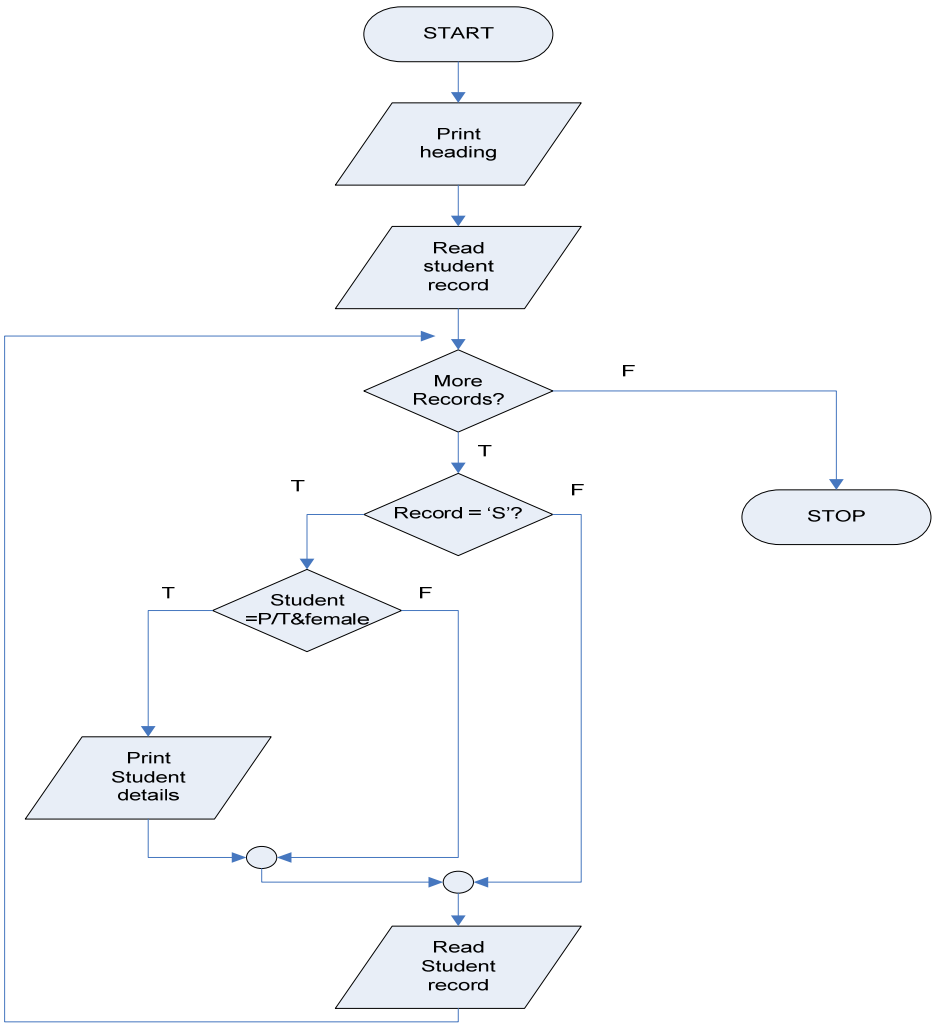
```
Produce_part_time_female_list
     Print 'PART TIME FEMALE STUDENTS' heading
     READ student record
     WHILE  more records
          IF student record = 'S' record THEN
               IF (attendance_pattern = P/T
                   AND gender = female) THEN
                         Print student_number, name,
                               address, age
               ENDIF
          ENDIF
          READ student record
     ENDWHILE
END
```

รูปแบบที่ 3 กำหนดเงื่อนไขของ IF ให้ละเอียดขึ้น

```
Produce_part_time_female_list
     Print 'PART TIME FEMALE STUDENTS' heading
     READ student record
     WHILE  more records
          IF student record = 'S' record
          AND attendance_pattern = P/T
          AND gender = female) THEN
               Print student_number, name, address, age
          ENDIF
          READ student record
     ENDWHILE
END
```

กำหนดเงื่อนไขให้ระเอียดขึ้น

สำหรับผังงานเขียนได้ดังนี้



รูปที่ 9.11 แสดงผังงานอัลกอริธึมในตัวอย่างที่ 9.8

ตัวอย่างที่ 9.9 จากตัวอย่างที่ 9.8 จงพัฒนาอัลกอริธึมให้แสดงผลเพิ่มเติมดังต่อไปนี้

- แสดงจำนวนนักเรียนทั้งหมด
- แสดงจำนวนนักเรียนเพศหญิงที่เรียนนอกเวลา

วิธีทำ จากปัญหาในข้อนี้จะต้องสร้างตัวนับขึ้นมาสองตัวดังนี้

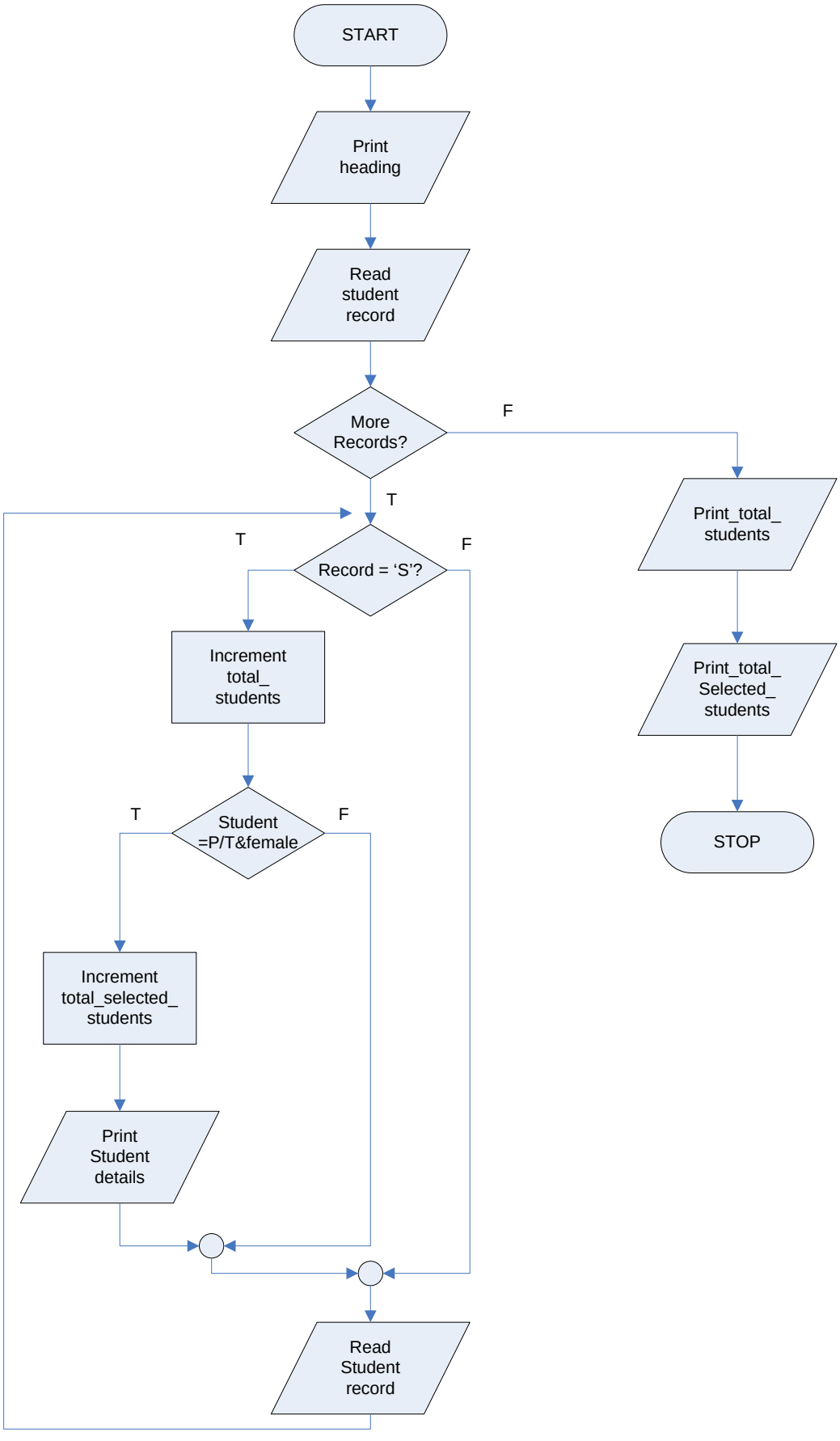
total_students ใช้นับจำนวนนักเรียนทั้งหมด

total_selected_students ใช้นับจำนวนนักเรียนที่เลือก

การประมวลผลในเบื้องต้นจะกำหนดให้ตัวนับทั้งสองตัวมีค่าเป็นศูนย์ เมื่ออ่านเรคอร์ดของนักเรียนเข้ามาแล้วให้เพิ่มตัวนับขึ้นหนึ่งค่า และถ้าเป็นนักเรียนหญิงที่เรียนนอกเวลาให้เพิ่มตัวนับนักเรียนที่เลือกขึ้นหนึ่งค่า ซึ่งจะเขียนอัลกอริธึมของโปรแกรมย่อยนี้ได้ดังนี้

```
Produce_part_time_female_list
    Print 'PART TIME FEMALE STUDENTS' heading
    Set total_students to zero
    Set total_selected_students to zero
    READ student record
    WHILE records exist
        IF student record = 'S' record THEN
            increment total_students
            IF (attendance_pattern = P/T
            AND gender = female) THEN
                increment total_selected_students
                Print student_number, name,
                      address, age
            ENDIF
        ENDIF
        READ student record
    ENDWHILE
    Print total_students
    Print total_selected_students
END
```

สำหรับผังงานของการแก้ปัญหานี้เขียนได้ดังนี้


START
Print heading
Read student record
More Records?
F
T
Print_total_ students
T
F
Record = 'S'?
Increment total_ students
Print_total_ Selected_ students
T
F
Student =P/T&female
STOP
Increment total_selected_ students
Print Student details
Read Student record

แบบฝึกหัดท้ายบท

ตอนที่ 1 จงเลือกคำตอบที่ถูกต้องที่สุดเพียงหนึ่งข้อ

1. โครงสร้างโปรแกรมแบบมีการทำซ้ำมีกี่แบบ
   ก. 1 แบบ ข. 2 แบบ
   ค. 3 แบบ ง. 4 แบบ
2. ซูโดโค้ดของการทำซ้ำที่ทราบจำนวนครั้งในการทำที่แน่นอนมักจะมีคำใดประกอบอยู่
   ก. “FOR” ข. “REPEAT”
   ค. “LOOP” ง. “UNTIL”
3. ขั้นตอนการทำงานที่ประกอบอยู่ภายในการทำงานแบบทำซ้ำคือข้อใด
   ก. การทำงานแบบลำดับ ข. การแสดงผลข้อมูล
   ค. การทำแบบมีทางเลือก ง. ถูกมากกว่าหนึ่งข้อ
4. ในการเขียนผังงานของงานที่มีการทำซ้ำ จะมีสัญลักษณ์ใดประกอบอยู่
   ก. สัญลักษณ์การทำแบบมีทางเลือก ข. สัญลักษณ์การรับข้อมูล
   ค. สัญลักษณ์เริ่มต้น ง. สัญลักษณ์การแสดงข้อมูล
5. ในการทำซ้ำแบบที่ทราบจำนวนครั้งที่แน่นอน ระบบจะทำสิ่งใดก่อน
   ก. เพิ่มค่าตัวนับ ข. ตรวจสอบเงื่อนไข
   ค. ทำงานในลูป ง. พิสูจน์ตัวนับ
6. การทำซ้ำจนระบบมีเงื่อนไขอย่างหนึ่งจึงหยุด มีลักษณะดังข้อใด
   ก. ตรวจสอบเงื่อนไขก่อนทำชุดคำสั่งในลูป
   ข. ตรวจสอบเงื่อนไขหลังจากทำชุดคำสั่งในลูป
   ค. ต้องมีตัวควบคุมการนับลูป
   ง. ในลูปต้องมีตัวแปรมากกว่าหนึ่งตัว
7. ในการเขียนซูโดโค้ดของการทำซ้ำแบบถ้าเงื่อนไขเป็นจริงจะทำชุดคำสั่งจะใช้คำว่าอะไร
   ก. “REPEAT….UNTIL” ข. “WHILE….ENDWHILE”
   ค. “UNTIL…GO” ง. ถูกมากกว่าหนึ่งข้อ
8. ในการทำซ้ำทุกรูปแบบจะต้องมีการประมวลผลแบบใด
   ก. การตรวจสอบเงื่อนไข ข. การคำนวณค่ากับตัวแปร
   ค. การเพิ่มค่าให้ตัวแปร ง. การลดค่าตัวแปรลงหนึ่งค่า
9. ข้อใดถูกต้องสำหรับการทำซ้ำที่ซูโดโค้ดมีคำว่า “REPEAT…UNTIL” ประกอบอยู่
   ก. ถ้าเงื่อนไขเป็นจริงจะออกจากลูป ข. ตรวจสอบเงื่อนไขก่อนทำลูป
   ค. มีการลดค่าตัวแปรควบคุม ง. ถูกทุกข้อ
10. ข้อใดถูกต้องสำหรับการทำซ้ำที่ซูโดโค้ดมีคำว่า “WHILE…ENDWHILE” ประกอบอยู่
    ก. ทำภายในลูปหนึ่งครั้งเสมอ ข. ตรวจสอบเงื่อนไขก่อนการทำซ้ำ
    ค. ต้องมีตัวแปรภายในลูปสองตัว ง. ถูกทุกข้อ

ตอนที่ 3 จงตอบคำถามต่อไปนี้

1. การทำงานแบบมีการทำซ้ำมีลักษณะเป็นอย่างไร

……

……

……

2. จงเขียนผังงานตามคำอธิบายโปรแกรมต่อไปนี้

เริ่มต้น

ให้ I เท่ากับ 0

ในขณะที่ I < 5 ทำ

รับค่า X

รับค่า Y

ถ้า Y มากกว่า X แล้ว

คำนวณค่า M มีค่าเท่ากับ X ยกกำลังสอง บวกกับ Y ยกกำลังสอง

มิฉะนั้น

คำนวณค่า M มีค่าเท่ากับ X บวกกับ Y

เพิ่มค่า I ขึ้น 1 ค่า

แสดงค่าของ M ทางจอภาพ

จบ

3. จากข้อ 2 โปรแกรมมีการทำซ้ำกี่ครั้ง และเมื่อออกจากลูปตัวแปร I มีค่าเท่าไร

……

……

4. ถ้าหากต้องการออกแบบโปรแกรมให้ทำการรับข้อมูลเลขจำนวนเต็มเข้าไป 10 ค่า จากนั้นให้แสดงผลรวม ค่าเฉลี่ย ค่าสูงสุด และค่าต่ำสุดออกมาทางจอภาพ
จงวิเคราะห์ปัญหานี้ พร้อมทั้งเขียนคำอธิบายโปรแกรม ซูโดโค้ดและผังงาน

5. จงเขียนอัลกอริธึมให้คอมพิวเตอร์แสดงสูตรคูณดังตัวอย่างต่อไปนี้

```
7 x 1     =     7
7 x 2     =     14
7 x 3     =     21
.....
.....
7 x 12    =     84
```

6. จากข้อ 5 ถ้าหากโปรแกรมสามารถป้อนค่าเข้าทางอินพุตได้ว่าต้องการสูตรคูณแม่ใด จะเขียนอัลกอริธึมได้อย่างไร

บทที่ 10

การประมวลผลแบบอาร์เรย์

เมื่อเครื่องคอมพิวเตอร์มีการประกาศตัวแปรเกิดขึ้น ชื่อตัวแปรจะเป็นตัวแทนตำแหน่งหน่วยความจำของคอมพิวเตอร์ ตัวแปรหนึ่งตัวจะแทนหน่วยความจำหนึ่งตำแหน่งสามารถเก็บข้อมูลได้หนึ่งตัว ถ้าหากต้องการเก็บข้อมูลหลายค่าจะต้องประกาศตัวแปรขึ้นมาหลายตัว แต่การประมวลผลของคอมพิวเตอร์จะมีการเก็บข้อมูลอีกแบบหนึ่งที่ใช้ชื่อเพียงชื่อเดียวแต่สามารถเก็บข้อมูลได้หลายตัวโดยที่ตัวแปรแต่ละตัวจะใช้เนื้อที่ของหน่วยความจำเท่ากัน ตัวแปรประเภทนี้เรียกว่าตัวแปรแบบอาร์เรย์ (array)

การเก็บข้อมูลแบบอาร์เรย์นั้นอาจมองอีกอย่างหนึ่งว่าเป็นการเก็บข้อมูลแบบตาราง เมื่อมีการพิจารณาถึงอาร์เรย์หนึ่งตัวก็เสมือนว่าเรากำลังมองข้อมูลอยู่ชุดหนึ่งที่อยู่เรียงลำดับกัน ในหนึ่งตำแหน่งของหน่วยความจำแบบอาร์เรย์เรียกว่า อีลิเมนต์ (element) หรือเซล(cell) ส่วนการระบุถึงอีลิเมนต์ในอาร์เรย์จะใช้ตัวเลขอินเด็กซ์ (index number) ในการอ้างถึงแต่ละตำแหน่งในอาร์เรย์นั้น ตัวอย่างเช่นถ้ามีข้อมูลอยู่กลุ่มหนึ่งซึ่งเป็นคะแนนของนักศึกษา 8 คน สามารถเก็บได้ดังนี้

หมายเลข	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]	X[8]
คะแนน	20	35	84	21	45	65	71	39

จากข้อมูลข้างต้นสามารถบอกได้ว่าข้อมูลอยู่ในอาร์เรย์ชื่อ X ในแต่ละเซลจะเก็บเลขจำนวนเต็ม ส่วนตัวเลขที่อยู่ในเครื่องหมาย [ ] คืออินเด็กซ์ ซึ่งจะต้องเป็นข้อมูลประเภทจำนวนเต็มเท่านั้น ข้อมูลข้างต้นอาจเขียนเต็ม ๆ ได้ว่า X[1..8] ถ้าหากต้องการติดต่อกับเซลล์ใดก็ให้ใช้อินเด็กซ์เป็นตัวชี้ ถ้าหากเราอ้างตัวแปร X[3] หมายความว่าเป็นการติดต่อกับอาร์เรย์ X เซลล์ที่ 3

ตัวอย่าง

X[2]	อ้างเซลล์ที่ 2 มีค่าเท่ากับ 35
X[2] + X[3]	นำเซลล์ที่ 2 บวกกับเซลล์ที่ 3 จะได้ 35 + 84 เท่ากับ 119
X[1+3]	อ้างเซลล์ที่ 4 มีค่าเท่ากับ 21
X[5] + 1	นำเซลล์ที่ 5 มาบวกด้วย 1 จะได้เท่ากับ 46

ถ้าหากในโปรแกรมมีการประกาศตัวแปรอาร์เรย์เอาไว้ เราสามารถนำตัวแปรนี้มาใช้ได้เหมือนกับตัวแปรทั่วไป เช่น

```
X[5]      :=      45;          ใส่ค่า 45 ลงในตัวแปรอาร์เรย์ X เซลล์ที่ 5
Print X[6])                    พิมพ์ค่าในตัวแปรอาร์เรย์ X เซลล์ที่ 6
```

สำหรับการใช้อินเด็กซ์ในการอ้างถึงเซลในอาร์เรย์นั้น ในการเขียนโปรแกรมบางภาษาจะเริ่มต้นด้วยเลข 0 เช่นภาษาซี บางภาษาจะเริ่มต้นด้วยเลข 1 เช่นภาษาปาสคาล ซึ่งผู้เขียนโปรแกรมจะต้องทราบด้วยว่าภาษาที่ใช้เขียนนั้นเริ่มต้นด้วยเลขใด

10.1. เหตุใดจึงต้องเก็บข้อมูลแบบอาร์เรย์

สมมุติว่ามีนักเรียนกลุ่มหนึ่งในกลุ่มนั้นมีอยู่ 9 คน ถ้าหากต้องการเก็บอายุของนักเรียนทั้ง 9 คน อาจทำได้โดยประกาศตัวแปรอาร์เรย์ขึ้นมา ในที่นี้ให้ชื่อว่า Age แล้วเก็บอายุของนักเรียนแต่ละคนลงไป การอ้างถึงอายุของนักเรียนแต่ละคนสามารถทำได้โดยใช้ชื่ออาร์เรย์ แล้วป้อนลำดับที่ของนักเรียนลงไป ก็จะได้อายุของนักเรียนลำดับนั้นออกมา เช่นถ้าหากต้องการอายุของนักเรียนคนที่ 7 ก็อ้างไปที่ Age(7) เป็นต้น

	Array Age	การอ้างถึง ชื่อตัวแปร
1	12	Age(1)
2	9	Age(2)
3	11	Age(3)
4	12	Age(4)
5	9	Age(5)
6	14	Age(6)
7	13	Age(7)
8	8	Age(8)
9	16	Age(9)

ตัวอย่างการเก็บอายุของนักเรียนในรูปของตัวแปรอาร์เรย์

การเก็บข้อมูลในลักษณะอาร์เรย์หรือตารางนี้จะทำให้สามารถอ้างถึงข้อมูลได้ง่าย ถ้าหากไม่มีการเก็บเป็นแบบอาร์เรย์ จะต้องประกาศตัวแปรสำหรับเก็บอายุขึ้นมา 9 ตัว การใช้งานตัวแปรแบบอาร์เรย์ส่วนใหญ่แล้วมักจะมีการประมวลผลในลักษณะต่อไปนี้

- โหลดข้อมูลเป็นค่าเริ่มต้นให้กับข้อมูลแต่ละเซลในอาร์เรย์
- ประมวลผลข้อมูลในแต่ละเซลของอาร์เรย์
- ค้นหาข้อมูลในอาร์เรย์
- พิมพ์ข้อมูลในอาร์เรย์ออกมาเป็นรายงาน

ถ้าหากต้องการเขียนโปรแกรมให้รับข้อมูลทางแป้นพิมพ์เข้าไปจำนวน 10 ค่า แล้วหาผลรวม เราอาจเขียนอัลกอริธึมได้ตามวิธีที่เคยได้ศึกษามาเช่น

```
Fine_sum_10_number
        Set sum to zero
        FOR I = 1 TO 10 DO          /** วนลูปรับข้อมูล 10 ค่า
                READ  x             /** รับค่า x
                sum = sum + x       /** คำนวณหาค่าผลรวมทีละค่า
        ENDFOR
        PRINT  x                    /** แสดงค่าผลรวม
END
```

การเขียนอัลกอริธึมดังกล่าว โปรแกรมสามารถรับตัวเลขเข้าไปทีละค่าจนครบ 10 ค่า แล้วหาผลรวมได้เช่นกัน แต่การเขียนโปรแกรมในลักษณะนี้ข้อมูลที่รับเข้าไปแต่ละค่าจะทับกับตัวแปรตัวเดิม เนื่องจากมีการประกาศตัวแปร x เพียงหนึ่งตัวเท่านั้น ทำให้เราไม่สามารถอ้างถึงข้อมูลที่เคยเก็บไว้แล้วได้ แต่ก็หาผลลัพธ์ได้เช่นกัน ถ้าหากนำอาร์เรย์มาใช้โดยตอนรับข้อมูลจะนำไปเก็บในอาร์เรย์แต่ละเซล แล้วนำมาหาผลรวม จะทำให้เราสามารถอ้างถึงข้อมูลที่เก็บไว้แล้วได้

ตัวอย่างที่ 10.1 จงเขียนอัลกอริธึม สำหรับโปรแกรมย่อยคำนวณหาค่าผลรวมของตัวเลขที่เก็บอยู่ในอาร์เรย์ทั้งหมด ตามจำนวนที่กำหนด

วิธีทำ กำหนดให้ array คือชื่ออาร์เรย์ที่เก็บข้อมูล, ให้ Sum เป็นตัวแปรเก็บผลรวม และให้ number_of_elements เป็นจำนวนเซลทั้งหมดของอาร์เรย์
ดังนั้นจะเขียนอัลกอริธึมได้ดังนี้

```
Find_sum_of_elements
        Set sum to zero                     /** กำหนดให้ sum เริ่มต้นเป็น 0
        FOR index = 1 to number_of_elements
                sum = sum + array(index)    /** หาค่าผลรวมทีละเซล
        ENDFOR
        Print sum
END
```

ตัวอย่างที่ 10.2 จากตัวอย่างที่ 10.1 จงเขียนอัลกอริธึมสำหรับหาค่าเฉลี่ยของข้อมูลทั้งหมดที่เก็บอยู่ในอาร์เรย์

วิธีทำ ค่าเฉลี่ยหาได้จากผลรวมหารด้วยจำนวนทั้งหมด สามารถเขียนอัลกอริธึมได้ดังนี้

```
Find_element_average
        Set sum to zero
        FOR index = 1 to number_of_elements
                sum = sum + array(index)
        ENDFOR
        average = sum / number_of_elements
        Print average
END
```

ตัวอย่างที่ 10.3 ถ้าหากมีข้อมูลอยู่ในอาร์เรย์ จงออกแบบอัลกอริธึมสำหรับหาค่าสูงสุดที่เก็บอยู่ในอาร์เรย์

วิธีทำ สมมุติว่าในอาร์เรย์เก็บข้อมูลดังต่อไปนี้

8 3 4 9 2 6

วิธีแก้ปัญหาอาจกำหนดตัวแปรขึ้นมาตัวหนึ่งสำหรับเก็บค่าสูงสุด แล้วให้ค่าตัวแปรนั้นมีค่าเท่ากับเซลที่ 1 ในที่นี้คือ 8 จากนั้นอ่านเซลต่อไปเข้ามาแล้วเปรียบเทียบว่ามีค่ามากกว่าค่าสูงสุดหรือไม่ ถ้าไม่มากกว่าก็ให้ข้ามไป แต่ถ้ามากกว่า ก็ให้ตัวแปรที่เก็บค่าสูงสุดเท่ากับค่าในเซลนั้น แล้ววนลูปเปรียบเทียบไปจนครบทุกเซลของอาร์เรย์ ถ้าหากให้ตัวแปรชื่อ largest_element เก็บค่าสูงสุด จะสามารถเขียนอัลกอริธึมของโปรแกรมย่อยหาค่าสูงสุดแล้วแสดงผลออกมาได้ดังนี้

```
Find_largest_element
        Set largest_element to array(1)
        FOR index = 2 to number_of_elements
                IF array(index) > largest_element THEN
                        largest_element = array(index)
                ENDIF
        ENDFOR
        Print largest_element
END
```

ตัวอย่างที่ 10.4 ถ้าหากมีข้อมูลเก็บอยู่ในอาร์เรย์ จงเขียนอัลกอริธึมสำหรับหาค่าต่ำสุดที่เก็บอยู่ในอาร์เรย์

วิธีทำ คล้ายกับตัวอย่างที่ 10.3 โดยกำหนดตัวแปรชื่อ smallest_element เป็นตัวแปรเก็บค่าต่ำสุด แล้วกำหนดให้เซลแรกเป็นค่าต่ำสุด จากนั้นเปรียบเทียบไปจนครบทุกเซล ซึ่งสามารถเขียนเป็นอัลกอริธึมได้ดังนี้

```
Find_smallest_element
        Set smallest_element to array(1)
        FOR index = 2 to number_of_elements
                IF array(index) < smallest_element THEN
                        smallest_element = array(index)
                ENDIF
        ENDFOR
        Print smallest_element
END
```

ตัวอย่างที่ 10.5 ถ้าหากมีข้อมูลอยู่ในอาร์เรย์ จงเขียนอัลกอริธึมสำหรับหาค่าสูงสุดและค่าต่ำสุดข้อข้อมูลที่เก็บอยู่ในอาร์เรย์

วิธีทำ คล้ายกับตัวอย่างที่ 10.4 และ 10.5 โดยให้การตรวจสอบหาค่าสูงสุดและค่าต่ำสุดทำไปพร้อมกันและทำภายในลูปการทำซ้ำ ซึ่งสามารถเขียนอัลกอริธึมได้ดังนี้

```
Find_range_of_elements
        Set smallest_element to array(1)
        Set largest_element to array(1)
        FOR index = 2 to number_of_elements
                IF array(index) < smallest_element THEN
                        smallest_element = array(index)
                ELSE
                        IF array(index) > largest_element THEN
                                largest_element = array(index)
                        ENDIF
                ENDIF
        ENDFOR
        Print the range as smallest_element followed by largest_element
END
```

ลูปทำซ้ำ

10.2 แนวคิดการใช้อาร์เรย์ในการประมวลผล

การประมวลผลแบบอาร์เรย์นั้นมีทั้งการใช้อาร์เรย์เดี่ยว ๆ และใช้อาร์เรย์หลาย ๆ อาร์เรย์ให้ทำงานสัมพันธ์กันที่เรียกว่าอาร์เรย์แบบขนาน ลองพิจารณาตัวอย่างการใช้อาร์เรย์ในการประมวลผลในลักษณะต่อไปนี้

ถ้าหากมีร้านขายสินค้าร้านหนึ่ง ได้เก็บรหัสสินค้าไว้เป็นข้อมูลในลักษณะอาร์เรย์ โดยเก็บอยู่ในหน่วยความจำในลักษณะดังต่อไปนี้

รหัสสินค้า
A1
B1
C2
D2

ถ้าหากในการขายสินค้าครั้งหนึ่ง ๆ ผู้ขายได้คีย์รหัสสินค้าเข้ามา เราอาจออกแบบระบบให้อ่านข้อมูลในอาร์เรย์นี้ออกมาทีละเซลเพื่อตรวจสอบว่ารหัสสินค้านั้นเหมือนกับข้อมูลที่เก็บอยู่ในอาร์เรย์นี้หรือไม่ ถ้าหากไม่เหมือนกับเซลใดเลยก็ให้ระบบแจ้งว่าป้อนรหัสผิดได้ นอกจากนี้ถ้าหากป้อนรหัสสินค้าเข้าไปก็สามารถให้เครื่องแจ้งได้ว่าสินค้านั้นเป็นลำดับที่เท่าใด แต่ถ้าให้อาร์เรย์ประมวลผลแบบขนาน โดยมีอาร์เรย์อีกตัวหนึ่งเก็บชื่อสินค้าไว้ ดังรูปต่อไปนี้

รหัสสินค้า
A1
B1
C2
D2

ชื่อสินค้า
คอมพิวเตอร์
เครื่องเล่นเทป
เครื่องรับวิทยุ
นาฬิกา

ถ้าหากการประมวลผลทำในลักษณะของอาร์เรย์คู่ โดยมีจำนวนสมาชิกเท่ากัน และมีความสัมพันธ์กันดังรูป ในอาร์เรย์ของรหัสสินค้าจะทำให้เราทราบลำดับของสินค้าตามที่ผู้ใช้ป้อนรหัสเข้าไป ดังนั้นเราสามารถนำลำดับของสินค้านี้ไปอ้างในอาร์เรย์ชื่อสินค้าได้ ว่าสินค้านี้ชื่ออะไร ตัวอย่างเช่น ถ้าหากป้อนรหัสสินค้าเป็น C2 แล้วให้โปรแกรมไปค้นในอาร์เรย์รหัสสินค้า พบว่าสินค้านี้อยู่ลำดับที่ 3 จากนั้นก็นำค่าลำดับนี้ไปแทนค่าอินเด็กซ์เพื่ออ้างไปที่อาร์เรย์ชื่อสินค้า ก็จะทำให้ทราบได้ว่าสินค้านี้ชื่อว่า “เครื่องรับวิทยุ” ถ้าหากมีอาร์เรย์เพิ่มขึ้นอีก เช่น เก็บราคา เก็บส่วนลด และเก็บข้อมูลอื่น ๆ เราสามารถใช้วิธีการนี้ในการประมวลผลเพื่อหาข้อมูลต่าง ๆ ที่เกี่ยวข้องกันได้

การประมวลผลในลักษณะนี้จะเห็นว่าการค้นหาข้อมูลในอาร์เรย์แรกซึ่งทำโดยการเปรียบเทียบค่าที่ป้อนเข้าไปกับค่าแต่ละเซลของอาร์เรย์ จะทำให้ได้ค่าลำดับออกมา แล้วใช้ลำดับนี้การค้นหาค่าในอาร์เรย์ต่าง ๆ ที่เกี่ยวข้องต่อไป

ในการเขียนโปรแกรมลักษณะนี้จะต้องสร้างตารางสำหรับเก็บข้อมูลขึ้นในหน่วยความจำ สำหรับการสร้างอาร์เรย์หรือตารางสำหรับเก็บข้อมูลต่าง ๆ จะเป็นการจองหน่วยความจำของคอมพิวเตอร์เพื่อใช้ในการเก็บข้อมูล วิธีในการจองหน่วยความจำนี้ขึ้นกับฟังก์ชันของภาษาที่ใช้เขียนโปรแกรม เมื่อมีการจองหน่วยความจำแล้ว การกำหนดค่าข้อมูลลงไปในอาร์เรย์จะทำได้สองวิธี คือให้โปรแกรมประมวลผลแล้วเก็บในอาร์เรย์ เช่นเกิดการการคำนวณต่าง ๆ หรืออ่านขึ้นมาจากไฟล์ หรือรับข้อมูลที่ป้อนเข้าไปทางแป้นพิมพ์เป็นต้น อีกวิธีหนึ่งจะเป็นการกำหนดข้อมูลให้กับอาร์เรย์โดยตรงในโปรแกรม ซึ่งวิธีนี้เหมาะสำหรับข้อมูลที่ไม่มีการเปลี่ยนแปลง เช่น รหัสสินค้า ชื่อสินค้า ตารางเงินเดือน ชื่อเดือน ชื่อวัน เป็นต้น ตัวอย่างเช่นถ้าหากต้องการเก็บชื่อเดือนลงไปในอาร์เรย์ชื่อ month_table จะเขียนเป็นซูโดโค้ดได้ดังนี้

```
Initialise_month_table
     month_table(1) = ‘January’
     month_table(2) = ‘February’
     :
     :
     month_table(12) = ‘December’
END
```

สำหรับกรณีที่ตารางมีการเปลี่ยนแปลงบ่อย เช่น การลดราคาสินค้า ตารางราคาสินค้า ก็อาจใช้วิธีการป้อนข้อมูลเข้าไปเก็บในอาร์เรย์หรือออกแบบให้มีการปรับค่าข้อมูลในอาร์เรย์เมื่อมีการรันโปรแกรมแต่ละครั้งแทน

ตัวอย่างที่ 10.6 โรงงานแห่งหนึ่งจ่ายค่าตอบแทนพนักงานเป็นรายชั่วโมง โดยโรงงานนี้มีพนักงานหลายคน แต่ละคนมีชั่วโมงทำงานไม่เท่ากัน และได้ค่าตอบแทนในอัตรารายชั่วโมงไม่เท่ากัน ถ้าหากโรงงานต้องทำรายการแสดงชั่วโมงการทำงาน และค่าจ้างพนักงานออกมาจะต้องออกแบบโปรแกรมอย่างไร

วิธีทำ งานลักษณะนี้ควรนำการประมวลผลตารางแบบขนานมาใช้ โดยการประกาศตัวแปรอาร์เรย์เพื่อเก็บข้อมูลของพนักงาน ประกอบด้วย อาร์เรย์รหัสพนักงาน, ชั่วโมงทำงาน, ค่าจ้างรายชั่วโมง และค่าตอบแทน ในลักษณะดังต่อไปนี้

ID	H	R	Total
A001	20	50	
A002	30	45	
A003	27	75	
B001	32	30	
B002	19	40	

โดยให้ ID เป็นอาร์เรย์เก็บรหัสพนักงาน
H เป็นอาร์เรย์เก็บจำนวนชั่วโมงทำงาน
R เป็นอาร์เรย์เก็บอัตราค่าจ้างรายชั่วโมง
Total เป็นอาร์เรย์เก็บจำนวนเงินค่าตอบแทน

ขั้นตอนต่อมาให้ระบบรับข้อมูลที่เป็นจำนวนชั่วโมงทำงาน และอัตราค่าจ้างรายชั่วโมงเข้าไปแล้วคำนวณอัตราค่าจ้างรวมออกมา โดยให้อาร์เรย์ Total ในลำดับเดียวกัน มีค่าเท่ากับค่าในอาร์เรย์ H คูณกับค่าในอาร์เรย์ R ในลำดับเดียวกัน เช่นถ้ามีพนักงาน 100 คน ก็เขียนอัลกอริธึมได้ดังนี้

```
FOR   I = 1 TO 100   DO
          Total(I) = H(i) x R(I)
ENDFOR
```

ทำซ้ำ 100 ครั้ง นำค่าในอาร์เรย์ H คูณกับ R แล้วเก็บใน Total

ถ้าหากต้องการทราบว่าพนักงานคนใดได้ค่าตอบแทนรวมเท่าไร ก็ทำโดยการนำรหัสของพนักงานคนนั้นไปค้นในอาร์เรย์ที่เก็บรหัสพนักงาน เมื่อได้ลำดับของพนักงานคนนั้นออกมา ก็นำค่าลำดับนี้ไปเป็นค่าอินเด็กซ์ในการค้นหาค่าตอบแทนในอาร์เรย์ Total ต่อไป

ตัวอย่างที่ 10.7 ถ้าหากมีอาร์เรย์ที่เก็บข้อมูลได้หลายค่า และต้องการเขียนโปรแกรมย่อยในการอ่านข้อมูลจากภายนอกเข้าไปเก็บในอาร์เรย์ โดยอ่านข้อมูลเข้ามาได้เรื่อย ๆ แต่ถ้าหากข้อมูลยังไม่หมด แต่อาร์เรย์เต็มแล้วให้ระบบแจ้งว่าอาร์เรย์มีขนาดเล็กไป จงออกแบบอัลกอริธึมในการประมวลผลโปรแกรมย่อยนี้

วิธีทำ การทำงานในลักษณะนี้อาจทำโดยใช้ลูปแบบ While วนรับข้อมูลเข้ามาเก็บในอาร์เรย์จนกว่าจะหยุดป้อน โดยให้อินเด็กซ์ของอาร์เรย์เริ่มที่ 1 และเพิ่มค่าขึ้นหนึ่งค่าที่ป้อนข้อมูลเข้าไป ซึ่งสามารถเขียนอัลกอริธึมได้ดังนี้

```
Read_values_into_array
        Set max_num_elements to required value          /*กำหนดค่าสูงสุดที่อาร์เรย์สามารถเก็บได้
        Set index to zero
        Read first input value
        WHILE (input values exist) AND (index < max_num_elements)
                index = index + 1
                array(index) = input value              /* เอาค่าไปเก็บในอาร์เรย์ทีละเซล
                Read next input value                   /*อ่านค่าต่อไปมาเก็บในตัวแปร
        ENDWHILE
        IF (input values exit) AND index = max_num_elements THEN
                Print 'Array size too small'
        ENDIF
END
```

ตัวอย่างที่ 10.8 จากตัวอย่างที่ 10.7 ถ้าหากข้อมูลที่ป้อนเข้าไปเป็นตัวเลข ถ้าหากตัวเลขที่ป้อนเข้าไปเป็น 9999 หมายความว่าตัวเลขนี้เป็นข้อมูลสุดท้ายที่จะป้อนเข้าไปในอาร์เรย์ จงเขียนอัลกอริธึมของโปรแกรมย่อยนี้

วิธีทำ ทำได้โดยใช้ลูป WHILE ในการตรวจสอบว่าการรับแต่ละครั้งนั้นค่าที่รับมาเป็น 9999 หรือไม่ ถ้าใช่ให้หยุดรับ แล้วเก็บค่านั้นลงไปในอาร์เรย์ถัดไป สามารถเขียนอัลกอริธึมได้ดังนี้

```
Read_values_into_variable_array
        Set max_num_elements to required value
        Set index to zero
        Read first input value
        WHILE (input values NOT = 9999) AND (index < max_num_elements)
                index = index + 1
                array(index) = input value          /* เอาค่าไปเก็บในอาร์เรย์แต่ละเซล
                Read next input value
        ENDWHILE
        IF index < max_num_elements THEN
                index = index + 1
                array(index) = 9999
        ELSE
                Print 'Array size too small'
        ENDIF
END
```

ทำซ้ำถ้าหากข้อมูลไม่ใช่ 9999 และยังไม่เต็มอาร์เรย์

ถ้าหากต้องการพิมพ์ข้อมูลในอาร์เรย์ทั้งหมดออกมาสามารถเขียนอัลกอริธึมได้ดังนี้

```
Write_values_of_array
    FOR index = 1 to number_of_elements DO
        Print array(index)
    ENDFOR
END
```

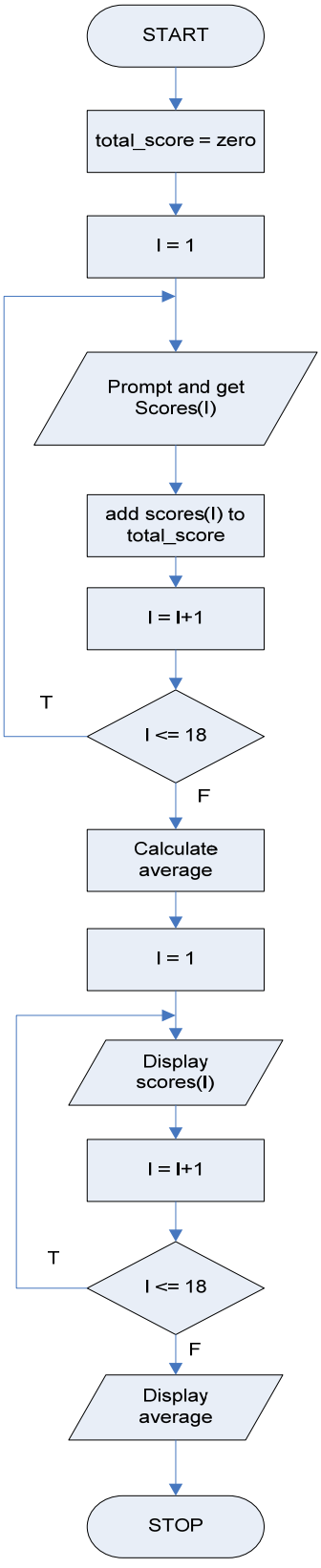
ตัวอย่างที่ 10.9 จงออกแบบแนวทางการพัฒนาโปรแกรม ถ้าหากต้องการให้รับข้อมูลที่เป็นคะแนนของนักเรียนเข้าไป 18 ค่า จากนั้นให้คอมพิวเตอร์คำนวณหาค่าเฉลี่ย แล้วแสดงคะแนนของนักเรียนทั้งหมดออกมาพร้อมกับค่าเฉลี่ย

วิธีทำ

ข้อมูลอินพุต	คะแนนของนักเรียน 18 คน
เอาต์พุตที่ต้องการ	แสดงคะแนนของนักเรียนทั้ง 18 คน และคะแนนเฉลี่ย
การประมวลผล	รับคะแนนเข้าไปทีละคน
	หาคะแนนรวม
	หาคะแนนเฉลี่ย
	แสดงคะแนนทั้งหมด และคะแนนเฉลี่ย

การทำงานในลักษณะนี้ควรนำการเก็บข้อมูลแบบอาร์เรย์มาใช้ ถ้าหากใช้ตัวแปรธรรมดาก็สามารถหาคะแนนเฉลี่ยได้ แต่จะไม่สามารถแสดงคะแนนดิบของนักเรียนทั้งหมดได้ สำหรับการประมวลผลเนื่องจากทราบจำนวนนักเรียนที่แน่นอน ดังนั้นจะใช้ลูปแบบ FOR ได้ โดยจะประกาศตัวแปรอาร์เรย์ชื่อ scores สำหรับเก็บคะแนนนักเรียนแต่ละคน แล้วใช้อินเด็กซ์แทนลำดับที่ของนักเรียนแต่ละคน จากปัญหานี้สามารถเขียนอัลกอริธึมและผังงานได้ดังนี้

```
Process_exam_scores
     Set total_score to zero
     FOR index = 1 to 18
          Prompt operator for score
          Get scores(index)
          total_score = total_score+scores(index)
     ENDFOR
     Compute average_score = total_score / 18
     FOR index = 1 to 18
          Display scores(index)
     ENDFOR
     Display average_score
END
```



ผังงานของโปรแกรมในตัวอย่างที่ 10.9

ตัวอย่างที่ 10.10 ถ้าหากมีข้อมูลอยู่ 100 ค่า เก็บอยู่ในอาร์เรย์ชื่อ number จงอธิบายแนวทางในการพัฒนาโปรแกรมให้คอมพิวเตอร์แสดงคะแนนเฉลี่ย และแจ้งว่ามีข้อมูลกี่ค่าที่มากกว่าคะแนนเฉลี่ย

วิธีทำ

อินพุต	ข้อมูล 100 ค่าเก็บอยู่ในอาร์เรย์ชื่อ number
เอาต์พุต	ต้องการคะแนนเฉลี่ย และจำนวนข้อมูลที่มีค่ามากกว่าค่าเฉลี่ย
วิธีการประมวลผล	อ่านข้อมูลที่เก็บอยู่ในอาร์เรย์ number
	คำนวณหาค่าเฉลี่ย
	นับว่ามีข้อมูลที่มากกว่าค่าเฉลี่ยกี่ค่า
	แสดงค่าเฉลี่ย
	แสดงจำนวนข้อมูลที่มากกว่าค่าเฉลี่ย

การหาค่าเฉลี่ยทำได้ตามวิธีในตัวอย่างที่ 10.9 จากนั้นให้โปรแกรมวนลูปอ่านข้อมูลจากอาร์เรย์อีกครั้ง เพื่อตรวจสอบว่ามีค่าใดบ้างที่มากกว่าค่าเฉลี่ย ถ้าค่าใดมากกว่าให้เพิ่มตัวนับขึ้นหนึ่งค่า ดังนั้นสามารถเขียนอัลกอริธึมและผังงานของปัญหานี้ได้ดังนี้

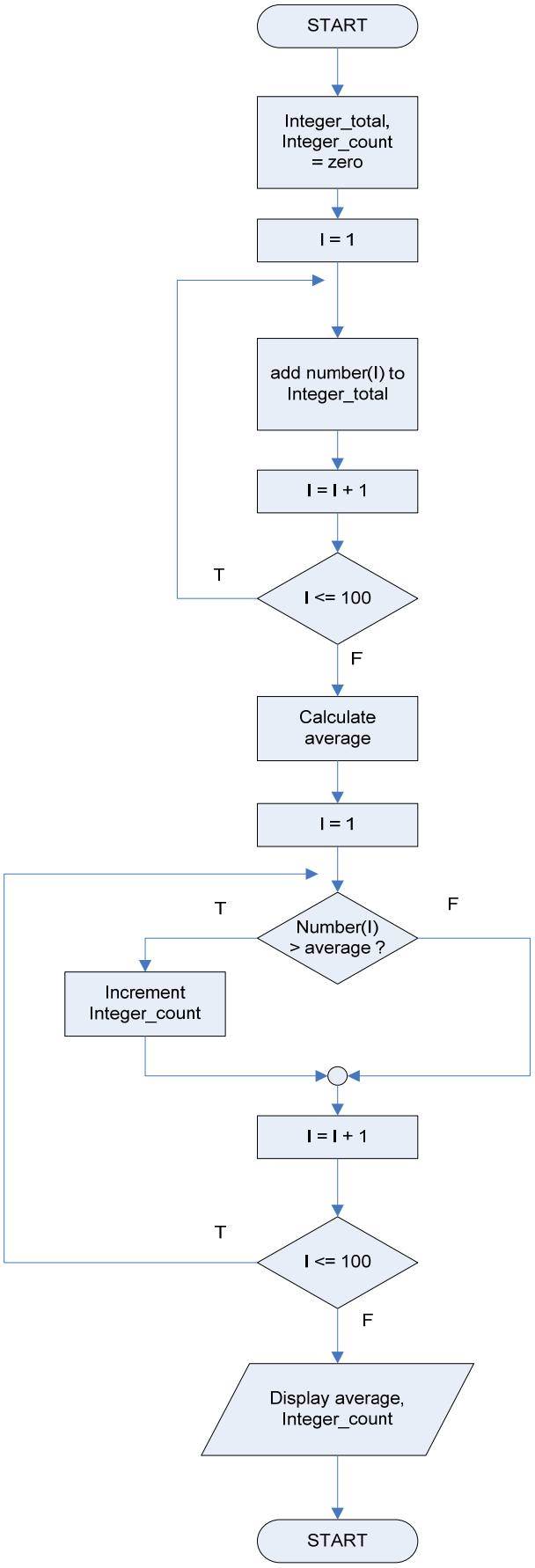
ให้ integer_total เป็นตัวแปรเก็บผลรวม โดยกำหนดค่าเริ่มต้นเป็นศูนย์

ให้ integer_count เป็นตัวแปรเก็บจำนวนที่มากกว่าค่าเฉลี่ย โดยกำหนดค่าเริ่มต้นเป็นศูนย์เช่นกัน

```
Process_integer_array
    Set integer_total to zero
    Set integer_count to zero
    FOR index = 1 to 100
        integer_total = integer_total + number(index)
    ENDFOR
    integer_average = integer_total / 100
    FOR index = 1 to 100
        IF number(index) > integer_average THEN
            add 1 to integer_count
        ENDIF
    ENDFOR
    Display integer_average, integer_count
END
```

วนลูปหาค่าเฉลี่ย

ถ้าหากข้อมูลใดสูงกว่าค่าเฉลี่ย ให้เพิ่มตัวนับ



ผังงานของปัญหาในตัวอย่างที่ 10.10

10.3 ตัวแปรอาร์เรย์ในภาษาซี

การประกาศตัวแปรอาร์เรย์ในภาษาซีนั้นจะเป็นการจองหน่วยความจำของคอมพิวเตอร์ให้เก็บข้อมูลแบบต่อเนื่องกันไป ถ้าหากเป็นอาร์เรย์แบบ 1 มิติจะเก็บข้อมูลต่อเนื่องกันไปเป็นแถว การประกาศตัวแปรจะใช้ชื่อเพียงชื่อเดียวแล้วตามด้วยเครื่องหมาย [ ] คร่อมค่าตัวเลขที่บอกจำนวนของข้อมูลที่ต้องการ โดยมีรูปแบบดังนี้

```
type            var_name[size]          หรือ
ประเภทของข้อมูล ชื่อตัวแปร[จำนวนสมาชิก]
```

โดย type เป็นประเภทของข้อมูลที่ใช้เก็บในอาร์เรย์ var_name เป็นชื่อของตัวแปรอาร์เรย์ และ size เป็นจำนวนเซลข้อมูลในอาร์เรย์ ตัวอย่างเช่น ถ้าต้องการประกาศตัวแปรอาร์เรย์ชื่อ A สำหรับเก็บจำนวนเต็มจำนวน 10 จำนวน สามารถเขียนได้ดังนี้

```
int   A[10];
```

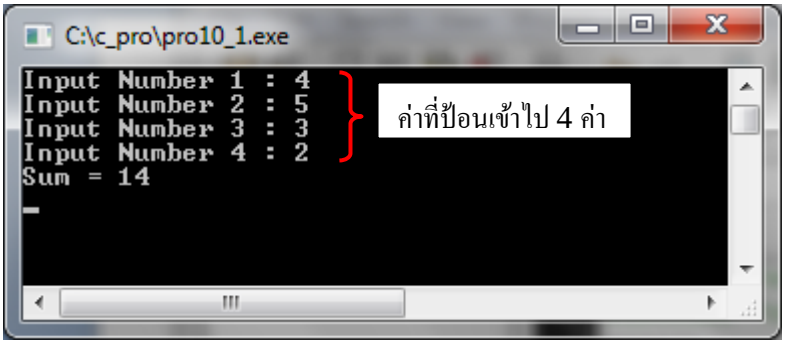
อินเด็กซ์ของอาร์เรย์ในภาษาซีจะเริ่มต้นที่ 0 และเมื่อประกาศตัวแปรอาร์เรย์ขึ้นมาแล้วจะสามารถใช้งานได้เหมือนกับตัวแปรทั่วไป ตัวอย่างเช่น ถ้าหากต้องการใส่ค่า 150 เข้าไปในเซลแรก และใส่ค่า 200 ลงไปในเซลสุดท้ายของอาร์เรย์ A จะเขียนได้ดังนี้

```
A[0]    =    150;
A[9]    =    200;
```

โปรแกรมที่ 10.1 เป็นตัวอย่างที่ประกาศอาร์เรย์สำหรับเก็บข้อมูลจำนวน 4 เซล แล้วให้รับข้อมูลจากแป้นพิมพ์มาเก็บในอาร์เรย์ จากนั้นนำข้อมูลทั้งหมดมารวมกันแล้วแสดงผลรวมออกทางจอภาพ

```
#include "stdio.h"
#include "conio.h"
main()
{
   int A[4];                    /*ตัวแปรอาร์เรย์เก็บข้อมูลได้ 4 เซล */
   int sum = 0;                 /*ตัวแปรเก็บผลรวม */
   printf("Input Number 1 : ");
   scanf("%d",&A[0]);
   printf("Input Number 2 : ");
   scanf("%d",&A[1]);
   printf("Input Number 3 : ");
   scanf("%d",&A[2]);
   printf("Input Number 4 : ");
   scanf("%d",&A[3]);
   sum = A[0] + A[1] + A[2] + A[3];
   printf("Sum = %d\n",sum);
   getch();
}
```

ผลลัพธ์การรันโปรแกรม



จากตัวอย่างโปรแกรมจะพบว่าการใช้คำสั่ง scanf() ในการรับข้อมูลมาเก็บในอาร์เรย์จะใช้ได้คล้ายกับตัวแปรทั่วไป โดยเซลของอาร์เรย์ที่เก็บข้อมูลจะถูกระบุโดยอินเด็กซ์

ในการประกาศตัวแปรอาร์เรย์นั้นเราสามารถกำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์ได้ โดยมีรูปแบบดังนี้

type array_name[size] = {value-list}

โดยข้อมูลที่กำหนดต้องอยู่ในเครื่องหมายปีกกา ถ้าหากมีข้อมูลหลายตัวจะใช้เครื่องหมาย , คั่น ตัวอย่างเช่น ถ้าหากต้องการกำหนดค่าเลขจำนวนเต็มให้กับตัวแปรอาร์เรย์ชื่อ A จำนวน 5 ตัว สามารถเขียนได้ดังนี้

```
int A[5] = {1, 8, 4, 9, 7};
```

หรือถ้าหากต้องการกำหนดค่าให้กับอาร์เรย์เป็นตัวอักขระสามารถทำได้ดังนี้

```
char A[3] = {'A', 'B', 'C'};
```

โปรแกรมที่ 10.2 จงแสดงแนวคิดและเขียนโปรแกรมรับจำนวนเต็มเข้ามา 10 ค่า จากนั้นให้หาค่าเฉลี่ย แล้วแสดงออกทางจอภาพ

วิธีทำ ปัญหานี้สามารถเขียนอัลกอริธึมได้คล้ายกับตัวอย่างที่ 10.9 โดยวิเคราะห์ปัญหาได้ดังนี้

ข้อมูลอินพุต รับข้อมูลจำนวนเต็มเข้ามา 10 ค่า

เอาต์พุตที่ต้องการ แสดงผลค่าเฉลี่ยออกทางจอภาพ

การประมวลผล รับคะแนนเข้าไปทีละค่าจนครบ 10 ค่า
นำคะแนนทั้งหมดมาบวกกัน
หาคะแนนเฉลี่ย แล้วแสดงผล

สำหรับโปรแกรมภาษาซีเขียนได้ดังนี้

```
#include "stdio.h"
#include "conio.h"
main()
{
   int number[10], i, sum = 0;          /*ประกาศตัวแปรอาร์เรย์ ตัวแปรสำหรับนับลูป และตัวแปรเก็บผลรวม */
   for(i = 0; i<10; i++)
   {
      printf("Input Number %d : ",i);
      scanf("%d",&number[i]);
   }
   for(i = 0; i<10; i++)
      sum = sum + number[i];
   printf("The average is %.2f\n",sum/10.0);     /*หาคะแนนเฉลี่ย แล้วแสดงผล */
   getch();
}
```

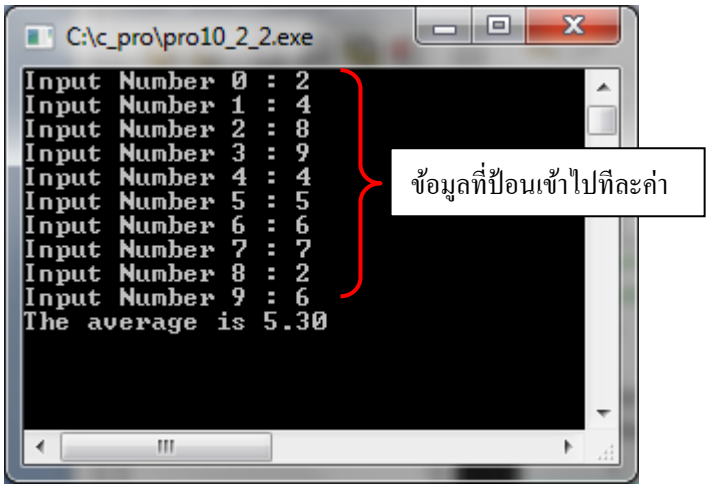
รับคะแนนเข้าไปทีละค่าจนครบ 10 ค่า

นำคะแนนทั้งหมดมาบวกกัน

จากการเขียนโปรแกรมจะพบว่าการเก็บตัวเลขทั้งสิบค่านั้นจะนำตัวแปรแบบอาร์เรย์มาใช้ จากนั้นเขียนโปรแกรมแบบวนลูปรับข้อมูลเข้าไปเก็บทีละค่า จากนั้นวนลูปนำข้อมูลทั้งหมดมาบวกกัน แล้วนำผลลัพธ์ที่ได้มาหารด้วย 10 เพื่อหาค่าเฉลี่ย จากปัญหานี้สามารถเขียนโปรแกรมให้สั้นลงได้โดยรวบการคำนวณหาผลรวมไว้ในลูปของการรับข้อมูล ทำให้โปรแกรมทำลูปเพียงครั้งเดียว โดยโปรแกรมใหม่เขียนได้ดังนี้

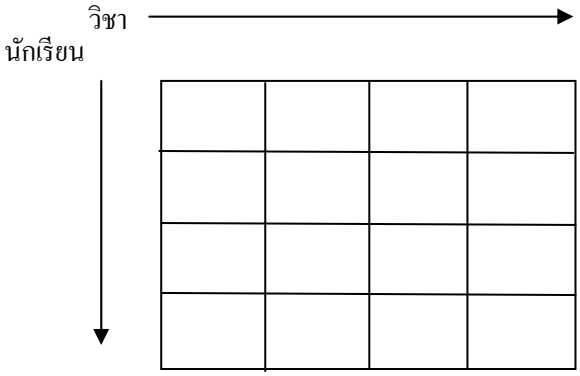
```
#include "stdio.h"
#include "conio.h"
main()
{
    int number[10], i, sum = 0;
    for(i = 0; i<10; i++)
    {
        printf("Input Number %d : ",i);
        scanf("%d",&number[i]);
        sum = sum + number[i];
    }
    printf("The average is %.2f\n",sum/10.0);
    getch();
}
```

รับข้อมูลเข้ามาแล้วหาผลรวมทันที



ตัวแปรอาร์เรย์แบบ 2 มิติ

อาร์เรย์แบบ 2 มิติ จะเป็นการเก็บข้อมูลในแนวแถวและหลัก การอ้างถึงเซลในอาร์เรย์จะต้องใช้อินเด็กซ์ที่อ้างไปยังแถวและหลัก การเก็บข้อมูลบางประเภทนั้นตัวแปรอาร์เรย์แบบมิติเดียวจะทำงานไม่สะดวกซึ่งอาจต้องใช้ตัวแปรอาร์เรย์แบบ 2 มิติ เช่นการเก็บคะแนนของนักศึกษา 4 คน แต่ละคนมีคะแนน 4 วิชา โดยโครงสร้างการเก็บข้อมูลเป็นดังรูปต่อไปนี้



การเก็บข้อมูลของนักเรียนแต่ละคนจะเก็บในแนวแถว ส่วนการเก็บคะแนนแต่ละวิชาจะเก็บในแนวหลัก การประกาศตัวแปรอาร์เรย์แบบ 2 มิตินี้ จะใช้ดัชนี 2 ตัว เพื่อระบุจำนวนสมาชิกในแต่ละแถวและหลัก ดังรูปแบบต่อไปนี้

```
type    array_name{Row] [ Column]
```

ตัวอย่างเช่น

```
int  AB[2][3]
```

หมายความว่าเป็นการประกาศตัวแปรชื่อ AB สำหรับใช้เก็บเลขจำนวนเต็ม มีสมาชิกทั้งหมด 6 ตัว (2 x 3) การอ้างสมาชิกแต่ละตัวทำได้ดังนี้

แถวที่ 0 AB[0][0] , AB[0][1] , AB[0][2]
แถวที่ 1 AB[1][0] , AB[1][1] , AB[1][2]

ตัวแปรอาร์เรย์แบบ 2 มิตินี้ สามารถกำหนดค่าเริ่มต้นให้กับอาร์เรย์ได้เช่นกัน ตัวอย่างเช่น

```
int  sqr[3][3]      =       {
                                    1 , 2 , 3,
                                    4 , 5 , 6 ,
                                    7 , 8 , 9
                            };
```

เป็นการประกาศตัวแปรชื่อ sqr เป็นตัวแปรอาร์เรย์แบบ 2 มิติสำหรับเก็บเลขจำนวนเต็ม โดย sqr[0][0] เก็บค่า 1 sqr[0][1] เก็บค่า 2 sqr[0][2] เก็บค่า 3 ตามลำดับ

แบบฝึกหัดท้ายบทที่ 10

ตอนที่ 1 จงตอบคำถามต่อไปนี้พอสังเขป

1. ท่านคิดว่าขนาดของตัวแปรอาร์เรย์หาได้อย่างไร
2. งานประเภทใดที่ควรใช้การเก็บข้อมูลแบบอาร์เรย์
3. งานในลักษณะใดควรใช้การประมวลผลอาร์เรย์แบบขนาน
4. จงเปรียบเทียบข้อดีข้อเสียระหว่างการเก็บข้อมูลโดยใช้ตัวแปรทั่วไป กับการประกาศตัวแปรแบบอาร์เรย์
5. จากตัวอย่างที่ 10.6 ถ้าไม่ใช่หน่วยความจำแบบอาร์เรย์จะได้หรือไม่ และเขียนอัลกอริธึมได้อย่างไร

ตอนที่ 2 แบบฝึกหัดฝึกทักษะการเขียนรหัสเทียมสำหรับการประมวลผลแบบอาร์เรย์

1. จงเขียนรหัสเทียมสำหรับรับข้อมูลเข้าไปเก็บจำนวน 10 ค่า จากนั้นให้คอมพิวเตอร์แสดงค่าทั้งหมด ค่าผลรวมของข้อมูล ค่าสูงสุด และค่าต่ำสุด
2. จากข้อ 1 ถ้าหากไม่ใช้อาร์เรย์จะเขียนรหัสเทียมได้อย่างไร และมีส่วนใดที่ทำไม่ได้บ้าง

3. ถ้าอาร์เรย์ A เป็นตัวแปรที่มี 20 เซล จงเขียนซูโดโค้ดสำหรับเก็บตัวอักขระภาษาอังกฤษ 20 ตัวแรกลงไป พร้อมทั้งนำออกมาแสดงผล
4. ร้านค้าร้านหนึ่งจะมีส่วนลดให้ลูกค้าตามยอดที่ซื้อดังนี้

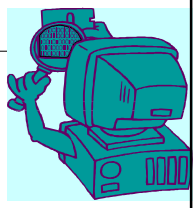
ยอดสินค้าไม่เกิน 10,000	ลด 5 %
ยอดสินค้ามากกว่า 10,000 แต่ไม่เกิน 30,000	ลด 10 %
ยอดสินค้าเกิน 30,000	ลด 15 %

ถ้าหากโปรแกรมรับยอดซื้อเข้าไป แล้วแสดงยอดเงินที่มีส่วนลดแล้วออกมา จงเขียนซูโดโค้ดแบบใช้อาร์เรย์ และไม่ใช้อาร์เรย์

5. นักเรียนห้องหนึ่งมี 40 คน เรียนวิชาฟิสิกส์ คณิตศาสตร์ และเคมี จงออกแบบแนวทางการพัฒนาโปรแกรม ที่สามารถเก็บคะแนนทั้ง 3 วิชาของนักเรียนเข้าไป แล้วแสดงผลเป็นคะแนนรวมของคะแนนแต่ละคนได้
6. จากข้อ 5 จงออกแบบแนวทางการเขียนโปรแกรมย่อย ถ้าหากต้องการทราบว่าผู้ที่ได้คะแนนสูงสุดคือใคร คะแนนต่ำสุดคือใคร และใครบ้างที่มีคะแนนสูงกว่าค่าเฉลี่ย

ภาษาโปรแกรม

(ภาษาซี)



1. ประวัติความเป็นมา

ภาษา **BCPL** ➡ ภาษา **B** ➡ ภาษา **C**

Basic Combined Programming Language

บนเครื่อง **PDP-7 (UNIX)** พ.ศ. 2513

พ.ศ. 2515 โดย เดนนิช ริทชี่

2. โครงสร้างพื้นฐานของภาษาซี

```
# header                                  ส่วนที่ 1
main( )
{                  /* เริ่มโปรแกรม */
   declaration     ส่วนที่ 2
   .....
   คำสั่งต่าง ๆ                           ส่วนที่ 3
}
```


ส่วนที่ 1 เป็นส่วนที่ระบุให้ซีคอมไพเลอร์เตรียมการทำงานที่กำหนดในส่วนนี้ไว้ โดยหน้าคำสั่งจะมีเครื่องหมาย # เช่น

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

เป็นการระบุให้นำไฟล์ stdio.h มารวมกับไฟล์นี้ เพื่อที่จะสามารถใช้คำสั่งที่อยู่ในไฟล์นี้มาใช้งานได้

หรือ

```
# define START 0
```

เป็นการกำหนดค่าคงที่ให้กับตัวแปร START โดยให้มีค่าเป็น 0

หรือ

```
# define temp 37
```

เป็นการกำหนดให้ตัวแปร temp มีค่าเท่ากับ 37

ส่วนที่ 2 declaration เป็นการกำหนดชนิดข้อมูลที่จะใช้ในโปรแกรมซึ่งตัวแปรหรือข้อมูลต่าง ๆ นั้นจะต้องถูกประกาศ(declare) ในส่วนนี้ก่อน จึงจะสามารถนำไปใช้ในโปรแกรมได้ เช่น

```
int stdno;
```

เป็นการกำหนดว่าตัวแปร stdno เป็นข้อมูลชนิดจำนวนเต็ม หรือ interger ซึ่งอาจได้แก่ค่า 0,4,-1,-3,.... เป็นต้น

```
float  score;
```

เป็นการกำหนดว่าตัวแปร score เป็นข้อมูลชนิดเลขมีจุดทศนิยม (floating point)ซึ่งอาจมีค่า 0.23, 1.34, -21.002, เป็นต้น

ส่วนที่ 3 Body คือส่วนของตัวโปรแกรม โดยจะต้องเริ่มต้นด้วยฟังก์ชัน main ( ) แล้วใส่เครื่องหมายกำหนดขอบเขตเริ่มต้นของตัวโปรแกรมคือ { หลังจากนั้นใส่คำสั่งหรือฟังก์ชันต่าง ๆ โดยแต่ละคำสั่งหรือฟังก์ชันนั้น ๆ จะต้องปิดด้วยเครื่องหมาย ; เมื่อต้องการจบโปรแกรมให้ใส่เครื่องหมาย } ปิดท้าย เช่น

```
main ( )
{        /* เริ่มต้นโปรแกรม */
    คำสั่งต่าง ๆ ;
    ฟังก์ชัน;
     ..............
     ..............
}        /* จบโปรแกรม */
```

เครื่องหมายต่าง ๆ

{ } - เป็นตัวกำหนดขอบเขตหรือบล็อกของฟังก์ชัน

( ) - เป็นการระบุตัวผ่านค่าหรืออาร์กิวเมนต์ให้กับฟังก์ชัน ถ้าภายในวงเล็บไม่มีข้อความใด ๆ แสดงว่าไม่มีตัวผ่านค่าที่ต้องการระบุสำหรับฟังก์ชันนั้น ๆ

/* */ - เป็นการกำหนด comment หรือข้อความ ที่ไม่ต้องการให้คอมไพเลอร์ปฏิบัติงาน ซึ่งข้อความที่อยู่ภายในเครื่องหมายนี้จะถือว่า ไม่ใช่คำสั่งปฏิบัติงาน

ตัวอย่างโปรแกรม

โปรแกรมที่ 1

```
# include <stdio.h>
int main (void )
{
    printf("Hello, Good morning. \n");
}
```

เป็นโปรแกรมสั่งพิมพ์ข้อความ Hello, Good morning.

โปรแกรมที่ 2

```
# include <stdio.h>
main ( )
{
    float   point;
    printf("\n\nPut your score in\n");
    scanf("%f", &point);
    printf("Your score is %f point\n\n", point);
}
```

เป็นโปรแกรมรับคะแนนและเก็บค่าที่ตัวแปร **point** หลังจากนั้นสั่งให้มีการพิมพ์คะแนนออกมา

ชนิดของข้อมูลและตัวแปรในภาษาซี

การกำหนดชื่อตัวแปร หลักการมีดังนี้

1. ต้องขึ้นต้นด้วยตัวอักษร
2. ห้ามใช้เครื่องหมายทางคณิตศาสตร์ในชื่อตัวแปร
3. สามารถใช้เครื่องหมาย underline '_' ได้
4. ห้ามใช้ reserved words เช่น int, float, etc.

Note: คอมไพเลอร์ในภาษาซีสามารถเห็นความแตกต่างของชื่อตัวแปรได้ยาวไม่เกิน 8 ตัวอักษร และชื่อตัวแปรจะแตกต่างกันถ้าใช้รูปแบบของตัวอักษรต่างกัน

แบบข้อมูลและขนาด

แบบข้อมูลหรือชนิดของตัวแปรต่าง ๆ ที่กำหนดไว้ในภาษาซี

char	ชนิดของตัวอักษรหรืออักขระ
int	ชนิดจำนวนเต็มปกติ
short	ชนิดจำนวนเต็มปกติ
long	ชนิดจำนวนเต็มที่มีความยาวเป็น 2 เท่า
unsigned	ชนิดของเลขที่ไม่คิดเครื่องหมาย
float	ชนิดเลขมีจุดทศนิยม
double	ชนิดเลขที่มีจุดทศนิยมความยาวเป็น 2 เท่า

ตารางแสดงเนื้อที่ในหน่วยความจำและค่าตัวเลขที่เก็บของข้อมูลแต่ละชนิด

ชนิดข้อมูล	เนื้อที่สำหรับเก็บ(ไบต์)	ค่าตัวเลขที่เก็บ
Char	1	เก็บตัวอักษร ASCII ได้ 1 ตัวหรือจำนวนเต็มระหว่าง 0 ถึง 255
Int	2	ค่าตัวเลขระหว่าง –32768 ถึง 32767
Short	2	ค่าตัวเลขระหว่าง –32768 ถึง 32767
Long	4	ค่าตัวเลขประมาณ ±2000 ล้าน
Unsigned	Unsigned short = 2 Unsigned long = 4	ค่าตัวเลขระหว่าง 0 ถึง 65535 ค่าตัวเลขระหว่าง 0 ถึง 4000 ล้าน
Float	4	ได้ค่าตัวเลขยกกำลัง 10^x โดย x มีค่าระหว่าง –37 ถึง +38
Double	8	ความถูกต้องของตัวเลขจะมีค่าสูงขึ้น

ในการเขียนโปรแกรม แบบข้อมูลที่ใช้จะแบ่งออกเป็น 4 กลุ่มใหญ่ ดังนี้

- ข้อมูลและตัวแปรชนิดอักขระ
- ข้อมูลและตัวแปรชนิดจำนวนเต็ม
- ข้อมูลและตัวแปรชนิดเลขมีจุดทศนิยม
- ข้อมูลและตัวแปรแบบสตริง

ข้อมูลและตัวแปรชนิดอักขระ

1 อักขระแทนด้วย char โดยอยู่ภายในเครื่องหมาย ' ' เช่น

```
# include <stdio.h>
main ( )
{
    char   reply;

    reply = 'y';
    ……….
}
```


การให้ค่าอักขระที่เป็นรหัสพิเศษหรือรหัสควบคุม

อักขระเหล่านี้ไม่สามารถให้ค่าโดยตรง แต่จะทำได้โดยการให้ค่าเป็นรหัส ASCII ซึ่งจะเขียนในรูปของเลขฐานแปด โดยใช้เครื่องหมาย '\' นำหน้า หรือใช้ตัวอักขระที่กำหนดให้กับรหัสนั้น ๆ เขียนตามเครื่องหมาย '\' สำหรับรหัสบางตัว เช่น

รหัส BELL แทนด้วย ASCII 007 ซึ่งกำหนดได้ดังนี้

```
beep  = '\007';
```

หรือรหัสควบคุมการขึ้นบรรทัดใหม่ ตัวอักขระที่กำหนดให้กับรหัสคือ n

สามารถกำหนดเป็น newline = '\n';

ตัวอย่างโปรแกรม

```
# include <stdio.h>
main ( )
{
   char  newline;

    newline = '\n';
    printf("Hello, Good morning. %c",newline);
    printf("Hello, Good morning.\n");
}
```


ข้อมูลและตัวแปรชนิดจำนวนเต็ม

จำนวนเต็มในภาษาซีสามารถใช้แทนได้ 4 รูปแบบคือ int, short, long และ unsigned

สำหรับการกำหนดตัวแปรแบบ unsigned คือจำนวนเต็มที่ไม่คิดเครื่องหมายนั้นจะต้องใช้ควบคู่กับรูปแบบข้อมูลจำนวนเต็มชนิดอื่น ๆ คือ int หรือ short หรือ long ตัวอย่างเช่น

```
unsigned  int   plusnum;
unsigned long  width;
unsigned short absno;  /* absolute number */
```


ข้อมูลและตัวแปรชนิดเลขมีจุดทศนิยม

สำหรับเลขมีจุดทศนิยมนั้นแทนได้ 2 แบบคือ float และ double โดย double เก็บค่าได้เป็น 2 เท่าของ float

สำหรับงานทางวิทยาศาสตร์ที่ต้องการความละเอียดในการเก็บค่ามักใช้การเก็บในรูปแบบนี้ คือเก็บแบบเอ็กโพเนนซ์ ดังตัวอย่างต่อไปนี้

ตัวเลข	แสดงแบบวิทยาศาสตร์	แบบเอ็กโพเนนซ์
9,000,000,000	$9.0*10^{9}$	9.0e9
345,000	$3.45*10^{5}$	3.45e5
0.00063	$6.3*10^{-4}$	6.3e-4
0.00000924	$9.24*10^{-6}$	9.24e-6

ข้อมูลและตัวแปรแบบสตริง

สตริงหมายถึงตัวอักขระหลาย ๆ ตัวมาประกอบกันเป็นข้อความ ซึ่งการที่นำตัวแปรหลาย ๆ ตัวมาเก็บรวมกันในภาษาซีนี้เรียกว่า อะเรย์ (array) ดังนั้นข้อมูลแบบสตริงคือ อะเรย์ของตัวอักขระ นั่นเอง

เครื่องหมายของอะเรย์คือ [ ] รูปแบบการกำหนดสตริงจึงมีลักษณะดังนี้

```
char    name[30];
```

หมายถึง ตัวแปร name เป็นชนิด char ที่มีความยาว 30 ตัวอักษร โดยเก็บเป็น อะเรย์ การเก็บนั้นจะเก็บเรียงกันทีละไบต์ และไบต์สุดท้ายเก็บรหัส null คือ \0 ดังนั้นจะเก็บได้จริงเพียง 29 ตัวอักษร

การกำหนดค่าให้ตัวแปรและการส่งผลลัพธ์

การกำหนดค่าให้ตัวแปรอาจทำได้โดยกำหนดในโปรแกรม หรือกำหนดในขณะที่มีการกำหนดชนิดก็ได้ เช่น

```
main ( )
{
   int age = 18;
   float height;

   height = 172.5;
   printf("Mr. Surasak is %d years old",age);
   printf(" and tall %f cms.\n",height);
}
```


ตัวอย่างของโปรแกรมในการกำหนดค่าและส่งค่าผลลัพธ์

```
# include <stdio.h>
main ( )
{
   int sum,valuea;
   int count = 1;

   valuea = 4;
   sum = count + valuea;
   printf("Total value is %d.\n",sum);
}
```

ผลลัพธ์จะปรากฏข้อความ : Total value is 5.

ฟังก์ชัน printf( ) และ scanf( )

รูปแบบของ printf ( )

printf(ส่วนควบคุมการพิมพ์, อาร์กิวเมนต์, อาร์กิวเมนต์,...)

ส่วนควบคุมการพิมพ์ เป็นสตริงที่มีข้อความและรูปแบบของการพิมพ์โดยอยู่ในเครื่องหมาย " "

อาร์กิวเมนต์ เป็นส่วนที่จะนำข้อมูลมาพิมพ์ ตามรูปแบบที่กำหนดมาในส่วนควบคุมการพิมพ์

รูปแบบที่ใช้สำหรับกำหนดการพิมพ์ในฟังก์ชัน printf

%d	พิมพ์ด้วยเลขฐานสิบ
%o	" " เลขฐานแปด
%x	" " เลขฐานสิบหก
%u	" " เลขฐานสิบแบบไม่คิดเครื่องหมาย
%e	" " ตัวเลขแบบวิทยาศาสตร์ เช่น 2.13e45
%f	" " ตัวเลขมีจุดทศนิยม
%g	" " รูปแบบ %e หรือ %f โดยเลือกแบบที่สั้นที่สุด

สำหรับสตริงมีรูปแบบการพิมพ์ดังนี้

%c	พิมพ์ด้วยตัวอักษรตัวเดียว
%s	" "ข้อความ

เครื่องหมายสำหรับปรับเปลี่ยนรูปแบบของข้อมูล

เครื่องหมายลบ	ให้พิมพ์ข้อมูลชิดขอบซ้าย (ปกติข้อมูลทั้งหมดจะพิมพ์ชิดขวา)
สตริงตัวเลข	ระบุความกว้างของฟิลด์
จุดทศนิยม	เป็นการกำหนดความกว้างของจุดทศนิยม

Note การปรับเปลี่ยนรูปแบบของข้อมูลนี้ทำได้โดย การใส่เครื่องหมายเหล่านี้ระหว่างเครื่องหมาย % และเครื่องหมายที่กำหนดรูปแบบการพิมพ์

รูปแบบของ scanf ( )

scanf(ส่วนควบคุมข้อมูล, อาร์กิวเมนต์, อาร์กิวเมนต์,...)

ส่วนควบคุมข้อมูล เป็นการกำหนดรูปแบบข้อมูลในเครื่องหมาย " "

อาร์กิวเมนต์ เป็นส่วนที่จะนำข้อมูลมาเก็บ(ในตัวแปร) ซึ่งชนิดของข้อมูลต้องตรงตามรูปแบบที่กำหนดในส่วนควบคุมข้อมูล

การกำหนดลักษณะอาร์กิวเมนต์มีได้ 2 แบบดังนี้

ถ้าข้อมูลนั้นอาจจะนำไปใช้ในการคำนวณ

- จะใส่เครื่องหมาย & หน้าตัวแปร

ถ้าข้อมูลนั้นเป็นข้อความที่จะนำไปเก็บไว้ในตัวแปรเลย

- ไม่จำเป็นต้องใส่เครื่องหมาย & หน้าตัวแปร

โอเปอเรเตอร์และนิพจน์

การแทนโอเปอเรเตอร์ทางคณิตศาสตร์สำหรับภาษาซี

+	การบวก
-	การลบ
*	การคูณ
/	การหาร
%	การหารเอาเศษ (โมดูลัส)

การเปลี่ยนชนิดของข้อมูล

ทำได้โดยระบุชนิดที่ต้องการเปลี่ยนภายในเครื่องหมาย ( ) แล้ววางหน้าตัวแปรหรือข้อมูลที่ต้องการเปลี่ยนแปลงชนิด

```
float  money;
```

ต้องการเปลี่ยนตัวแปร float ไปเป็น integer ทำได้ดังนี้

```
(int) money;
```

```
int cost;
    cost = 2.7+4.5;
    cost = (int)2.7+(int)4.5;
```


การเพิ่มค่าและลดค่าตัวแปร

++n	เพิ่มค่า n อีก 1
- -n	ลดค่า n ลง 1

ความแตกต่างระหว่าง count++ และ ++count

เช่น

```
count = 5;
x  = count++;
```

จะได้ค่า x เท่ากับ 5 แล้วค่า count เท่ากับ 6

```
count = 5;
x  = ++count;
```

จะได้ค่า x เท่ากับ 6

นิพจน์กำหนดค่า (Assignment expression)

เครื่องหมายที่ใช้กำหนดค่าคือ =

โดยเป็นการกำหนดค่าทางขวาของเครื่องหมาย ให้กับตัวแปรที่อยู่ทางซ้าย เช่น j = 7+2

หรือ k = k+4

3.4.6 เครื่องหมายและนิพจน์เปรียบเทียบ

> หรือ >=	มากกว่า หรือ มากกว่าเท่ากับ
< หรือ <=	น้อยกว่า หรือ น้อยกว่าเท่ากับ
==	เท่ากับ
!=	ไม่เท่ากับ

ความแตกต่างของเครื่องหมาย = และ ==

เครื่องหมาย = เป็นตัวกำหนดค่า

ในขณะที่เครื่องหมาย == เป็นเครื่องหมายเปรียบเทียบ ตัวอย่างเช่น

point = 44;

หมายถึง เป็นการกำหนดค่าให้กับตัวแปร point ให้มีค่าเท่ากับ 44

point == 44;

หมายถึง เป็นการตรวจสอบว่าค่า **point** มีค่าเท่ากับ 44 หรือไม่

เครื่องหมายและนิพจน์เปรียบเทียบแบบตรรกศาสตร์

&&	และ (and)
||	หรือ (or)
!	ไม่ (not)

ค่าของนิพจน์เปรียบเทียบเชิงตรรก

นิพจน์ที่ 1 && นิพจน์ที่ 2 เป็นจริง เมื่อนิพจน์ทั้งสองเป็นจริง

นิพจน์ที่ 1 || นิพจน์ที่ 2 เป็นจริง เมื่อนิพจน์ใดนิพจน์หนึ่งเป็นจริงหรือ ทั้งสองนิพจน์นั้นเป็นจริง

! นิพจน์เปรียบเทียบ เป็นจริง เมื่อนิพจน์เปรียบเทียบเป็นเท็จ

คำสั่ง if

รูปแบบของคำสั่ง

```
if (เงื่อนไข)
    คำสั่งที่ต้องทำ ถ้าเงื่อนไขนั้นเป็นจริง;
```

ตัวอย่างเช่น

```
if (score >= 80)
   grade = 'A'; /* simple statement */
```

หรือ

```
if (math >= 60 && eng >= 55)
{   grade = 'S'; /* compound statement */
    printf("Your grade is %c\n",grade);
}
```


คำสั่ง if ….. else …..

รูปแบบของคำสั่ง

```
if (คำสั่งหรือนิพจน์เงื่อนไข)
        คำสั่งที่ต้องทำเมื่อเงื่อนไขนั้นเป็นจริง
else    คำสั่งที่ต้องทำเมื่อเงื่อนไขนั้นไม่เป็นจริง
```

ตัวอย่างเช่น

```
if (value1  > value2)
     min = value2;
else
     min = value1;
```


เครื่องหมายพิเศษที่ใช้ในการเปรียบเทียบเงื่อนไข ? :

รูปแบบทั่วไปของคำสั่งเปรียบเทียบเงื่อนไข ? : มีดังนี้

นิพจน์ที่ 1 ? นิพจน์ที่ 2 : นิพจน์ที่ 3

ความหมายคือ

```
if  นิพจน์ที่ 1 เป็นจริง
      ทำตามคำสั่งในนิพจน์ที่ 2
else
      ทำตามคำสั่งในนิพจน์ที่ 3
```

เช่น `x = (y< 0) ? -y : y;`

คำสั่งตรวจสอบเงื่อนไขหลาย ๆ ทาง : switch และ break

รูปแบบคำสั่ง

```
switch (นิพจน์)
    {
      case  label1   : statement1;
        case  label2   : statement2;
       ……..
       ……..
      default        :  statementn;
     }
```


ตัวอย่าง

```
switch  (ch)
{
     case '1'  :
          printf("Red\n");
     case '2'  :
          printf("Blue\n");
     case '3'  :
          printf("Yellow\n");
     default   :
          printf("White\n");
}
```

ตัวอย่าง

```
switch   (ch)
{
        case '1' : printf("Red\n");
             break;
        case '2' : printf("Blue\n");
             break;
        case '3' : printf("Yellow\n");
                   break;
        default :  printf("White\n");
   }
```


คำสั่ง loop หรือคำสั่งวนซ้ำ

คำสั่งลูป while

รูปแบบ

```
while   (นิพจน์เงื่อนไข)
{
    คำสั่งที่วนลูป;
    ……
    ……
}
```

compound statements

คำสั่งลูป for

รูปแบบ

```
for ( นิพจน์ที่ 1  ;  นิพจน์ที่ 2 ;  นิพจน์ที่ 3 )
{
    คำสั่งวนรอบ;
    …….
}
```

เป็นคำสั่งที่ใช้ในการควบคุมให้มีการวนรอบคำสั่งหลาย ๆ รอบ โดยนิพจน์ที่ 1 คือการกำหนดค่าเริ่มต้นให้กับตัวแปรที่ใช้ในการวนรอบ นิพจน์ที่ 2 เป็นการเปรียบเทียบ ก่อนที่จะวนรอบถ้าเงื่อนไขของนิพจน์เป็นจริงจะมีการทำงานตามคำสั่งวนรอบ นิพจน์ที่ 3 เป็นคำสั่งในการกำหนดค่าที่จะเปลี่ยนแปลงไปในแต่ละรอบ

คำสั่งวนรอบแบบที่ตรวจสอบเงื่อนไขทีหลัง : do while

รูปแบบ

```
do
    statement;
while (นิพจน์เงื่อนไข);
```

เช่น

```
num = 2;
do
   {
      num++;
      printf("Now no is %d\n",num);
   } while (num == 10)
```


คำสั่งควบคุมอื่น ๆ break, continue, goto และ labels

คำสั่ง break

ใช้เมื่อต้องการให้การทำงานสามารถหลุดออกจากลูปและกระโดดไปยังคำสั่งที่อยู่นอกลูปทันที โดยไม่ต้องตรวจสอบเงื่อนไขใด ๆ

คำสั่ง continue

ใช้เมื่อต้องการให้การทำงานนั้น ย้อนกลับไปวนรอบใหม่อีกครั้ง ซึ่งมีลักษณะที่ตรงข้ามกับคำสั่ง break

คำสั่ง goto และ labels

คำสั่ง goto ประกอบด้วย 2 ส่วน คือ

- ตัวคำสั่ง goto เป็นคำสั่งให้กระโดดไปยังตำแหน่งที่กำหนด โดยจะกำหนดเป็นชื่อ เรียกว่า label name
- ชื่อ (label name) เป็นตัวกำหนดตำแหน่งที่คำสั่งจะกระโดดไปทำงาน

ข้อควรระวัง ! คำสั่งนี้ถือเป็นคำสั่งที่ควรหลีกเลี่ยงในการเขียนโปรแกรม แต่ถ้าจำเป็นหรือหลีกเลี่ยงไม่ได้เท่านั้น จึงจะใช้คำสั่งนี้

ตัวอย่างโปรแกรมที่ใช้คำสั่ง goto

```
#include<stdio.h>
main()
{   int sum,n;

    for(n=1;n<10;n++)
        if (n==5)
                goto part1;
        else printf("%d\n",n);
    part1 :  printf("Interrupt with no. 5\n");
}
```


ฟังก์ชัน (Function)

ฟังก์ชัน (Functions)

การออกแบบโปรแกรมในภาษาซีจะอยู่บนพื้นฐานของการออกแบบโมดูล (Module Design) โดยการแบ่งโปรแกรมออกเป็นงานย่อย ๆ (หรือโมดูล) แต่ละงานย่อยจะทำงานอย่างใดอย่างหนึ่งเท่านั้น และไม่ควรจะมีขนาดใหญ่จนเกินไป งานย่อยเหล่านี้เมื่อนำไปเขียนโปรแกรมในภาษาซีจะเป็นการเขียนในลักษณะของฟังก์ชัน

ตัวอย่าง โปรแกรมเพื่อบวกเลขสองจำนวนที่รับจากผู้ใช้และแสดงผลการคำนวณ

สามารถแบ่งการทำงานเป็นงานย่อยได้ดังนี้

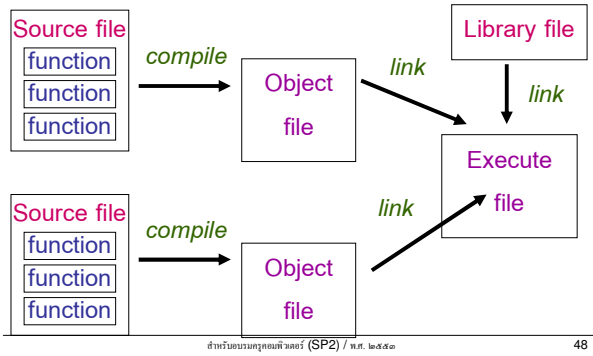
- รับข้อมูล 2 จำนวนจากผู้ใช้
- บวกเลข 2 จำนวนแล้วเก็บผลลัพธ์
- แสดงผลลัพธ์ของการทำงาน

ตัวอย่าง (ต่อ)

จะได้ว่าโปรแกรมประกอบด้วยฟังก์ชัน 4 ฟังก์ชัน คือ

- ฟังก์ชันหลัก
  - ฟังก์ชันการรับข้อมูล
  - ฟังก์ชันในการบวกเลข
  - ฟังก์ชันแสดงผลลัพธ์

ขั้นตอนการสร้างโปรแกรมด้วยภาษา C



4.1 รูปแบบของฟังก์ชัน

แบบที่ 1

int , char , float , double ฯลฯ

```
ชนิดข้อมูลที่คืนค่า ชื่อฟังก์ชัน ( การประกาศตัวแปร )
    {
        การประกาศตัวแปรภายในฟังก์ชัน;
        คำสั่ง;
        return  (ค่าข้อมูลที่ต้องการส่งค่ากลับ);
    }
```


รูปแบบของฟังก์ชัน (ต่อ)

แบบที่ 2

```
void   ชื่อฟังก์ชัน ( การประกาศตัวแปร )
{
       การประกาศตัวแปรภายในฟังก์ชัน;
       คำสั่ง;
}
```


ตัวอย่าง 4.1 แสดงการทำงานของโปรแกรมการบวกเลขจำนวนจริง 2 จำนวนที่รับจากผู้ใช้

```
#include <stdio.h>
double  InputDouble ( )
{
        double  x;
        printf ( "\nInput real value : " );
        scanf ( "%.2f  ", &x );
        return ( x );
}
```


ตัวอย่าง 4.1 (ต่อ)

```
double  SumDouble ( double  x,  double  y )
{
        return ( x + y );
}
void  PrintOut ( double  x )
{
        printf ( "\n Result of sum is : %.2f",  x );
}
```


ตัวอย่าง 4.1 (ต่อ)

```
void  main ( )
{
        double   a1, a2, sumVal;
        a1  =  InputDouble( );
        a2  =  InputDouble( );
        sumVal  =  SumDouble ( a1, a2 );
        PrintOut ( sumVal );
}
```


4.2 การประกาศโพรโทไทป์ของฟังก์ชัน

การประกาศโปรโตไทป์เป็นสิ่งจำเป็นในภาษาซีเนื่องจากภาษาซีเป็นภาษาในลักษณะที่ต้องมีการประกาศฟังก์ชันก่อนจะเรียกใช้ฟังก์ชันนั้น (Predefined Function)

จากตัวอย่างที่ 4.1 จะเห็นว่าฟังก์ชัน main ( ) จะอยู่ใต้ฟังก์ชันอื่น ๆ ที่มีการเรียกใช้ เป็นลักษณะที่ต้องประกาศฟังก์ชันที่ต้องการเรียกใช้ก่อนจากเรียกใช้ฟังก์ชันนั้น แต่หากต้องการย้ายฟังก์ชัน main ( ) ขึ้นไปไว้ด้านบน จะต้องมีการประกาศโปรโตไทป์ของฟังก์ชันที่ต้องการเรียกใช้ก่อนเสมอ

ตัวอย่าง 4.2 แสดงการทำงานของโปรแกรมการบวกเลขจำนวนจริง 2 จำนวนที่รับจากผู้ใช้ในลักษณะที่มีการประกาศโปรโตไทป์

```
#include <stdio.h>
double  InputDouble ( );
double  SumDouble ( double ,  double  );
void     PrintOut ( double );
```


ตัวอย่าง 4.2 (ต่อ)

```
void  main (  )
{
        double   a1, a2, sumVal;
        a1  =  InputDouble( );
        a2  =  InputDouble( );
        sumVal  =  SumDouble ( a1, a2 );
        PrintOut ( sumVal );
}
```


จะเห็นว่าในโปรโตไทป์ไม่มีการประกาศชื่อตัวแปร มีแต่การเขียนประเภทของตัวแปรไว้ภายในเป็นการช่วยให้คอมไพเลอร์สามารถตรวจสอบจำนวนของตัวแปร ประเภทของตัวแปร ประเภทของการคืนค่า ภายในโปรแกรมว่ามีการเรียกใช้งานสิ่งต่าง ๆ เกี่ยวกับฟังก์ชันนั้นถูกต้องหรือไม่ นอกจากนี้เราอาจจะแยกส่วนโปรโตไทป์ไปเขียนไว้ในอินคลูซไฟล์ก็ได้เช่นเดียวกัน

4.3 การเรียกใช้ฟังก์ชัน

การเรียกใช้ฟังก์ชันที่มีการคืนค่า จะใช้รูปแบบดังต่อไปนี้

ค่าที่รับ = ฟังก์ชัน (อาร์กิวเมนต์)

ตัวอย่าง

a1 ต้องมีชนิดเป็น double เนื่องจากค่าที่จะส่งคืนกลับมาจากฟังก์ชันมีชนิดเป็น double

```
a1  =  InputDouble ( );
```

ใช้คู่กับโปรโตไทป์

```
double  InputDouble ( );
```

ตัวอย่าง

a1 และ a2 ต้องมีชนิดเป็น double
เพื่อให้ตรงกับชนิดตัวแปรของอาร์กิวเมนท์
ที่ประกาศในโปรโตไทป์

```
sumVal =  SumDouble (a1,a2 );
```
ใช้คู่กับโปรโตไทป์
```
double  InputDouble ( );
```


ตัวอย่าง

```
PrintOut( sumVal );
```
ใช้คู่กับโปรโตไทป์
```
void PrintOut ( double );
```

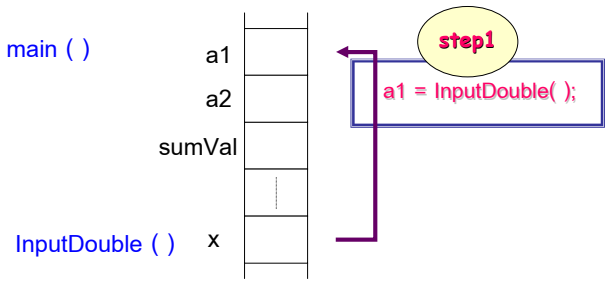
ประกาศให้รู้ว่าฟังก์ชันนี้ไม่มีการคืนค่า

4.4 ขอบเขต (Scope)

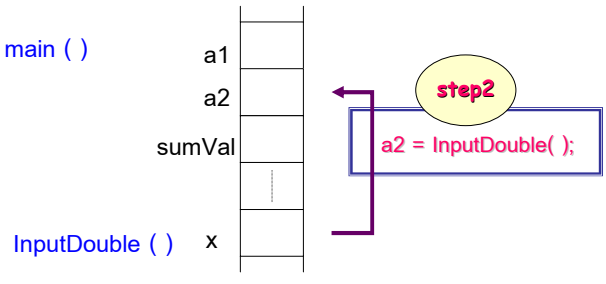
การทำงานของโปรแกรมภาษาซีจะทำงานที่ฟังก์ชัน main ( ) ก่อนเสมอ เมื่อฟังก์ชัน main ( ) เรียกใช้งานฟังก์ชันอื่น ก็จะมีการส่งคอนโทรล (Control) ที่ควบคุมการทำงานไปยังฟังก์ชันนั้น ๆ จนกว่าจะจบฟังก์ชัน หรือพบคำสั่ง return

เมื่อมีการเรียกใช้งานฟังก์ชันจะมีการจองพื้นที่หน่วยความจำสำหรับตัวแปรที่ต้องใช้ภายในฟังก์ชันนั้น และเมื่อสิ้นสุดการทำงานของฟังก์ชันก็จะมีการคืนพื้นที่หน่วยความจำส่วนนั้นกลับสู่ระบบ การใช้งานตัวแปรแต่ละตัวจะมีขอบเขตของการใช้งานขึ้นอยู่กับตำแหน่งที่ประกาศตัวแปรนั้น

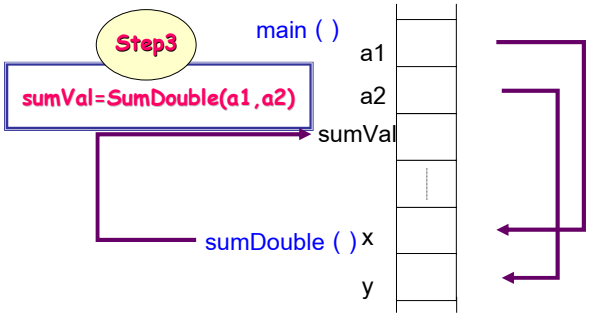
ตัวอย่าง จากตัวอย่าง 4.1 และ 4.2 สามารถแสดงขอบเขตการทำงานได้ดังนี้



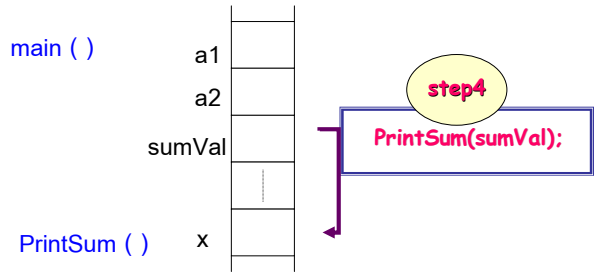
ตัวอย่าง (ต่อ)



ตัวอย่าง (ต่อ)



ตัวอย่าง (ต่อ)



จะเห็นว่าตัวแปร x ที่ประกาศในแต่ละขั้นตอนจะทำงานอยู่ภายในฟังก์ชันที่มีการประกาศค่าเท่านั้น และใช้พื้นที่ในการเก็บข้อมูลคนละส่วนกัน

ขอบเขตการทำงานของตัวแปรแต่ละตัวจะกำหนดอยู่ภายบล็อกของคำสั่งภายในเครื่องหมายปีกกา ({ }) หรือการประกาศในช่วงของการประกาศฟังก์ชัน เรียกตัวแปรเหล่านี้ว่า **ตัวแปรโลคอล** (Local Variable)

นอกจากนี้สามารถประกาศตัวแปรไว้ที่ภายนอกฟังก์ชันบริเวณส่วนเริ่มของโปรแกรมจะเรียกว่า **ตัวแปรโกลบอล** (Global Variable) ซึ่งเป็นตัวแปรที่สามารถเรียกใช้ที่ตำแหน่งใด ๆ ในโปรแกรมก็ได้ ยกเว้นในกรณีที่มีการประกาศตัวแปรที่มีชื่อเดียวกันตัวแปรโกลบอลภายในบล็อกหรือฟังก์ชัน

ตัวอย่าง 4.3

แสดงการทำงานของโปรแกรมในลักษณะที่มีตัวแปรโกลบอล แสดงขอบเขตการใช้งานของตัวแปรภายในโปรแกรม

```
#include  <stdio.h>
int   x;
void  func1 ( )
{
        x = x + 10;
        printf ( "func1 -> x : %d\n", x );
}
```


ตัวอย่าง 4.3 (ต่อ)

```
void  func2 ( int  x )
{
        x = x + 10;
        printf ( "func2 -> x : %d\n", x );
}
void  func3 ( ) {
        int  x=0;
        x  = x + 10;
        printf ( "func3 -> x : %d\n", x );
}
```

ตัวอย่าง 4.3 (ต่อ)

```
void main ( )
{
        x = 10;
        printf ( “main (start) -> x : %d\n”, x );
        func1 ( );
        printf ( “main (after func1) -> x : %d\n”, x );
        func2 ( x );
        printf ( “main (after func2) -> x : %d\n”, x);
        func3 ( );
        printf ( “main (after func3) -> x : %d\n”, x);
}
```


ตัวอย่าง 4.3 (ต่อ)

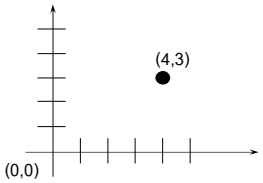
ผลการทำงาน

```
main (start) -> x : 10
func1 -> x : 20
main (after func1) -> x : 20
func2 -> x : 30
main (after func2) -> x : 20
func3 -> x : 10
main (after func3) -> x : 20
```


ข้อมูลแบบโครงสร้างและยูเนียน

(Structures and Unions)

5.1 ความรู้ทั่วไปเกี่ยวกับโครงสร้าง



หากต้องการเก็บข้อมูลจุดบนแกนโคออดิเนท จะประกอบไปข้อมูลแกน x และ y เป็นข้อมูลจำนวนเต็มประเภท int ประเภทข้อมูลที่ใช้ได้แก่ประเภทข้อมูลแบบโครงสร้าง สามารถประกาศประเภทข้อมูลที่ใช้ดังนี้

การประกาศประเภทข้อมูลแบบโครงสร้าง

```
struct   point {
                int  x;
                int  y;
            };
```

Member

หมายเหตุ การประกาศชื่อสมาชิกภายใน struct จะใช้ชื่อใดก็ได้ อาจจะซ้ำกับชื่อตัวแปรที่อยู่ภายนอก struct แต่ชื่อที่อยู่ภายใน struct เดียวกันห้ามประกาศชื่อซ้ำกัน

การประกาศตัวแปรข้อมูลแบบโครงสร้าง

แบบที่ 1

```
struct   point {
                int  x;
                int  y;
            } x, y , z;
```

หมายเหตุ จะเห็นว่าชื่อของ struct จะประกาศหรือไม่ก็ได้ หากไม่มีการประกาศจะไม่สามารถนำ struct นั้นกลับมาใช้ได้อีก

แบบที่ 2

```
struct    point {
                int  x;
                int  y;
          };

struct  point x,y,z
```

การประกาศแบบข้อมูลโครงสร้าง

การประกาศตัวแปรข้อมูลแบบโครงสร้าง

การกำหนดค่าเริ่มต้นให้กับตัวแปรข้อมูลแบบโครงสร้าง

```
struct  point pt = {320,200};
```

การอ้างถึงสมาชิกภายในตัวแปรข้อมูลแบบโครงสร้าง

```
struct_name.member
```


ตัวอย่าง

เมื่อต้องการอ้างถึงสมาชิกภายใน struct ว่าอยู่ตรงกับจุดใดบนแกนโคออดิเนทจะใช้

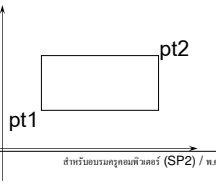
```
printf ( "%d, %d", pt.x, pt.y );
```

หรือหากต้องการคำนวณระยะทางจะว่าห่างจากจุดเริ่มต้น (0, 0) เท่าใดสามารถใช้

```
double  dist, sqrt (double);
dist =sqrt ((double)pt.x * pt.x +(double)pt.y * pt.y );
```


หมายเหตุ สมาชิกของข้อมูลประเภท struct อาจจะเป็นตัวแปรประเภทใดก็ได้ ทั้งข้อมูลพื้นฐาน และประเภทข้อมูลอื่น ๆ เช่น อาเรย์ และยังประกาศ ตัวแปรของข้อมูลประเภท struct ได้อีกด้วย

ตัวอย่าง



หากต้องการเก็บข้อมูลของสี่เหลี่ยมดังรูป สามารถทำการประกาศตัวแปรได้ดังนี้


```
struct    rect {
          struct point  pt1;
          struct point  pt2;
};

struct  rect  screen;
int co_x;
co_x = screen.pt1.x
```

การประกาศแบบข้อมูลโครงสร้าง

การประกาศตัวแปรข้อมูลแบบโครงสร้าง

การอ้างถึงสมาชิก

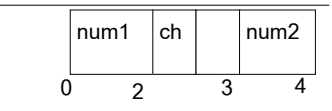
5.2 การเก็บข้อมูลแบบโครงสร้าง

การเก็บข้อมูลแบบโครงสร้างภายในหน่วยความจำจะเก็บตามลำดับที่มีการประกาศสมาชิกของข้อมูลนั้น โดยทั่วไปข้อมูลแบบโครงสร้างจะประกอบขึ้นจากข้อมูลหลาย ๆ ชนิด และข้อมูลแต่ละชนิดมักจะมีการจองพื้นที่ใช้งานแต่ต่างกัน เนื่องจากการจองพื้นที่หน่วยความจำในระบบส่วนใหญ่จะจองที่แอดเดรสที่หารด้วย 2 หรือ 4 ลงตัว

ตัวอย่าง

```
struct alignment {
    int     num1;
    char    ch;
    int     num2;
} example;
```

member



จะเห็นว่า num2 จะไม่สามารถใช้พื้นที่ที่ติดกับ ch ได้ เนื่องจาก num2 เป็นข้อมูลประเภทเลขจำนวนต้องใช้พื้นที่ที่มีแอดเดรสหารด้วย 2 หรือ 4 ลงตัว ทำให้เกิดที่ว่างที่ไม่สามารถนำมาใช้ประโยชน์ได้ เพราะฉะนั้นการประกาศสมาชิกของโครงสร้างจะมีผลต่อการใช้พื้นที่ในหน่วยความจำด้วย

5.3 การใช้ข้อมูลแบบโครงสร้างกับฟังก์ชัน

การทำงานของตัวแปรที่เป็นประเภทโครงสร้างสามารถทำงานต่าง ๆ ได้เช่นเดียวกับตัวแปรอื่น ๆ ยกเว้นการเปรียบเทียบตัวแปร struct กับตัวแปร struct เนื่องจากข้อมูลของตัวแปร struct จะเก็บอยู่ในตัวแปรที่เป็นสมาชิกของ struct การเปรียบเทียบจึงต้องทำผ่านตัวแปรที่เป็นสมาชิกของ struct เท่านั้น การใช้งานตัวแปร struct กับฟังก์ชันสามารถทำได้หลายลักษณะ ทั้งการให้ฟังก์ชันคืนค่าเป็น struct การส่งอากิวเมนท์ให้ฟังก์ชันเป็นตัวแปร struct

ตัวอย่าง 5.1 ฟังก์ชันใช้ในการกำหนดค่าให้กับตัวแปร struct

```
struct point makepoint ( int x, int y )
{
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

หมายเหตุ ตัวอย่างนี้แสดงฟังก์ชันที่ทำการส่งค่ากลับเป็นรูปแบบโครงสร้าง

การเรียกใช้งานฟังก์ชัน

```
struct rect screen;
struct point middle;
struct point makepoint ( int, int );
screen.pt1 = makepoint ( 0, 0 );
screen.pt2 = makepoint ( XMAX, YMAX );
middle = makepoint ((screen.pt1.x + screen.pt2.x) / 2,
                    (screen.pt1.y + screen.pt2.y) / 2 );
```


ตัวอย่าง 5.2 ฟังก์ชันการบวก x และ y ของจุด 2 จุด และคืนค่าผลของการบวกเป็น struct

```
struct point addpoint(struct point p1, struct point p2)
{
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}
```

หมายเหตุ ตัวอย่างนี้แสดงการส่งอาร์กิวเมนท์แบบ struct ให้กับฟังก์ชัน

ตัวอย่าง 5.3 ฟังก์ชันการหาว่าจุดอยู่ในพื้นที่สี่เหลี่ยมหรือไม่

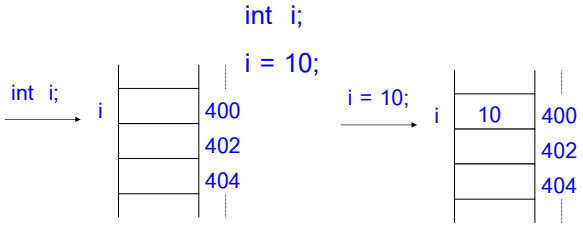
```
int pinrect ( struct point p, struct rect r )
{
    return p.x >= r.pt1.x && p.x < r.pt2.x &&
           p.y >= r.pt1.y && p.y < r.pt2.y;
}
```

หมายเหตุ ตัวอย่างนี้เป็นการหาว่าจุดที่ระบุอยู่ในพื้นที่สี่เหลี่ยมหรือไม่ โดยส่งค่าจุดและพื้นที่สี่เหลี่ยมเป็นอากิวเมนท์ให้กับฟังก์ชัน หากจุดอยู่ในพื้นที่สี่เหลี่ยมจะคืนค่า1 แต่หากจุดอยู่นอกพื้นที่สี่เหลี่ยมจะคืนค่าเป็น 0

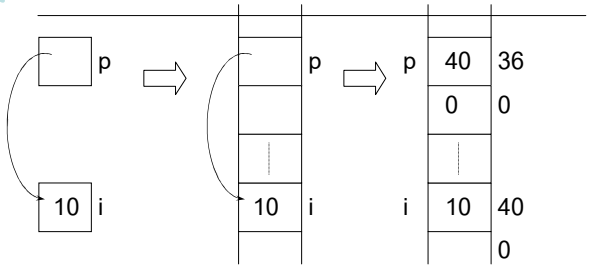
ตัวชี้และอาร์เรย์

(Pointer and Array)

6.1 ตัวชี้กับแอดเดรส (Pointers and Address)



รูปที่ 6.1 การแทนข้อมูลในหน่วยความจำของตัวแปรประเภทพื้นฐาน



รูปที่ 6.2 การแทนข้อมูลในหน่วยความจำของตัวแปรประเภทตัวชี้

6.2 การประกาศตัวแปรประเภทตัวชี้

การประกาศตัวแปรประเภทพอยน์เตอร์จะใช้ Unary Operator * ซึ่งมีชื่อเรียกว่า Indirection หรือ Dereferencing Operator โดยจะต้องประกาศประเภทของตัวแปรพอยน์เตอร์ให้สอดคล้องกับประเภทของตัวแปรที่เราต้องการ (ยกเว้นตัวแปรพอยน์เตอร์ประเภท void ที่สามารถชี้ไปยังตัวแปรประเภทใดก็ได้)

ตัวอย่าง

```
int  *ip;
```

เป็นการประกาศตัวแปร ip ให้เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรประเภท int

```
double  *dp,  atof(char  *);
```

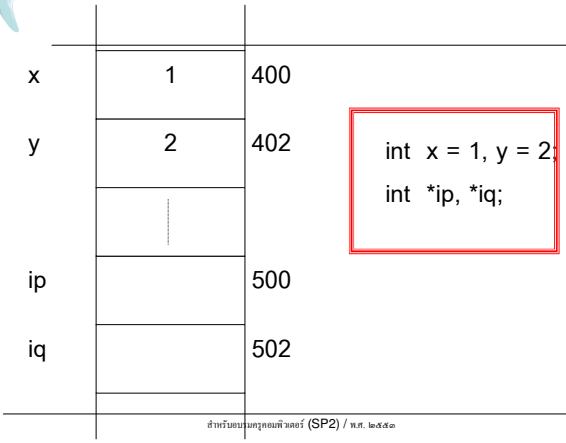
เป็นการประกาศตัวแปร dp เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรประเภท double และประกาศฟังก์ชัน atof มีพารามิเตอร์เป็นตัวแปรพอยน์เตอร์ประเภท char

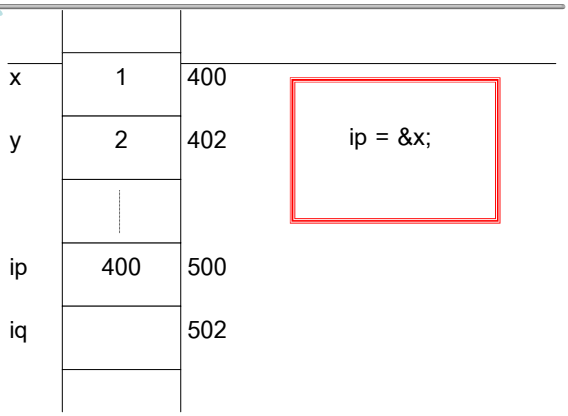
6.3 การกำหนดค่าและการอ่านค่าตัวแปรประเภทตัวชี้

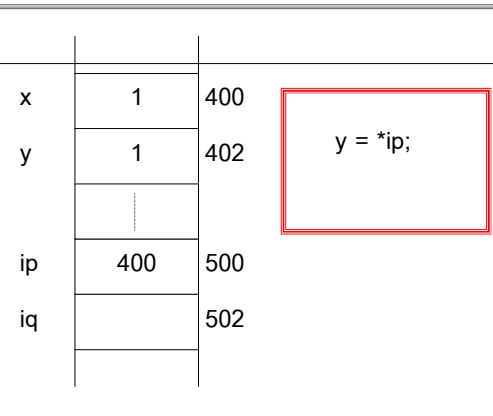
การกำหนดค่าให้กับตัวแปรพอยน์เตอร์จะเป็นการกำหนดแอดเดรสของตัวแปรที่มีประเภทสอดคล้องกับประเภทของตัวแปรพอยน์เตอร์เท่านั้น โดยการใช้ Unary Operator & เป็นโอเปอเรเตอร์ที่อ้างถึงแอดเดรสของออปเจ็ค (Object) ใด ๆ

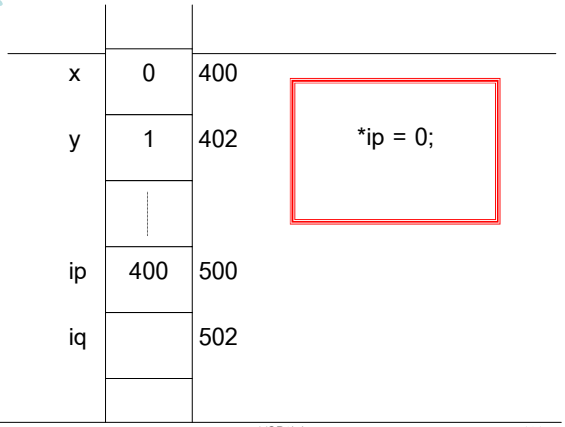
```
int  x = 1, y = 2;
int  *ip, *iq;
ip   =  &x;
y    =  *ip;
*ip  =  0;
y    = 5;
ip   =  &y;
*ip  =  3;
iq   = ip;
```

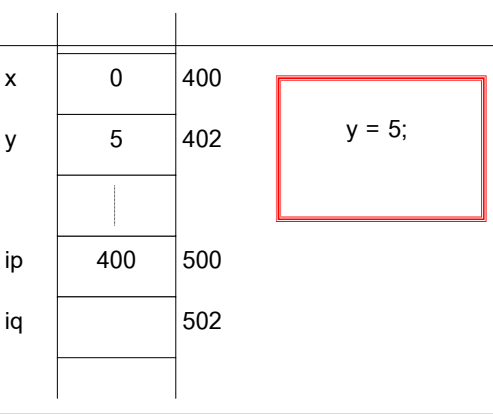
รูปที่ 6.3 การกำหนดค่าและการอ่านค่าตัวแปรตัวชี้

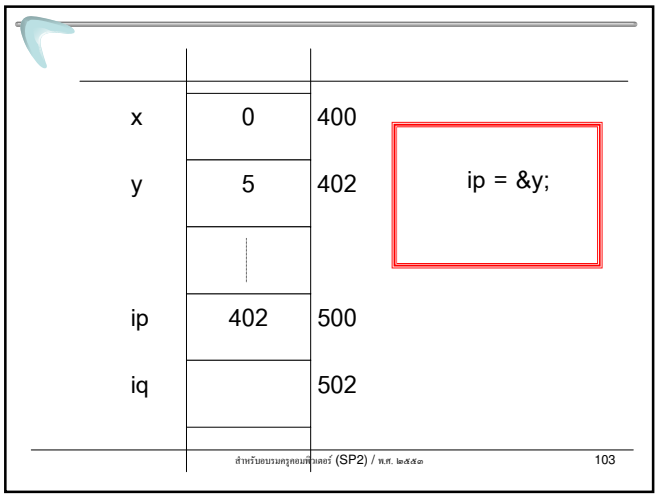


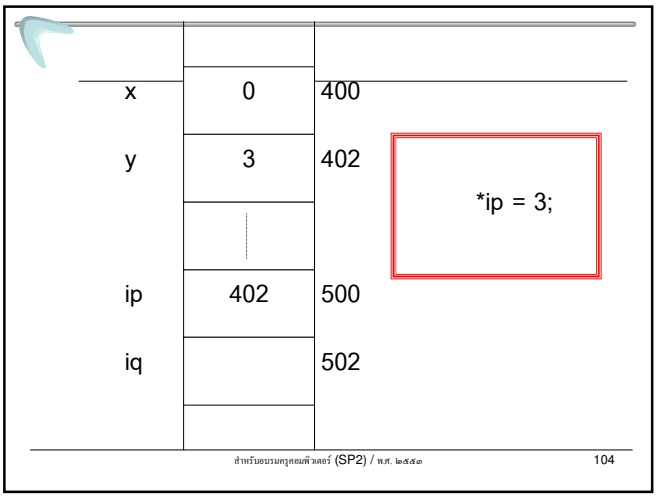


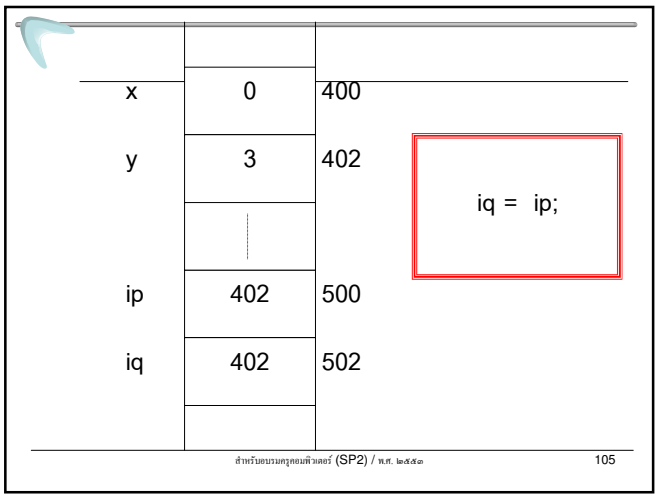












6.4 ตัวชี้และอาร์กิวเมนท์ของฟังก์ชัน (Pointer and Function Arguments)

เนื่องจากภาษาซีมีการส่งอากิวเมนต์ให้กับฟังก์ชันแบบ By Value และฟังก์ชันสามารถคืนค่า (return) ค่าได้เพียงหนึ่งค่า หากต้องการให้ฟังก์ชันมีการเปลี่ยนแปลงค่าและคืนค่ากลับมายังฟังก์ชันที่เรียกใช้มากกว่าหนึ่งค่าจะต้องนำพอยน์เตอร์เข้ามาช่วย

ตัวอย่างเช่น หากต้องการเขียนฟังก์ชันเพื่อสลับค่าของตัวแปร 2 ตัว ผลลัพธ์ที่ต้องการได้จากฟังก์ชันนี้จะมี 2 ค่าของตัวแปรที่ทำการสลับค่า หากอาร์กิวเมนต์เป็นตัวแปรธรรมดาจะไม่สามารถแก้ปัญหานี้ได้ จึงต้องใช้พอยน์เตอร์เข้ามาช่วย โดยการส่งค่าแอดเดรสของตัวแปรทั้ง 2 ให้กับฟังก์ชันที่จะสลับค่าของตัวแปรทั้ง 2 ผ่านทางตัวแปรพอยน์เตอร์ที่เป็นอาร์กิวเมนต์ของฟังก์ชัน

ตัวอย่าง 6.1

โปรแกรมตัวอย่างการสลับค่าตัวแปร 2 ตัว โดยผ่านฟังก์ชัน จะแสดงการส่งอาร์กิวเมนต์ในเป็นพอยน์เตอร์

```
#include <stdio.h>
void swap (int *, int *);
```

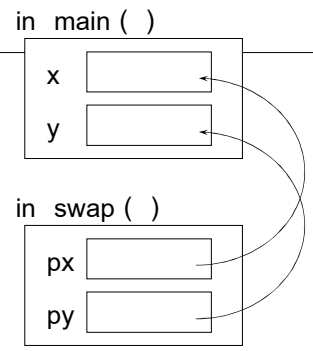
ตัวอย่าง 6.1 (ต่อ)

```
void  main ( )
{
    int  x = 5,  y = 10;
    printf("Before swap : x = %d", x, ", y = %d\n", y);
    swap ( &x,  &y);
    printf("After swap : x = %d", x, ", y = %d\n", y);
}
```


ตัวอย่าง 6.1 (ต่อ)

```
void  swap (int  *px,  int  *py)
{
    int  temp;
    temp  =  *px;
    *px    = *py;
    *py    =  temp;
}
```


อาร์กิวเมนท์ที่เป็นประเภทพอยน์เตอร์จะช่วยให้ฟังก์ชันสามารถเปลี่ยนค่าให้กับตัวแปรที่ส่งเข้ามาได้ เนื่องจากอาร์กิวเมนท์นั้นจะเก็บแอดเดรสของตัวแปรที่ส่งเข้ามา เมื่อมีการเปลี่ยนแปลงค่าของอาร์กิวเมนท์ผ่าน Dereferencing Operator (*) ค่าของตัวแปรที่ส่งเข้ามาจะถูกเปลี่ยนค่าพร้อมกันในทันที



รูปที่ 6.4 แสดงความสัมพันธ์ของการส่งอาร์กิวเมนท์แบบพอยน์เตอร์กับฟังก์ชัน

6.5 ตัวชี้กับอาร์เรย์ (Pointer and Arrays)

อาร์เรย์เป็นประเภทข้อมูลที่เก็บชุดของข้อมูลประเภทเดียวกัน มักใช้กับการทำงานที่ต้องทำงานกับตัวแปรชนิดเดียวกันหลายตัวที่มีการทำงานเหมือนกัน เช่น คะแนนของนักศึกษาภายในห้อง 20 คน เป็นต้น อาร์เรย์ในภาษาซีจะนำหลักการของพอยน์เตอร์เข้ามาใช้ การทำงานใด ๆ ของอาร์เรย์สามารถใช้พอยน์เตอร์เข้ามาแทนที่

การประกาศอาร์เรย์

```
int  table[10];
```

เป็นการกำหนดอาร์เรย์ชื่อ table เป็นอาร์เรย์ประเภท int ที่มีสมาชิกทั้งหมด 10 ตัว ตั้งแต่ table[0], table[1], table[2], ... , table[9] สมาชิกภายในอาร์เรย์จะเริ่มที่ 0 เสมอ และสมาชิกตัวสุดท้ายจะอยู่ที่ตำแหน่งของขนาดที่ประกาศไว้ลบด้วย 1

table[0] table[1] table[2] table[9]

รูปที่ 6.5 แสดงภาพจำลองของอาร์เรย์ขนาดสมาชิก 10 ตัว

การอ้างถึงสมาชิกในอาร์เรย์

จะใช้ระบบดัชนีโดยผ่านเครื่องหมาย [ ] เช่น อ้างถึงสมาชิกตัวที่ 3 ของอาร์เรย์ด้วย table[2] เป็นต้น การใช้งานสมาชิกของอาร์เรย์สามารถใช้งานได้เหมือนตัวแปรพื้นฐานทั่วไป

ตัวอย่าง

```
sumThird  =  table[0] + table[1] + table[2];
table[0]    =  5;
if ( a[0] > a[9] )
        printf (“First is greater than last\n” );
```


เราสามารถอ้างถึงสมาชิกทุกตัวภายในอาร์เรย์อย่างอิสระ ภายในขอบเขตของขนาดที่ได้ประกาศอาร์เรย์ไว้ แต่การใช้อาร์เรย์มักจะเป็นการเข้าถึงสมาชิกในลักษณะทั่วไปโดยใช้ตัวแปรประเภท int มาช่วย

ตัวอย่าง

สมมติให้ i, j, k เป็นตัวแปรประเภท int

```
for (int k = 0; k < 9; k++)
        printf (“Value at %d  = %d\n”, k+1, table[k]);
table[i + j] = 0;
table[7 – table[j]] = j;
```


สิ่งที่ต้องระวัง

ในภาษาซีจะไม่มีการกำหนดให้ตรวจสอบขอบเขตของอาร์เรย์ โปรแกรมเมอร์จะต้องพยายามเขียนโปรแกรมที่เกี่ยวข้องกับสมาชิกของอาร์เรย์ภายในขอบเขตที่ประกาศอาร์เรย์ไว้ หากมีการอ้างอิงถึงสมาชิกอาร์เรย์นอกขอบเขตที่ได้ระบุไว้ เช่น table[12] สิ่งที่ได้คือการไปอ่านข้อมูลในพื้นที่ของหน่วยความจำที่อาจจะเก็บค่าของตัวแปรตัวอื่น หรือค่าอื่นใดที่ไม่อาจคาดเดาได้

ตัวอย่าง 6.2 ให้อ่านค่าของจำนวนเต็ม 5 จำนวนจากคีย์บอร์ด และแสดงผลในลำดับที่กลับกัน

```
# include  <stdio.h>
#define  SIZE  5
main ( ) {
        int  k;
        int  table[SIZE];
        for  (k = 0;  k < SIZE;  k++)
                scanf ("%d", &table[k]);
        for (k = SIZE-1; k >= 0;  k--)
                printf ("%d\n", table[k]);
}
```


สมาชิกของอาร์เรย์อาจเป็นประเภทข้อมูลพื้นฐานใด ๆ ก็ได้ หรืออาจเป็นข้อมูลประเภท Enumeration เช่น

```
#define  TSIZE          10
#define  NAMESIZE   20
#define  ADDRSIZE    30
enum  month { JAN, FEB, MAR, APR, MAY,
              JUN, JUL, AUG, SEP, OCT,
              NOV, DEC }
```



```
typedef  enum  month  Month;
int  age[TSIZE];
float  size[TSIZE+1];
Month  date[8];
char  name[NAMESIZE], address[ADDRSIZE];
```


6.6 การใช้ตัวชี้กับอาร์เรย์

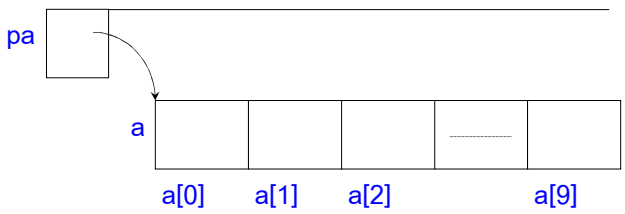
การทำงานใด ๆ ของอาร์เรย์สามารถใช้พอยน์เตอร์เข้ามาช่วย ซึ่งจะทำให้มีความเร็วในการทำงานสูงขึ้น สมมติว่ามีอาร์เรย์ a และพอยน์เตอร์ pa ดังนี้

```
int   a[10];
int   *pa;
```

กำหนดให้พอยน์เตอร์ pa ชี้ไปยังอาร์เรย์ a ด้วยคำสั่ง

```
pa = &a[0];  /*  หรือใช้คำสั่ง  pa = a;  */
```

pa จะเก็บค่าแอดเดรสเริ่มต้นของอาร์เรย์ a



รูปที่ 6.6 แสดงตัวชี้ชี้ไปยังแอดเดรสเริ่มต้นของอาร์เรย์

การนำไปใช้งานจะสามารถอ่านค่าอาร์เรย์ผ่านพอยน์เตอร์ได้ดังนี้

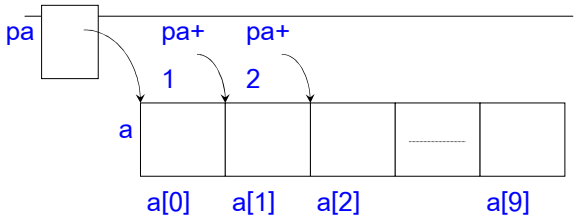
```
x  =  *pa;
```

จะเป็นการกำหนดค่าให้ x มีค่าเท่ากับ a[0] การเลื่อนไปอ่านค่าสมาชิกตำแหน่งต่าง ๆ ของอาร์เรย์ผ่านทางพอยน์เตอร์สามารถทำได้โดยการเพิ่มค่าพอยน์เตอร์ขึ้น 1 เพื่อเลื่อนไปยังตำแหน่งถัดไป หรือเพิ่มค่าขึ้น N เพื่อเลื่อนไป N ตำแหน่ง หรืออาจจะลดค่าเพื่อเลื่อนตำแหน่งลง

กรณีที่ pa ชี้อยู่ที่ a[0] คำสั่ง

pa+1;

จะเป็นการอ้างถึงแอดเดรสของ a[1] หากเป็น pa+i เป็นการอ้างถึงแอดเดรส a[i] หาก ต้องการอ้างถึงข้อมูลภายในของสมาชิกของอาร์เรย์ตำแหน่งที่ a[i] จะใช้ *(pa+i)



รูปที่ 6.7 แสดงการอ้างถึงตำแหน่งในอาร์เรย์ผ่านตัวชี้

การสั่งให้บวก 1 หรือบวก i หรือ ลบ i เป็นเหมือนการเลื่อนไปยังสมาชิกของอาร์เรย์ตำแหน่งที่ต้องการ เนื่องจากประเภทของข้อมูลแต่ละประเภทของอาร์เรย์ เช่น int, float, double และอื่น ๆ มีขนาดของข้อมูลที่ต่างกัน ทำให้ขนาดของสมาชิกภายในอาร์เรย์แต่ละประเภทมีขนาดแตกต่างกันด้วย การสั่งให้บวกหรือลบด้วยจำนวนที่ต้องการนั้นจะมีกลไกที่ทำหน้าที่คำนวณตำแหน่งที่ต้องการให้สอดคล้อง กับข้อมูลแต่ละประเภทโดยอัตโนมัติ

นอกจากนี้ยังสามารถใช้พอยน์เตอร์แทนอาร์เรย์ การอ้างโดยใช้ a[i] สามารถใช้ *(a+i) เนื่องจากทุกครั้งที่อ้างถึง a[i] ภาษาซีจะทำหน้าที่แปลงเป็น *(a+i) เพราะฉะนั้นการเขียนในรูปแบบใดก็ให้ผลลัพธ์ในการทำงานเช่นเดียวกัน และการอ้างถึงแอดเดรส เช่น &a[i] จะมีผลเท่ากับการใช้ a+i

ในลักษณะเดียวกันการใช้งานพอยน์เตอร์ก็สามารถใช้คำสั่งในลักษณะอาร์เรย์ก็ได้ เช่น การอ้างถึง *(pa+i) สามารถเขียนด้วย pa[i] ก็ได้ผลเช่นเดียวกัน

สิ่งที่แตกต่างกันของอาร์เรย์และพอยน์เตอร์ คือ พอยน์เตอร์เป็นตัวแปร แต่อาร์เรย์ไม่ใช่ตัวแปร สมมติให้ a เป็นอาร์เรย์และ pa เป็นพอยน์เตอร์ การอ้างถึง pa = a หรือ pa++ จะสามารถคอมไพล์ได้ แต่จะไม่สามารถใช้คำสั่ง a = pa หรือ a++ ได้

เมื่อมีการส่งชื่อของอาร์เรย์ให้แก่ฟังก์ชัน จะเป็นการส่งตำแหน่งแอดเดรสของสมาชิกตัวแรกของอาร์เรย์ให้แก่ฟังก์ชัน ดังนั้นพารามิเตอร์ในฟังก์ชันนั้นจะเป็นตัวแปรประเภทพอยน์เตอร์

ตัวอย่าง 6.3 ฟังก์ชันที่รับพารามิเตอร์เป็นพอยน์เตอร์โดยอาร์กิวเมนท์ที่ส่งมาเป็นอาร์เรย์

```
int  strlen (char *s)
{
    int  n;
    for ( n = 0; *s != '\0'; s++ )
        n++;
    return  n;
}
```


จะเห็นว่า s เป็นพอยน์เตอร์ ในฟังก์ชันจะมีการตรวจสอบข้อมูลว่ามีค่าเท่ากับ '\0' หรือไม่ และมีการเลื่อนตำแหน่งทีละ 1 ค่า (นับว่าข้อมูลมีความยาวเพิ่มขึ้นทีละ1) โดยใช้ s++

การเรียกใช้ฟังก์ชัน strlen สามารถทำได้หลายลักษณะ

```
strlen ("hello world");      /* string constant */
strlen (array);              /* char  array[10]  */
strlen (ptr);                /*  char  *ptr;      */
```


นอกจากนี้ยังอาจจะประกาศพารามิเตอร์ภายในฟังก์ชัน strlen ได้ใน 2 ลักษณะ คือ char *s แบบในตัวอย่าง หรืออาจจะใช้ char s[ ] ก็ได้ โดยทั่วไปจะใช้ในลักษณะแรก เพราะช่วยให้รู้ได้ทันทีว่า s เป็นตัวแปรพอยน์เตอร์ และยังสามารถส่งส่วนใดส่วนของอาร์เรย์ให้แก่ฟังก์ชันก็ได้ โดยไม่จำเป็นต้องส่งสมาชิกตัวแรกก็ได้เช่นกัน

ตัวอย่าง

```
f (&a[2])
```

หรือ

```
f (a+2)
```

เป็นการส่งแอดเดรสของสมาชิก a[2] ให้กับฟังก์ชัน f การประกาศฟังก์ชัน f สามารถทำได้โดยการประกาศ

```
f (int  arr[ ])  { ..... }
```

หรือ

```
f (int  *arr) { ..... }
```


6.7 การคำนวณกับแอดเดรส

ให้ p เป็นพอยน์เตอร์ชี้ไปยังอาร์เรย์ใด ๆ คำสั่ง p++ เป็นการเลื่อน p ไปยังสมาชิกถัดไป และคำสั่ง p += i เป็นการเลื่อนพอยน์เตอร์ไป i ตำแหน่งจากตำแหน่งปัจจุบัน นอกจากนี้ยังสามารถใช้เครื่องหมายความสัมพันธ์ (Relational Operator) เช่น ==, !=, <, >= และอื่น ๆ ทำงานร่วมกับพอยน์เตอร์ได้ สมมติให้ p และ q ชี้ไปยังสมาชิกของอาร์เรย์เดียวกัน


```
p  <  q
```

จะเป็นจริงเมื่อ p ชี้ไปที่สมาชิกที่อยู่ก่อนหน้าสมาชิกที่ q ชี้อยู่ การเปรียบเทียบในลักษณะจะใช้ได้ต่อเมื่อ p และ q ชี้ไปที่อาร์เรย์เดียวกันเท่านั้น

นอกจากนี้ยังสามารถใช้การลบหรือการบวกกับพอยน์เตอร์ได้เช่นเดียวกัน แต่สิ่งที่ควรระวังคือการทำเช่นนั้นจะต้องอยู่ในขอบเขตขนาดของอาร์เรย์เท่านั้น

ตัวอย่าง 6.3 ฟังก์ชัน strlen( ) ปรับปรุงให้กระชับขึ้น

```
int  strlen (char *s)
{
     char  *p = s;
     while  (*p != '\0')
          p++;
     return  p-s;
}
```


เนื่องจาก s ชี้อยู่ที่ตำแหน่งเริ่มต้น โดยมี p ชี้ไปที่ s เช่นเดียวกัน แต่จะมีการเลื่อน p ไปทีละหนึ่งตำแหน่ง จนกว่าค่าที่ตำแหน่งที่ p ชี้อยู่จะเท่ากับ '\0' เมื่อนำ p ค่าสุดท้ายมาลบกับ s ที่ตำแหน่งเริ่มต้นก็จะได้ความยาวของข้อมูลที่ส่งเข้ามา

6.8 ตัวชี้ตัวอักษรและฟังก์ชัน (Character Pointer and Function)

การทำงานกับข้อความหรือที่เรียกว่า สตริง (String) เป็นการใช้ข้อมูลตัวอักษรหลาย ๆ ตัว หรืออาร์เรย์ของข้อมูลประเภท char หรืออาจจะใช้พอยน์เตอร์ชี้ไปยังข้อมูลประเภท char การทำงานกับค่าคงที่สตริง (String Constant) สามารถเขียนภายในเครื่อง " "

ตัวอย่าง

"I am a string"

เมื่อมีการใช้ค่าคงที่สตริงจะมีการพื้นที่ในหน่วยความจำเท่ากับความยาวของค่าคงที่สตริงบวกด้วย 1 เนื่องจากลักษณะการเก็บข้อมูลประเภทข้อความในหน่วยความจำจะมีการปะตัวอักษร null หรือ '\0' ต่อท้ายเสมอเพื่อให้รู้ว่าเป็นจุดสิ้นสุดของข้อมูล การจองพื้นที่ดังกล่าวจะเหมือนการจองพื้นที่ของข้อมูลประเภทอาร์เรย์ เป็นอาร์เรย์ของ char

I		a	m		a		s	t	r	i	n	g	\0

รูปที่ 6.8 แสดงแบบจำลองการเก็บข้อมูลประเภทสตริงในหน่วยความจำ

ค่าคงที่สตริงที่พบเห็นได้เสมอได้แก่ข้อความที่ใช้ในฟังก์ชัน printf ( ) เช่น

printf ("Hello, world\n");

ฟังก์ชัน printf ( ) จะรับพารามิเตอร์เป็นพอยน์เตอร์ชี้ไปยังแอดเดรสของข้อมูลที่ตำแหน่งเริ่มต้นของอาร์เรย์ และนำข้อความนั้นแสดงออกทางอุปกรณ์แสดงข้อมูลมาตรฐาน

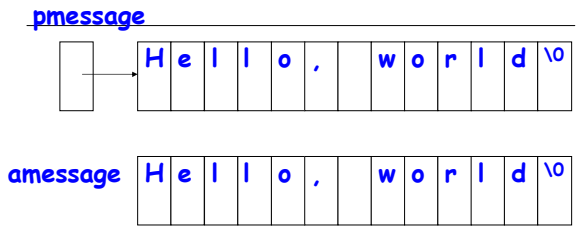
ในการเขียนโปรแกรมจะสามารถใช้พอยน์เตอร์ชี้ไปค่าคงที่สตริงใด ๆ ก็ได้ เช่น

```
char  *pmessage = “Hello, world”;
```

pmessage จะเป็นพอยน์เตอร์ประเภท char ชี้ไปที่อาร์เรย์ของตัวอักษร จะแตกต่างจากการใช้อาร์เรย์ทั่วไปเช่น

```
char  amessage[ ] = “Hello, world”;
```


ลักษณะของอาร์เรย์เช่น amessage จะมีการจองพื้นที่ใช้กับอาร์เรย์ขนาด 13 ตัวอักษร รวมทั้ง null ส่วนลักษณะของพอยน์เตอร์ที่ชี้ไปยังค่าคงที่สตริง จะมีการจองพื้นที่ให้กับค่าคงที่สตริงขนาด 13 ตัวอักษรเช่นเดียวกัน แต่จะมีการจองพื้นที่ให้กับพอยน์เตอร์และทำการชี้พอยน์เตอร์นั้นไปยังพื้นที่ของค่าคงที่สตริงที่จองเอาไว้



รูปที่ 6.9 การจองพื้นที่ให้กับอาร์เรย์และตัวชี้ชี้ไปยังค่าคงที่สตริง

ตัวอย่าง 6.5 ฟังก์ชัน strcpy ( ) ทำหน้าที่สำเนาข้อความจากตัวแปรหนึ่งไปยังอีกตัวแปรหนึ่งเขียนในลักษณะอาร์เรย์

```
void    strcpy ( char  *s,  char  *t )
{
      int  i=0;
      while ( ( s[i] = t[i] )  !=  ‘\0’ )
            i++;
}
```


ตัวอย่าง 6.6 ฟังก์ชัน strcpy ( ) เขียนในลักษณะพอยน์เตอร์

```
void strcpy ( char  *s,  char  *t )
{
    while ( ( *s = *t )  !=  ‘\0’ )  {
        s++;
        t++;
    }
}
```


ตัวอย่าง 6.7 ฟังก์ชัน strcpy ( ) เขียนในลักษณะพอยน์เตอร์แบบสั้น

```
void  strcpy ( char  *s,  char  *t )
{
    while ( ( *s++ = *t++ ) != ‘\0’ )  ;
}
```

การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง struct

กรณีการส่งอากิวเมนท์เป็นตัวแปร struct จะไม่เหมาะกับ struct ที่มีขนาดใหญ่ เนื่องจากทุกครั้งที่ส่งตัวแปร struct จะเป็นการสำเนาตัวแปรตัวใหม่ขึ้นมาในฟังก์ชัน ซึ่งจะทำให้ช้าและเปลืองพื้นที่หน่วยความจำ เราจะใช้พอยน์เตอร์เข้ามาช่วยแก้ปัญหานี้

โดยส่งแอดเดรสของตัวแปร struct มายังฟังก์ชันซึ่งรับอากิวเมนท์เป็นพอยน์เตอร์ อากิวเมนท์จะชี้ไปยังแอดเดรสเริ่มต้นของตัวแปร struct จะช่วยให้การทำงานเร็วขึ้นและเปลืองหน่วยความจำน้อยลง แต่สิ่งที่ต้องระวังคือหากมีการเปลี่ยนแปลงค่าที่อากิวเมนท์พอยน์เตอร์ชี้อยู่ ค่าในตัวแปร struct ที่ส่งมายังฟังก์ชันจะเปลี่ยนตามโดยอัตโนมัติ

ตัวอย่าง

```
struct point origin, *pp;
pp = &original;
printf ( “origin is (%d, %d)\n”, (*pp).x, (*pp).y );
```

จะได้ตัวแปร pp ชี้ไปยังข้อมูลแบบโครงสร้างชื่อ struct point การเขียน *pp จะเป็นการอ้างถึงโครงสร้าง

การอ้างถึงสมาชิกสามารถทำได้โดยอ้าง (*pp).x หรือ (*pp).y

หมายเหตุ สิ่งที่ต้องระวังคือ (*pp).x จะไม่เหมือนกับ *pp.x เนื่องจากเครื่องหมาย . จะมีลำดับความสำคัญสูงกว่า * ทำจะการแปลความหมาย *pp.x จะเหมือนกับการอ้าง *(pp.x) ซึ่งจะทำให้เกิดความผิดพลาดขึ้น

การอ้างถึงสมาชิกอาจเขียนอีกลักษณะหนึ่งโดยใช้เครื่องหมาย -> สมมติ p เป็นพอยน์เตอร์ รูปแบบการใช้เป็นดังนี้

```
p->member-of-structure
```

จะสามารถแปลงประโยคการใช้พอยน์เตอร์อ้างสมาชิกของ struct จากตัวอย่างข้างบนได้ว่า

```
printf ( “origin is (%d, %d)\n”, pp->x, pp->y);
```


ตัวอย่าง

หากมีพอยน์เตอร์ชี้ไปยัง struct rect ดังนี้

```
struct rect r, *rp = r;
```

การอ้างถึงสมาชิกต่อไปนี้จะมีผลเท่ากับการอ้างถึงสมาชิกตัวเดียวกัน

```
r.pt1.x
rp->pt1.x
(r.pt1).x
(rp->pt1).x
```


6.9 ตัวชี้ (pointer) ชี้ไปยังโครงสร้าง (pointer to structures)

พอยน์เตอร์เป็นตัวแปรที่เก็บแอดเดรสของตัวแปรอื่น สามารถใช้ชี้ไปยังข้อมูลประเภทใด ๆ การใช้พอยน์เตอร์ชี้ไปยังโครงสร้างสามารถทำได้ดังนี้

แบบที่ 1

```
typedef  struct {
            int  day;
            int  month;
            int  year;
         } Date;

Date  today;

Date  *ptrdate;
```

การประกาศแบบข้อมูลโครงสร้าง

การประกาศตัวแปรข้อมูลแบบโครงสร้าง

การประกาศตัวแปร pointer ชี้ไปยัง โครงสร้าง

แบบที่ 2

```
struct  date {
            int  day;
            int  month;
            int  year;
         } *ptrdate;
```


แบบที่ 3

```
typedef  struct {
            int  day;
            int  month;
            int  year;
         } Date;

typedef  Date  *PtrDate;

PtrDate  ptrdate;
```

การประกาศแบบข้อมูลโครงสร้าง

การประกาศประเภทตัวแปร pointer ชี้ไปยังโครงสร้าง

การประกาศตัวแปร pointer ชี้ไปยัง โครงสร้าง

การประกาศตัวแปร ptrdate ทั้ง 3 ลักษณะ จะสามารถใช้งานได้เหมือนกันทั้งหมด

หากต้องการให้ ptrdate ชี้ไปยังตัวแปรโครงสร้างสามารถทำได้ดังนี้

```
ptrdate  =  &today;
```


<u>การอ้างถึงสมาชิกของโครงสร้างผ่านตัวแปรพอยน์เตอร์</u>

```
ptrdate->day  = 7;
if  ( ptrdate->day  ==  31  &&  ptrdate->month  ==  12 )  .....
            scanf ( "%d", &ptrdate->year );
```

<u>การอ้างถึงสมาชิกโครงสร้างโดยใช้เครื่องหมาย ->

```
(*ptrdate).day  = 7;
if ( (*ptrdate).day  ==  31  &&  (*ptrdate).month  == 12 )  .....
            scanf ( "%d", &((*ptrdate).year) );
```


ตัวอย่าง 6.8 โปรแกรมตัวอย่างการใช้ตัวชี้ (pointer) ชี้ไปยังโครงสร้าง

```
#include  <stdio.h>
struct  date  {      /*date template */
     int   day;
     int   month;
     int   year;
};
typedef  struct  date   Date;
typedef  Date   *PtrDate;
```

ตัวอย่าง 6.8 (ต่อ)

```
main ( ) {
    Date      today;
    PtrDate   ptrdate;
    ptrdate = &today;
    ptrdate->day     = 27;
    ptrdate->month   = 9;
    ptrdate->year    = 1985;
    printf ( "Today\'s date is %2d/%2d/%4d\n",
       ptrdate->day, ptrdate->month, ptrdate->year );
}
```


นอกจากนี้ยังสามารถทำการกำหนดค่าเริ่มต้นให้กับตัวแปรแบบโครงสร้าง เช่น

```
Date  xmas = { 25, 12, 1986 };
```

และหากมีการกำหนดค่าเริ่มต้นให้กับสมาชิกของโครงสร้างไม่ครบทุกตัว หากตัวแปรนั้นเป็น external หรือ static ค่าของสมาชิกที่ขาดไปจะถูกกำหนดให้เป็น 0 แต่หากเป็นประเภท automatic จะไม่สามารถคาดได้ว่าค่าของสมาชิกที่ไปจะเป็นค่าใด

6.10 อาร์เรย์ของโครงสร้าง

การใช้งานโครงสร้างนอกจากใช้ในลักษณะของตัวแปรแล้วยังสมารถใช้งานในลักษณะของอาเรย์ได้อีกด้วย เช่น การเก็บข้อมูลประวัติของพนักงาน จะมีโครงสร้างที่ใช้เก็บข้อมูลของพนักงานแต่ละคน หากใช้ในลักษณะของตัวแปรปกติจะสามารถเก็บข้อมูลของพนักงานได้เพียง 1 คน ซึ่งพนักงานทั้งบริษัทอาจจะมีหลายสิบหรือหลายร้อยคน การเก็บข้อมูลในลักษณะนี้จะใช้อาเรย์เข้ามาช่วย เช่น

```
Person  staff[STAFFSIZE];
```


การอ้างโดยใช้คำสั่งต่าง ๆ

staff	อ้างถึงอาเรย์ของโครงสร้าง
staff[i]	อ้างถึงสมาชิกที่ i ในอาเรย์
staff[i].forename	อ้างถึงชื่อหน้าของสมาชิกที่ I ของอาเรย์
staff[i].surname[j]	อ้างถึงตัวอักษรตัวที่ j ในนามสกุลของสมาชิกที่ i ของอาเรย์

การเรียกใช้งานสมาชิกบางตัวในอาร์เรย์ของโครงสร้างผ่านฟังก์ชัน

การใช้ข้อมูลสมาชิกแต่ละตัวจะอ้างถึงโดยการอ้างผ่านระบบดัชนีเหมือนอาร์เรย์ทั่วไป เช่น ฟังก์ชันที่ใช้ในการพิมพ์ชื่อสมาชิกคนที่ระบุ จะเรียกใช้โดย

```
print_person ( staff[k] );
```

รูปแบบฟังก์ชันสามารถกำหนดด้วย

```
void  print_person ( Person  employee )
```


การเรียกใช้งานสมาชิกทุกตัวในอาร์เรย์ของโครงสร้างผ่านฟังก์ชัน

หากต้องการเรียกใช้งานฟังก์ชันที่ทำงานกับทั้งอาร์เรย์ เช่น การเรียกใช้งานฟังก์ชันที่ทำการเรียงลำดับอาร์เรย์ตามชื่อหน้า จะต้องส่งอาร์เรย์และขนาดของอาร์เรย์ไปยังฟังก์ชันนั้น เช่น

```
sort_forename ( staff, STAFFSIZE );
```

รูปแบบฟังก์ชันสามารถกำหนดด้วย

```
void  sort_forename ( Person  staff[ ], int  size )
```

การกำหนดค่าเริ่มต้นให้กับอาร์เรย์ของโครงสร้าง

การกำหนดค่าเริ่มต้นให้กับอาร์เรย์ของโครงสร้างสามารถทำได้โดย

```
Person  staff[ ] = {  { “Bloggs”, “Joe”,    MALE,     21 },
                      { “Smith”,  “John”,   MALE,     30 },
                      { “Black”,  “Mary”,   FEMALE,   25 }  };
```


6.11 อาร์เรย์แบบหลายมิติ (Multi-dimensional Arrays)

จากพื้นฐานที่ผ่านมาเรื่องอาร์เรย์จะเป็นลักษณะของอาร์เรย์มิติเดียว แต่อาร์เรย์อาจจะมีมากกว่า 1 มิติก็ได้ เช่น ข้อมูลคะแนนสอบของนักศึกษาแต่ละคนภายในชั้นซึ่งแบ่งเป็นคะแนนเก็บหลายส่วน จะพบว่าหากต้องการเก็บข้อมูลคะแนนสอบของนักศึกษาแต่ละคนสามารถใช้อาร์เรย์มิติเดียว ดังตัวอย่าง


```
#define  NUMBER_OF_PAPERS   5
int  student [ NUMBER_OF_PAPERS ];
/* int  student[5];   */
```

student[0]	student[1]	student[2]	student[3]	student[4]
5.6	8.5	12.6	24.1	16.0

แต่หากเพิ่มเติมว่าให้เก็บข้อมูลคะแนนสอบของ
นักศึกษาทุกคน จะต้องใช้อาร์เรย์หลายมิติเข้ามาเกี่ยวข้อง ตัวอย่างเช่นการเก็บข้อมูลคะแนนสอบของนักศึกษา 2 คนโดยมีคะแนนสอบของการสอบทั้งสิ้น 5 ครั้ง

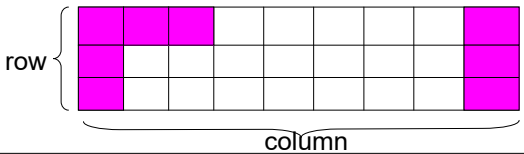
ครั้งที่	1	2	3	4	5
นาย ก	5.6	8.5	12.6	24.1	16.0
นาย ข	6.0	7.2	15.0	25.0	18.0

รูปที่ 6.10 แสดงตัวอย่างการเก็บข้อมูลคะแนนของ นักศึกษา

การอ้างอิงถึงข้อมูลในอาร์เรย์ 2 มิติ

เราจะมองอาร์เรย์ 2 มิติในลักษณะที่ประกอบด้วยแถว(row) และคอลัมน์(column) โดยข้อมูลที่อ้างอิงตัวแรกหมายถึง แถว และข้อมูลถัดมาคือ คอลัมน์

marks[2][8]



จากลักษณะความต้องการเก็บข้อมูลดังกล่าวจะต้องเตรียมอาร์เรย์เพื่อเก็บข้อมูลในลักษณะ 2 มิติ สามารถประกาศอาร์เรย์ดังนี้

```
#define  NUMBER_OF_PAPERS        5
#define  NUMBER_OF_STUDENTS      50
int
marks[NUMBER_OF_STUDENTS][NUMBER_OF_PAPERS];
/*  int marks[50][5];    */
```

โปรแกรมการรับค่าอาร์เรย์ 2 มิติ

```
#include<stdio.h>
main()
{   float  score[10][3];
    int i,j;
    printf("Please put score\n");
    for(i=0;i<10;i++)
      for(j=0;j<3;j++)
          scanf("%f",&score[i][j]);
}
```

score[0][0]	score[0][1]	score[0][2]
score[1][0]	score[1][1]	score[1][2]
score[9][0]	score[9][1]	score[9][2]

ลักษณะข้อมูล

score[0][0]	score[0][1]	score[0][2]
score[1][0]	score[1][1]	score[1][2]
score[2][0]	score[2][1]	score[2][2]
score[9][0]	score[9][1]	score[9][2]

หน่วยความจำ

200	202	204	206	208	210
[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]

210	212	214	216	218	220
[2][0]	[2][1]	[2][2]	[3][0]	[3][1]	[3][2]

220	222	224	226	228	230
[4][0]	[4][1]	[4][2]	[5][0]	[5][1]	[5][2]

Computer Algorithm : Analysis and Design

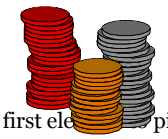
Krisana Chinnasarn, Ph.D.
Assistant Professor of Computer Science
Web: http://www.cs.buu.ac.th/~krisana
Stacks and Queues
Reading: *Chap.3 Weiss*

Stacks

- Stack: what is it?
- Applications
- Implementation(s)

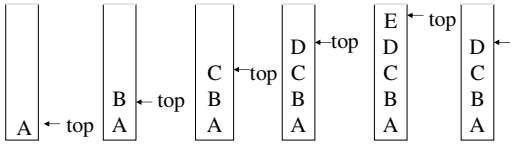
What is a stack?

- Stores a set of elements in a particular order
- Stack principle: LAST IN FIRST OUT
- = LIFO
- It means: the last element inserted is the first one to be removed
- Example



- Which is the first ele[illegible] pick up?

Last In First Out



Stack Applications

- Real life
  - Pile of books
  - Plate trays
- More applications related to computer science
  - Program execution stack (read more from your text)
  - Evaluating expressions

Array-based Stack Implementation

- Allocate an array of some size (pre-defined)
  - Maximum N elements in stack
- Bottom stack element stored at element 0
- last index in the array is the *top*
- Increment *top* when one element is pushed, decrement after pop

Stack Implementation: CreateS, isEmpty, isFull

```
Stack createS(max_stack_size) ::=
  #define MAX_STACK_SIZE 100 /* maximum stack size */
  typedef struct {
          int key;
          /* other fields */
          } element;
  element stack[MAX_STACK_SIZE];
  int top = -1;

  Boolean isEmpty(Stack) ::= top< 0;

  Boolean isFull(Stack) ::= top >= MAX_STACK_SIZE-1;
```


Push

```
void push(int *top, element item)
{
 /* add an item to the global stack */
    if (*top >= MAX_STACK_SIZE-1) {
         stack_full( );
         return;
    }
    stack[++*top] = item;
}
```


Pop

```
element pop(int *top)
{
 /* return the top element from the stack */
    if (*top == -1)
        return stack_empty( );  /* returns and error key */
    return stack[(*top)--];
 }
```


List-based Stack Implementation: Push

```
void push(pnode top, element item)
{
  /* add an element to the top of the stack */
  pnode temp =
                (pnode) malloc (sizeof (node));
  if (IS_FULL(temp)) {
    fprintf(stderr, " The memory is full\n");
    exit(1);
    }
    temp->item = item;
    temp->next= top;
    top= temp;
}
```


Pop

```
element pop(pnode top) {
/* delete an element from the stack */
   pnode temp = top;
   element item;
   if (IS_EMPTY(temp))  {
     fprintf(stderr,  "The stack is empty\n");
     exit(1);
   }
   item = temp->item;
   top = temp->next;
    free(temp);
    return item;
}
```

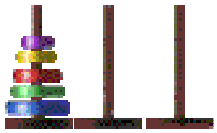

Algorithm Analysis

- push O(?)
- pop O(?)
- isEmpty O(?)
- isFull O(?)

- What if *top* is stored at the beginning of the array?

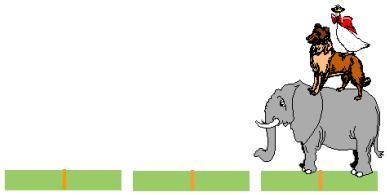
The Towers of Hanoi
A Stack-based Application

- GIVEN: three poles
- a set of discs on the first pole, discs of different sizes, the smallest discs at the top
- GOAL: move all the discs from the left pole to the right one.
- CONDITIONS: only one disc may be moved at a time.
- A disc can be placed either on an empty pole or on top of a larger disc.



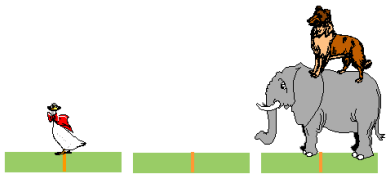
13

Towers of Hanoi



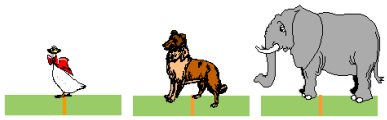
14

Towers of Hanoi



15

Towers of Hanoi

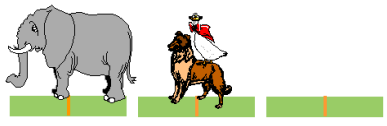


16

Towers of Hanoi

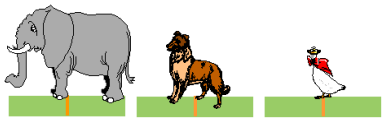
17

Towers of Hanoi



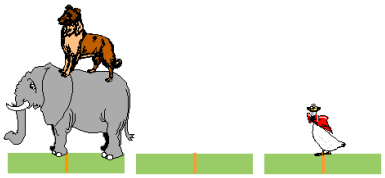
18

Towers of Hanoi



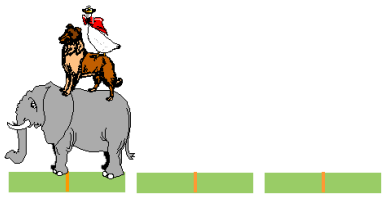
19

Towers of Hanoi



20

Towers of Hanoi



21

Towers of Hanoi – Recursive Solution

```
void hanoi (int discs,
            Stack fromPole,
            Stack toPole,
            Stack aux) {
Disc d;
if( discs >= 1) {
    hanoi(discs-1, fromPole, aux, toPole);
    d = fromPole.pop();
    toPole.push(d);
    hanoi(discs-1,aux, toPole, fromPole);
}
```

22

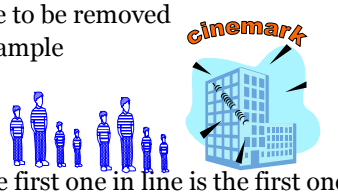
Is the End of the World Approaching?

- Problem complexity 2^n
- 64 gold discs
- Given 1 move a second

23

Queue

- Stores a set of elements in a particular order
- Stack principle: FIRST IN FIRST OUT
- = FIFO
- It means: the first element inserted is the first one to be removed
- Example



- The first one in line is the first one to be served

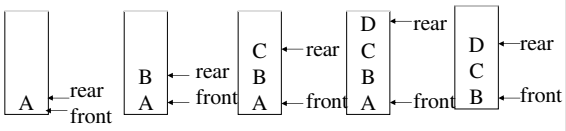
24

Queue Applications

- Real life examples
  - Waiting in line
  - Waiting on hold for tech support
- Applications related to Computer Science
  - Threads
  - Job scheduling (e.g. Round-Robin algorithm for CPU allocation)

25

First In First Out



26

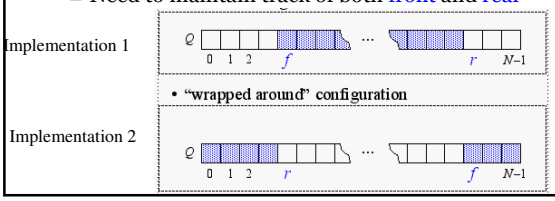
Applications: Job Scheduling

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					queue is empty
-1	0	J1				Job 1 is added
-1	1	J1	J2			Job 2 is added
-1	2	J1	J2	J3		Job 3 is added
0	2		J2	J3		Job 1 is deleted
1	2			J3		Job 2 is deleted

27

Array-based Queue Implementation

- As with the array-based stack implementation, the array is of fixed size
  - A queue of maximum N elements
- Slightly more complicated
  - Need to maintain track of both front and rear



28

Implementation 1: createQ, isEmptyQ, isFullQ

```
Queue createQ(max_queue_size) ::=
# define MAX_QUEUE_SIZE 100/* Maximum queue size */
typedef struct {
        int key;
        /* other fields */
        } element;
element queue[MAX_QUEUE_SIZE];
int rear = -1;
int front = -1;
Boolean isEmpty(queue) ::= front == rear
Boolean isFullQ(queue) ::= rear == MAX_QUEUE_SIZE-1
```

29

Implementation 1: enqueue

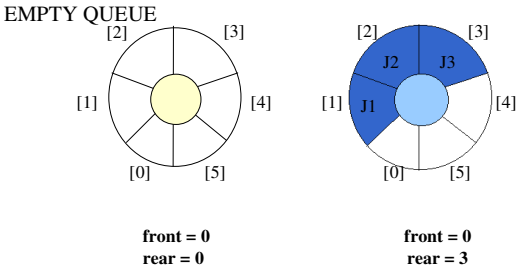
```
void enqueue(int *rear, element item)
{
/* add an item to the queue */
   if (*rear == MAX_QUEUE_SIZE_1) {
      queue_full( );
      return;
   }
   queue [++*rear] = item;
}
```

30

Implementation 1: dequeue

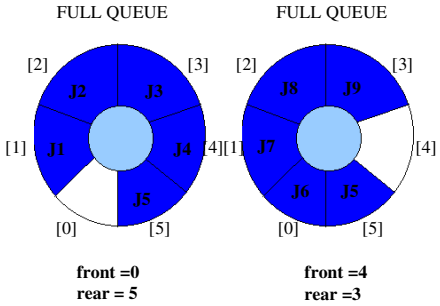
```
element dequeue(int *front, int rear)
{
/* remove element at the front of the queue */
   if ( *front == rear)
       return queue_empty( );     /* return an error key */
   return queue [++ *front];
}
```


Implementation 2: Wrapped Configuration



Can be seen as a circular queue

Leave one empty space when queue is full
Why?



How to test when queue is empty?
How to test when queue is full?

Enqueue in a Circular Queue

```
void enqueue(int front, int *rear, element item)
{
/* add an item to the queue */
    *rear = (*rear +1) % MAX_QUEUE_SIZE;
     if (front == *rear) /* reset rear and print error */
     return;
   }
     queue[*rear] = item;
}
```


Dequeue from Circular Queue

```
element dequeue(int* front, int rear)
{
  element item;
  /* remove front element from the queue and put it in item */
     if (*front == rear)
        return queue_empty( );
               /* queue_empty returns an error key */
     *front = (*front+1) % MAX_QUEUE_SIZE;
     return queue[*front];
}
```


List-based Queue Implementation: Enqueue

```
void enqueue(pnode &front, pnode rear, element item)
{  /* add an element to the rear of the queue */
   pnode temp =
                    (pnode) malloc(sizeof (queue));
   if (IS_FULL(temp)) {
     fprintf(stderr, " The memory is full\n");
     exit(1);
     }
     temp->item = item;
     temp->next= NULL;
     if (front)  { (rear) -> next= temp;}
     else front = temp;
     rear = temp;    }
```

Dequeue

```
element dequeue(pnode &front) {
/* delete an element from the queue */
   pnode temp = front;
   element item;
   if (IS_EMPTY(front))  {
      fprintf(stderr,  "The queue is empty\n");
      exit(1);
   }
   item = temp->item;
   front = temp->next;
    free(temp);
    return item;
}
```

37

Algorithm Analysis

- enqueue O(?)
- dequeue O(?)
- size O(?)
- isEmpty O(?)
- isFull O(?)

- What if I want the first element to be always at Q[o] ?

38

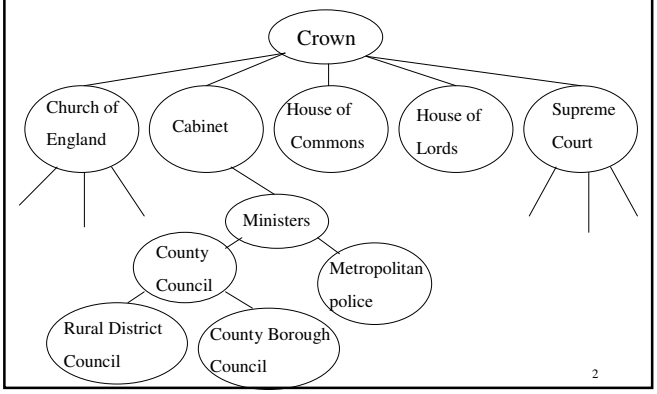
Computer Algorithm : Analysis and Design

Krisana Chinnasarn, Ph.D.
Assistant Professor of Computer Science
Web: http://www.cs.buu.ac.th/~krisana

Trees. Binary Trees.

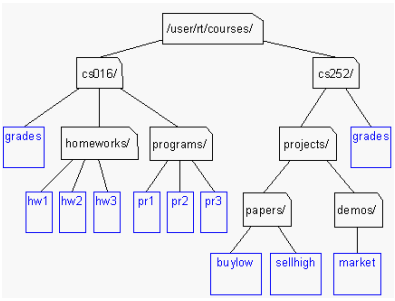
Reading: *Chap.4 (4.1-4.2) Weiss*

The British Constitution



More Trees Examples

- Unix / Windows file structure

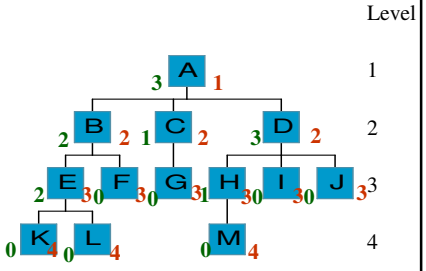


Definition of Tree

- A tree is a finite set of one or more nodes such that:
- There is a specially designated node called the root.
- The remaining nodes are partitioned into n>=0 disjoint sets T_1, ..., T_n, where each of these sets is a tree.
- We call T_1, ..., T_n the subtrees of the root.

Level and Depth

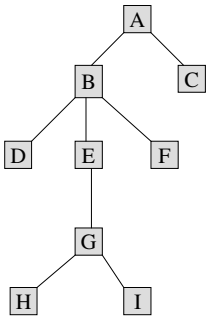
node (13)
degree of a node
leaf (terminal)
nonterminal
parent
children
sibling
degree of a tree (3)
ancestor
level of a node
height of a tree (4)



Terminology

- The degree of a node is the number of subtrees of the node
  - The degree of A is 3; the degree of C is 1.
- The node with degree 0 is a leaf or terminal node.
- A node that has subtrees is the *parent* of the roots of the subtrees.
- The roots of these subtrees are the *children* of the node.
- Children of the same parent are *siblings*.
- The ancestors of a node are all the nodes along the path from the root to the node.

Tree Properties

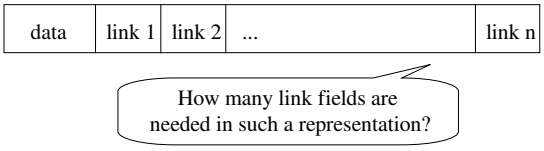


Property	Value
Number of nodes	
Height	
Root Node	
Leaves	
Interior nodes	
Number of levels	
Ancestors of H	
Descendants of B	
Siblings of E	
Right subtree	

7

Representation of Trees

- List Representation
  - (A (B (E (K, L), F), C (G), D (H (M), I, J)))
  - The root comes first, followed by a list of sub-trees

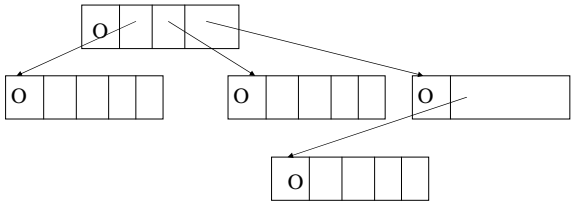


How many link fields are needed in such a representation?

8

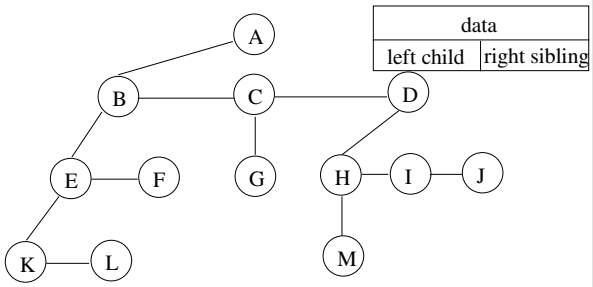
A Tree Node

- Every tree node:
  - object – useful information
  - children – pointers to its children nodes



9

Left Child - Right Sibling



10

Tree ADT

- Objects: any type of objects can be stored in a tree
- Methods:
- accessor methods
  - root() – return the root of the tree
  - parent(p) – return the parent of a node
  - children(p) – returns the children of a node
- query methods
  - size() – returns the number of nodes in the tree
  - isEmpty() - returns true if the tree is empty
  - elements() – returns all elements
  - isRoot(p), isInternal(p), isExternal(p)

11

Tree Implementation

```
typedef struct tnode {
    int key;
    struct tnode* lchild;
    struct tnode* sibling;
} *ptnode;
```

- Create a tree with three nodes (one root & two children)
- Insert a new node (in tree with root R, as a new child at level L)
- Delete a node (in tree with root R, the first child at level L)

12

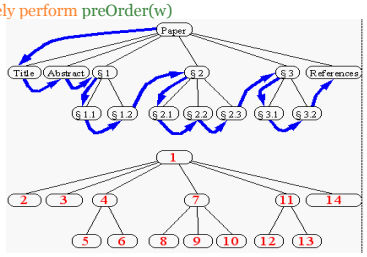
Tree Traversal

- Two main methods:
  - Preorder
  - Postorder
- Recursive definition

- PREorder:
  - visit the root
  - traverse in preorder the children (subtrees)
- POSTorder
  - traverse in postorder the children (subtrees)
  - visit the root

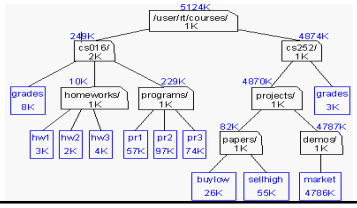
Preorder

- **preorder traversal**
  Algorithm preOrder(v)
    "visit" node v
    for each child w of v **do**
      recursively perform preOrder(w)



Postorder

- **postorder traversal**
  Algorithm postOrder(v)
    for each child w of v **do**
      recursively perform postOrder(w)
    "visit" node v
- du (**disk usage) command in Unix**



Preorder Implementation

```
public void preorder(ptnode t) {
    ptnode ptr;
    display(t->key);
    for(ptr = t->lchild; NULL != ptr; ptr = ptr->sibling) {
        preorder(ptr);
    }
}
```

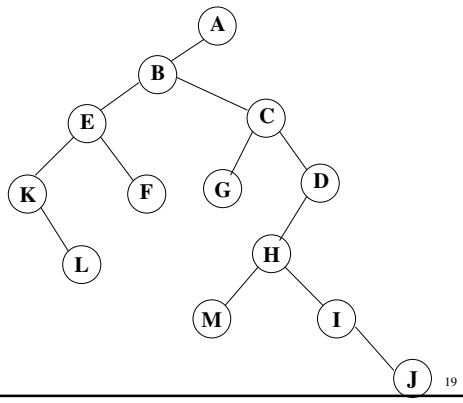

Postorder Implementation

```
public void postorder(ptnode t) {
    ptnode ptr;
    for(ptr = t->lchild; NULL != ptr; ptr = ptr->sibling) {
        postorder(ptr);
    }
    display(t->key);
}
```


Binary Trees

- A special class of trees: max degree for each node is 2
- Recursive definition: A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called *the left subtree* and *the right subtree*.
- Any tree can be transformed into binary tree.
  - by left child-right sibling representation

Example



19

ADT Binary Tree

objects: a finite set of nodes either empty or consisting of a root node, left *BinaryTree*, and right *BinaryTree*.

method:

for all *bt*, *bt1*, *bt2* ∈ *BinTree*, *item* ∈ *element*

Bintree create()::= creates an empty binary tree

Boolean isEmpty(*bt*)::= if (*bt*==empty binary tree) return *TRUE* else return *FALSE*

20

BinTree makeBT(*bt1*, *item*, *bt2*)::= return a binary tree whose left subtree is *bt1*, whose right subtree is *bt2*, and whose root node contains the data *item*

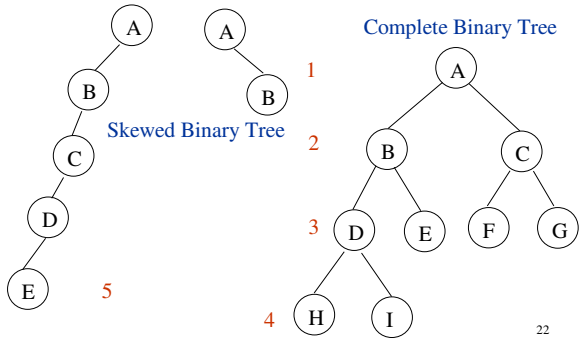
Bintree leftChild(*bt*)::= if (IsEmpty(*bt*)) return error
else return the left subtree of *bt*

element data(*bt*)::= if (IsEmpty(*bt*)) return error
else return the data in the root node of *bt*

Bintree rightChild(*bt*)::= if (IsEmpty(*bt*)) return error
else return the right subtree of *bt*

21

Samples of Trees



22

Maximum Number of Nodes in BT

- The maximum number of nodes on level i of a binary tree is 2^{i-1}, i>=1.
- The maximum nubmer of nodes in a binary tree of depth k is 2^k-1, k>=1.

Prove by induction.

$$\sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$

23

Relations between Number of Leaf Nodes and Nodes of Degree 2

For any nonempty binary tree, T, if n_0 is the number of leaf nodes and n_2 the number of nodes of degree 2, then $n_0=n_2+1$

proof:

Let *n* and *B* denote the total number of nodes & branches in *T*.

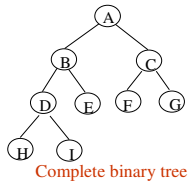
Let n_0, n_1, n_2 represent the nodes with no children, single child, and two children respectively.

$n= n_0+n_1+n_2$, $B+1=n$, $B=n_1+2n_2$ ==> $n_1+2n_2+1= n$, $n_1+2n_2+1= n_0+n_1+n_2$ ==> $n_0=n_2+1$

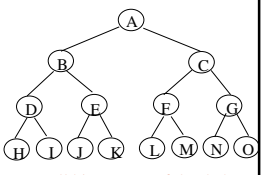
24

Full BT vs. Complete BT

- A full binary tree of depth k is a binary tree of depth k having 2^k-1 nodes, $k>=0$.
- A binary tree with n nodes and depth k is complete *iff* its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k.



Complete binary tree

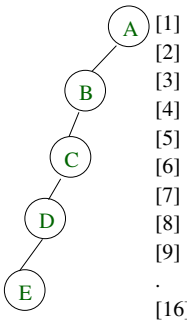


Full binary tree of depth 4

Binary Tree Representations

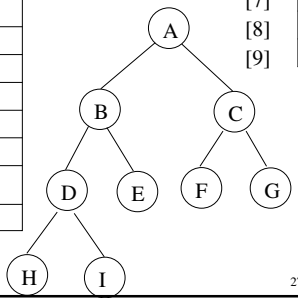
- If a complete binary tree with n nodes (depth = $\log n + 1$) is represented sequentially, then for any node with index i, $1<=i<=n$, we have:
  - *parent*(i) is at $i/2$ if i!=1. If i=1, i is at the root and has no parent.
  - *leftChild*(i) is at $2i$ if $2i<=n$. If $2i$>n, then i has no left child.
  - *rightChild*(i) is at $2i+1$ if $2i+1<=n$. If $2i+1$ >n, then i has no right child.

Sequential Representation



[1]	A
[2]	B
[3]	--
[4]	C
[5]	--
[6]	--
[7]	--
[8]	D
[9]	--
.	.
[16]	E

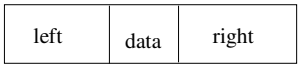
(1) waste space
(2) insertion/deletion problem

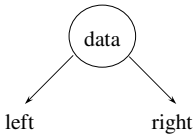


[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I

Linked Representation

```
typedef struct tnode *ptnode;
typedef struct tnode {
 int data;
 ptnode left, right;
};
```

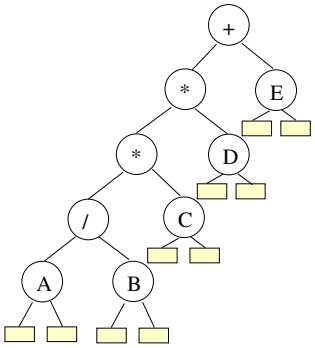




Binary Tree Traversals

- Let L, V, and R stand for moving left, visiting the node, and moving right.
- There are six possible combinations of traversal
  - lRr, lrR, Rlr, Rrl, rRl, rlR
- Adopt convention that we traverse left before right, only 3 traversals remain
  - lRr, lrR, Rlr
  - inorder, postorder, preorder

Arithmetic Expression Using BT



inorder traversal
A / B * C * D + E
infix expression
preorder traversal
+ * * / A B C D E
prefix expression
postorder traversal
A B / C * D * E +
postfix expression
level order traversal
+ * E * D / C A B

Inorder Traversal *(recursive version)*

```
void inorder(ptnode ptr)
/* inorder tree traversal */
{
    if (ptr) {
        inorder(ptr->left);
        printf("%d", ptr->data);
        indorder(ptr->right);
    }
}
```

A / B * C * D + E

Preorder Traversal *(recursive version)*

```
void preorder(ptnode ptr)
/* preorder tree traversal */
{
    if (ptr) {
        printf("%d", ptr->data);
        preorder(ptr->left);
        predorder(ptr->right);
    }
}
```

+ * * / A B C D E

Postorder Traversal *(recursive version)*

```
void postorder(ptnode ptr)
/* postorder tree traversal */
{
    if (ptr) {
        postorder(ptr->left);
        postdorder(ptr->right);
        printf("%d", ptr->data);
    }
}
```

A B / C * D * E +

Level Order Traversal

(using queue)

```
void levelOrder(ptnode ptr)
/* level order tree traversal */
{
  int front = rear = 0;
  ptnode queue[MAX_QUEUE_SIZE];
  if (!ptr) return; /* empty queue */
  enqueue(front, &rear, ptr);
  for (;;) {
    ptr = dequeue(&front, rear);
```

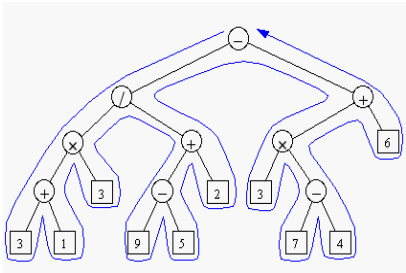


```
    if (ptr) {
      printf("%d", ptr->data);
      if (ptr->left)
        enqueue(front, &rear,
                ptr->left);
      if (ptr->right)
        enqueue(front, &rear,
                ptr->right);
    }
    else break;
  }
}
```

+ * E * D / C A B

Euler Tour Traversal

- generic traversal of a binary tree
- the preorder, inorder, and postorder traversals are special cases of the Euler tour traversal
- "walk around" the tree and visit each node three times:
  - on the left
  - from below
  - on the right



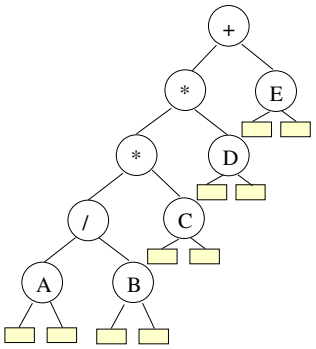
Euler Tour Traversal (cont'd)

```
eulerTour(node v) {
  perform action for visiting node on the left;
  if v is internal then
     eulerTour(v->left);
  perform action for visiting node from below;
  if v is internal then
     eulerTour(v->right);
  perform action for visiting node on the right;
}
```


Euler Tour Traversal (cont'd)

- preorder traversal = Euler Tour with a "visit" only on the left
- inorder = ?
- postorder = ?
- Other applications: compute number of descendants for each node v:
  - counter = 0
  - increment counter each time node is visited on the left
  - #descendants = counter when node is visited on the right – counter when node is visited on the left + 1
- Running time for Euler Tour?

Application: Evaluation of Expressions



inorder traversal
A / B * C * D + E
infix expression
preorder traversal
+ * * / A B C D E
prefix expression
postorder traversal
A B / C * D * E +
postfix expression
level order traversal
+ * E * D / C A B

Inorder Traversal *(recursive version)*

```
void inorder(ptnode ptr)
/* inorder tree traversal */
{
    if (ptr) {
        inorder(ptr->left);
        printf("%d", ptr->data);
        inorder(ptr->right);
    }
}
```

A / B * C * D + E

Preorder Traversal *(recursive version)*

```
void preorder(ptnode ptr)
/* preorder tree traversal */
{
    if (ptr) {
        printf("%d", ptr->data);
        preorder(ptr->left);
        preorder(ptr->right);
    }
}
```

+ * * / A B C D E

Postorder Traversal *(recursive version)*

```
void postorder(ptnode ptr)
/* postorder tree traversal */
{
    if (ptr) {
        postorder(ptr->left);
        postorder(ptr->right);
        printf("%d", ptr->data);
    }
}
```

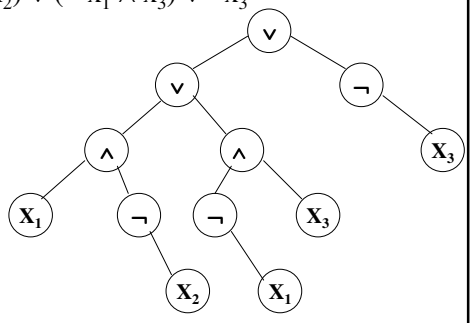
A B / C * D * E +

Application: Propositional Calculus Expression

- A variable is an expression.
- If x and y are expressions, then ¬x, x∧y, x∨y are expressions.
- Parentheses can be used to alter the normal order of evaluation (¬ > ∧ > ∨).
- Example: $x_1 \vee (x_2 \wedge \neg x_3)$

Propositional Calculus Expression

$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3) \vee \neg x_3$



postorder traversal (postfix evaluation)

Node Structure

left	*data*	*value*	*right*

```
typedef emun {not, and, or, true, false } logical;
typedef struct tnode *ptnode;
typedef struct node {
        logical     data;
        short int   value;
        ptnode right, left;
        } ;
```


Postorder Eval

```
void post_order_eval(ptnode node)
{
/* modified post order traversal to evaluate a propositional
calculus tree */
  if (node) {
    post_order_eval(node->left);
    post_order_eval(node->right);
    switch(node->data) {
      case not:  node->value =
            !node->right->value;
            break;
```


Postorder Eval (cont'd)

```
      case and:    node->value =
            node->right->value &&
            node->left->value;
            break;
       case or:      node->value =
            node->right->value | |
            node->left->value;
            break;
       case true:   node->value = TRUE;
            break;
       case false:  node->value = FALSE;
      }
   }
}
```

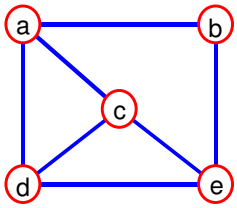
Computer Algorithm : Analysis and Design

Krisana Chinnasarn, Ph.D.
Assistant Professor of Computer Science
Web: www.cs.buu.ac.th/~krisana
Graphs (I)
Reading: Chap.9 : *Weiss,* Chap.4 :Baase

Contents

What is a Graph?

- A graph G = (V,E) is composed of:
  - V: set of vertices
  - E: set of edges connecting the vertices in V
- An edge e = (u,v) is a pair of vertices
- Example:

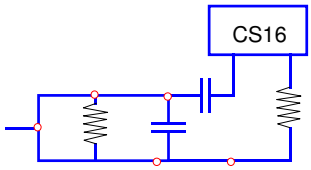


V= {a,b,c,d,e}

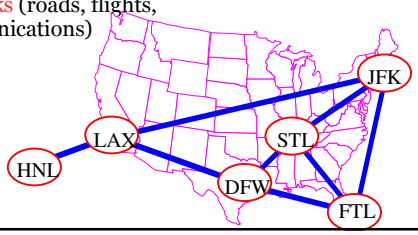
E= {(a,b),(a,c),(a,d), (b,e),(c,d),(c,e), (d,e)}

Applications

- electronic circuits



- networks (roads, flights, communications)



Terminology: Adjacent and Incident

- If (v_0, v_1) is an edge in an undirected graph,
  - v_0 and v_1 are adjacent
  - The edge (v_0, v_1) is incident on vertices v_0 and v_1
- If <v_0, v_1> is an edge in a directed graph
  - v_0 is adjacent to v_1, and v_1 is adjacent from v_0
  - The edge <v_0, v_1> is incident on v_0 and v_1

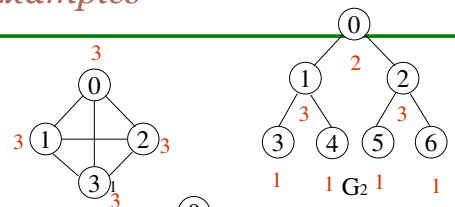
Terminology: Degree of a Vertex

- The degree of a vertex is the number of edges incident to that vertex
- For directed graph,
  - the in-degree of a vertex v is the number of edges that have v as the head
  - the out-degree of a vertex v is the number of edges that have v as the tail
  - if d_i is the degree of a vertex i in a graph G with n vertices and e edges, the number of edges is

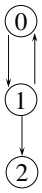
$$e = (\sum_{0}^{n-1} d_i) / 2$$

Why? Since adjacent vertices each count the adjoining edge, it will be counted twice

Examples



directed graph
in-degree
out-degree



in:1, out: 1

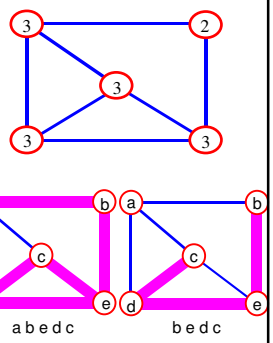
in: 1, out: 2

in: 1, out: 0

G3

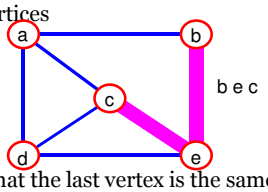
Terminology: Path

- path: sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.

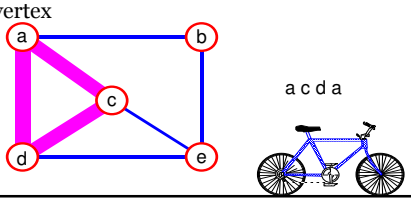


More Terminology

- simple path: no repeated vertices

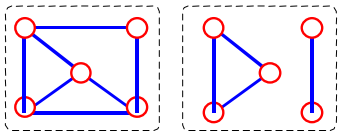


- cycle: simple path, except that the last vertex is the same as the first vertex



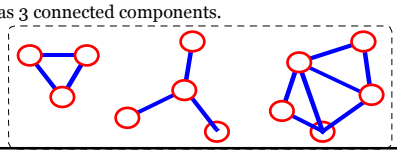
Even More Terminology

- connected graph: any two vertices are connected by some path

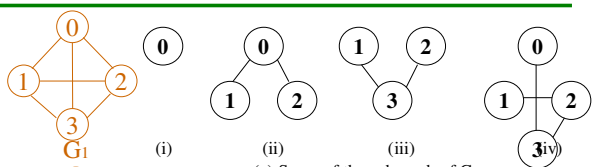


connected not connected

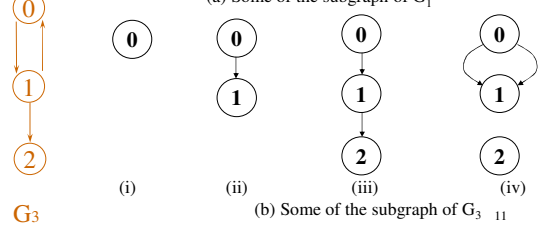
- subgraph: subset of vertices and edges forming a graph
- connected component: maximal connected subgraph. E.g., the graph below has 3 connected components.



Subgraphs Examples



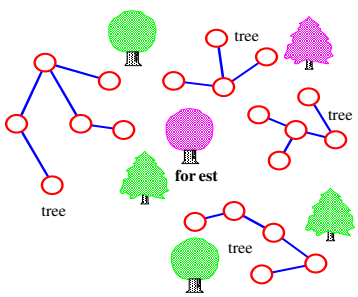
(a) Some of the subgraph of G_1



(b) Some of the subgraph of G_3

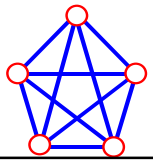
More...

- tree - connected graph without cycles
- forest - collection of trees



Connectivity

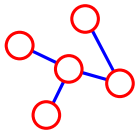
- Let **n** = #vertices, and **m** = #edges
- **A complete graph**: one in which all pairs of vertices are adjacent
- *How many total edges in a complete graph?*
  - Each of the n vertices is incident to **n**-1 edges, however, we would have counted each edge twice! Therefore, intuitively, m = **n**(**n** -1)/2.
- Therefore, if a graph is not complete, m < **n**(**n** -1)/2



n = 5
m = (5 * 4)/2 = 10

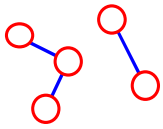
More Connectivity

n = #vertices
m = #edges
- For a tree **m** = **n** - 1



n = 5
m = 4

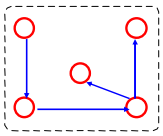
If **m** < **n** - 1, G is not connected



n = 5
m = 3

Oriented (Directed) Graph

- A graph where edges are directed



Directed vs. Undirected Graph

- An undirected graph is one in which the pair of vertices in a edge is unordered, (v_0, v_1) = (v_1,v_0)
- A directed graph is one in which each edge is a directed pair of vertices, <v_0, v_1> != <v_1,v_0>

tail → head

ADT for Graph

objects: a nonempty set of vertices and a set of undirected edges, where each edge is a pair of vertices
functions: for all *graph* ∈ *Graph*, *v*, v_1 and v_2 ∈ *Vertices*
Graph Create()::=return an empty graph
Graph InsertVertex(*graph*, *v*)::= return a graph with *v* inserted. *v* has no incident edge.
Graph InsertEdge(*graph*, v_1,v_2)::= return a graph with new edge between v_1 and v_2
Graph DeleteVertex(*graph*, *v*)::= return a graph in which *v* and all edges incident to it are removed
Graph DeleteEdge(*graph*, v_1, v_2)::=return a graph in which the edge (v_1, v_2) is removed
Boolean IsEmpty(*graph*)::= if (*graph*==*empty graph*) return TRUE else return FALSE
List Adjacent(*graph*,*v*)::= return a list of all vertices that are adjacent to *v*

Graph Representations

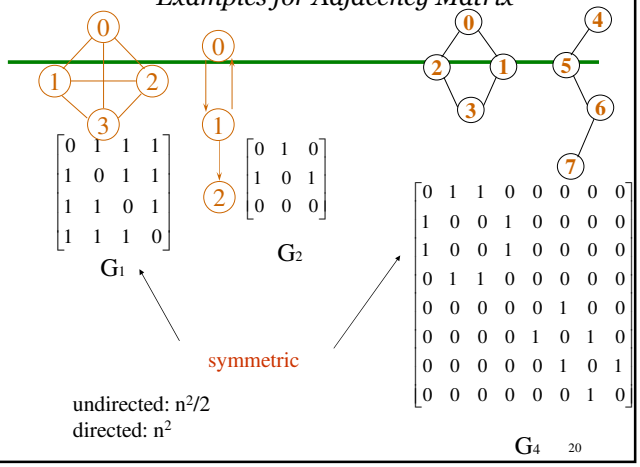
- Adjacency Matrix
- Adjacency Lists

Adjacency Matrix

- Let G=(V,E) be a graph with n vertices.
- The adjacency matrix of G is a two-dimensional n by n array, say adj_mat
- If the edge (v_i, v_j) is in E(G), adj_mat[i][j]=1
- If there is no such edge in E(G), adj_mat[i][j]=0
- The adjacency matrix for an undirected graph is symmetric; the adjacency matrix for a digraph need not be symmetric

19

Examples for Adjacency Matrix



undirected: $n^2/2$
directed: n^2

20

Merits of Adjacency Matrix

- From the adjacency matrix, to determine the connection of vertices is easy
- The degree of a vertex is $\sum_{j=0}^{n-1} adj\_mat[i][j]$
- For a digraph (= directed graph), the row sum is the out_degree, while the column sum is the in_degree

$$ind(vi) = \sum_{j=0}^{n-1} A[j,i] \quad outd(vi) = \sum_{j=0}^{n-1} A[i,j]$$

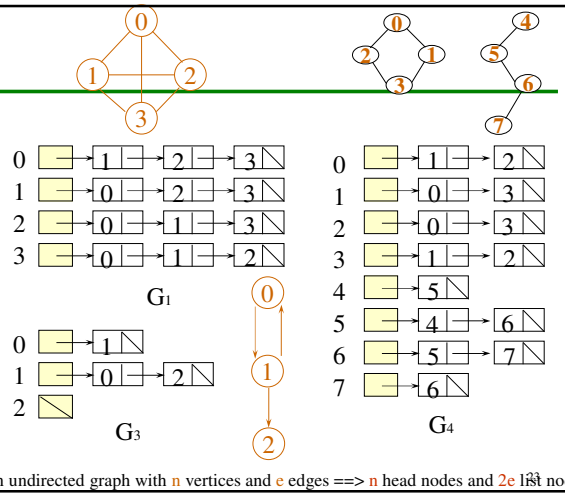
21

Adjacency Lists (data structures)

Each row in adjacency matrix is represented as an adjacency list.

```
#define MAX_VERTICES 50
typedef struct node *node_pointer;
typedef struct node {
    int vertex;
    struct node *link;
};
node_pointer graph[MAX_VERTICES];
int n=0; /* vertices currently in use */
```

22



An undirected graph with n vertices and e edges ==> n head nodes and 2e list nodes

23

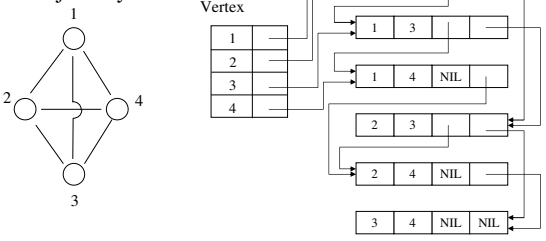
Some Operations

- degree of a vertex in an undirected graph
  - –# of nodes in adjacency list
- # of edges in a graph
  - –determined in O(n+e)
- out-degree of a vertex in a directed graph
  - –# of nodes in its adjacency list
- in-degree of a vertex in a directed graph
  - –traverse the whole data structure

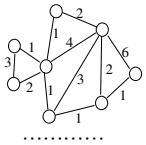
24

1. Definitions and Representations

- Representations
  - Adjacency Matrix
  - Adjacency Lists
  - Adjacency Multi-lists



2. Min-Cost Spanning Tree



$G=(V, E, W)$

weight

- *Greedy*→ (1) Sort E in the non-decreasing order of weights

  $O(e\log e) \leftarrow n^2 \log n$ Why?

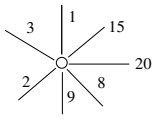
  (2) Choose $n-1$ edges from the sorted list of E as long as the current selected edge does not form a cycle with the previously-chosen edges

  $O(e\log e + eG(n))$

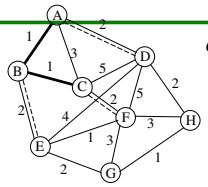
  Kruskal's algorithm !!!

  Any Better way ?

Observation



Why not local search ?



$G = (V, E, W)$

tree vertices = {A, B, C} $=^{\Delta} V_T$

fringe vertices = {E,F,D} $=^{\Delta} V_F$

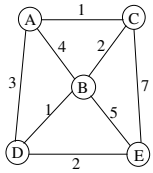
unseen vertices = {G, H} $=^{\Delta} V_U$

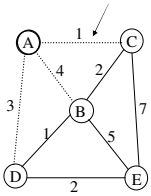
CE = {{A,D},{B,E},{C,F}}

Dijkstra / Prim Algorithm

Select an arbitrary vertex to start the tree;
WHILE there are fringe vertices **do**
    select an edge of minimum weight between a tree vertex and a fringe vertex;
    add the selected edge and the fringe vertex to the tree;
end

Dijkstra / Prim Algorithm

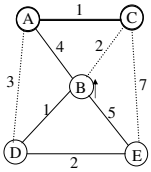




V_T = { A }

V_F = { C, B, D }

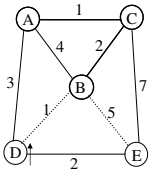
CE = { { A, C }, { A, B }, { A, D } }
1 4 3



V_T = { A, C }

V_F = { B, D, E }

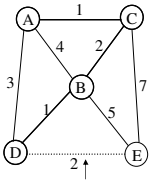
CE = { { A, D }, { C, B }, { C, E } }
3 2 7



V_T = { A, B, C }

V_F = { D, E }

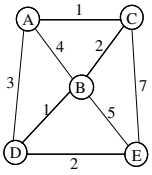
CE = { { B, D }, { B, E } }
1 5



V_T = { A, B, C, D }

V_F = { E }

CE = { { D, E } }



V_T = { A, B, C, D, E }

V_F = Ø

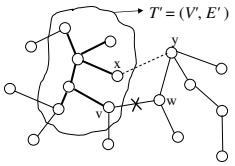
CE = Ø

SI { Select an arbitrary vertex to start the tree }
$v_T = \emptyset$, $v_F = \emptyset$, $CE = \emptyset$;
$v_T = v_T \bigcup \{v\}$;
Update v_F *and* CE ;
while $v_F \neq \emptyset$ do ;
e : = min - weight edge in CE ;
Update v_T, v_F, CE ;
end {while}

No sorting !!!
Why this algorithm works ?

Lemma : Let $G = (V, E, W)$ be a connected weighted graph and $E' \subseteq E$ be a subset of the edges in some minimum spanning tree $T = (V, E_T)$ for G. Let V' be the vertices incident with edges in E'. If $\{x, y\}$ is an edge of minimum weight such that $x \in V'$ and $y \notin V'$, then $E' \cup \{x, y\} \subseteq E_T$.

[Proof]



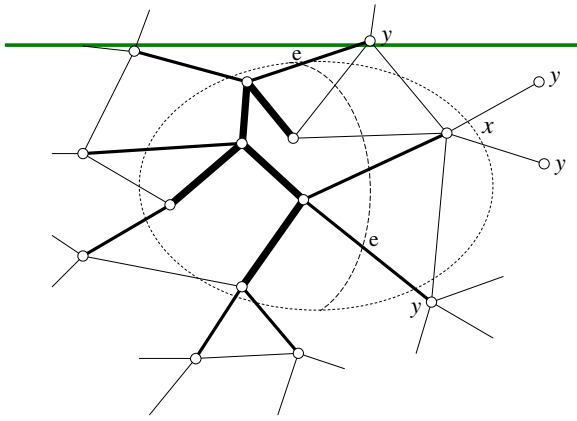
$T' \subseteq T$

$T = (V, E_T)$

i) $\{x, y\} \in E_T$ trivial

ii) $\{x, y\} \notin E_T$

See the figure.



Algorithm : *Minimum spanning tree*(Dijkstra/Prim)

Input : $G = (V,E,W)$, a weighted graph.

Output : The edges in a minimum spanning tree.

1. { Initialization }
   Let x be an arbitrary vertex ;
   $E_T := \emptyset$; *stuck* := false ; $V_T := \{x\}$;
2. { Main loop ; x has just been brought into the tree. Update fringe and candidates. }
   while $V_T \neq V$ and not *stuck* do
3. { Replace some candidate edges. }
   for each fringe vertex y adjacent to x do
   if $W(xy) < W$(the candidate edge e incident with y) then
   xy replaces e as the candidate edge for y ;
   end { if }
   end{for}

   Update candidate edges
4. { Find new fringe vertices and candidate edges. }
   for each unseen y adjacent to x do
   y is now a fringe vertex and xy is a candidate ;
   end { for } ;

   Add new fringe vertices and edges. (candidate)
5. { Ready to choose next edge. }
   if there are no candidates then *stuck* := true { no spanning tree} ;
   else
6. { Choose next edge. }
   Find a candidate edge e, with minimum weight ;
   x := the fringe vertex incident with e.
   Add x and e to the tree.
   { x and e are no longer fringe and candidate. }
   end { if }
   end { while }

   Choose new edges.

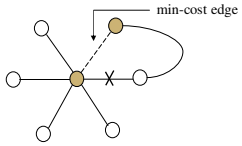
Time complexity

Kruskal	$O(e \log e + eG(e))$
Dijkstra & Prim	$O(e + n^2)$
	$O(e + n \log n)$

Theorem : Dijkstra/Prim Algorithm correctly constructs a min-cost spanning tree.

[proof] (By induction on # of edges chosen)

($m = 1$)



($m = k$) Assume that all the edges so far selected are in a min-cost spanning tree for G.

($m = k+1$) By the previous Lemma, the result follows.

Algorithm : *Minimum spanning tree (Dijkstra/Prim)*

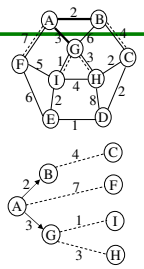
1. { Initialization }
   Let x be an arbitrary vertex ;
   $E_T = \emptyset$; *stuck* := false ; $V_T = \{x\}$;
2. { Main loop ; x has just been brought into the tree.
   Update fringe and candidates. }
   while $V_T \neq V$ and not *stuck* do

   $O(n)$
3. { Replace some candidate edges. }
   for each fringe vertex y adjacent to x do
   if $W(xy) < W$(the candidate edge e incident with y) then
   xy replaces e as the candidate edge for y ;
   end { if }
   end{for}
4. { Find new fringe vertices and candidate edges. }
   for each unseen y adjacent to x do
   y is now a fringe vertex ;
   xy is now a candidate ;
   end { for } ;

   $O(|E|)$
5. { Ready to choose next edge. }
   if there are no candidates then *stuck* := true { no spanning tree} ;
   else
6. { Choose next edge. }
   Find a candidate edge e, with minimum weight ;
   x := the fringe vertex incident with e.
   Add x and e to the tree.
   { x and e are no longer fringe and candidate. }
   end { if }
   end { while }

   $O(n^2)$

$\therefore O(|E|+n^2)$



	fringeLink	*fringeWgt*	*parent*	*status*	*adjacencyList*
A				*intree*	
B	nil	2	*A*	*intree*	
C	*F*	4	*B*	*fringe*	
D				*unseen*	
E				*unseen*	
F	nil	7	*A*	*fringe*	
G	*F*	3	*A*	*intree*	
H	*C*	3	*G*	*fringe*	
I	*H*	1	*G*	*fringe*	

(Adjacency lists are not shown. Nodes are assumed to be in alphabetical order within each list.)

Shaded entries in *fringeLink* are no longer in use. *fringeList* = *I*.

[Minimum spanning tree data structure]

Lower Bound

$\Omega(|E|)$

Why ?

(1) a trivial lower bound $O(|V| + |E|) = O(|E|)$

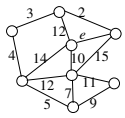
(2) adversary argument

Every edge is required to be examined at least once.

Why ?

Suppose that some edge e is not examined at all.

(see below)



$G = (V, E)$

$w(f) \geq 2 \quad \forall f \in E/\{e\}$

e must not be contained in a min-cost spanning tree.

why ?

Now, $w(e) \triangleq 1 \Longrightarrow$ Contradiction ! why ?

$\therefore \Omega(|E|)$

3. A shortest path algorithm

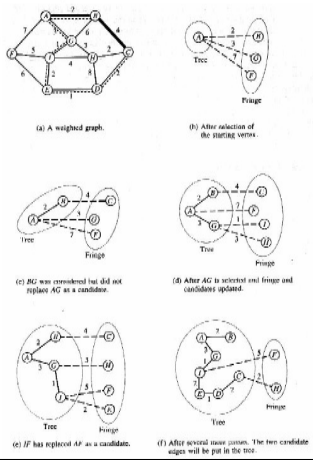
P : Given $G = (V, E, W)$ and $s,d \in V$,
find a shortest path between s and d.

Does the Dijkstra/Prim algorithm also work for solving the shortest path finding problem?

No !!!

Why ?

See a shortest path from A to C !!!



Dijkstra's Algorithm

Step 0

{ initialization }

$v_1 := 0$

$v_j := w_{1j}, \ j = 2, 3, 4, \cdots, n$

$p := \{1\}, \ \boxed{T := \{2, 3, 4, \cdots, n\}}$

Step 1

{ Designation of permanent of label }

find k such that $v_k = \min_{j \in T} \{ v_j \}$

$T := T \setminus \{k\}$

$P := P \cup \{k\}$

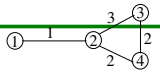
if $T = \emptyset$, stop (if $k = d$, stop)

Step 2

{ Revision of Tentative label }

$v_j := \min\{ v_j, v_k + w_{kj}) \quad \boxed{\forall j \in T}$

go to step 1.



$s \equiv 1 \quad d \equiv 4$

$w_{12} = 1, w_{23} = 3, w_{24} = 2, w_{34} = 2$

$v_1 := 0, v_2 = 1, v_3 = \infty, v_4 = \infty$

$P := \{1\}, \quad T := \{2, 3, 4\}$

$\min_{j \in T}\{v_j\} = \min\{v_2, v_3, v_4\}$

1 ∞ ∞

$\therefore k=2$

Is k=4 ? No !!!

$T := \{2,3,4\}/\{2\} = \{3,4\}$

$P := P \cup \{2\} = \{1,2\}$

$v_3 = \min\{v_3, v_2+w_{23}\} = \min\{\infty, 1+3\} = 4$

$v_4 = \min\{v_4, v_2+w_{24}\} = \min\{\infty, 1+2\} = 3$

$\min_{j \in T}\{v_j\} = \min\{v_3, v_4\} = \min\{4, 3\} = v_4$

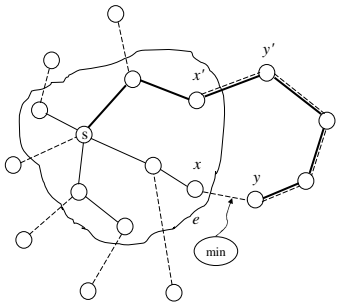
$\therefore k = 4$

$T := \{3,4\}/\{4\} = \{3\}$

$P := P \cup \{4\} = \{1,2,4\}$

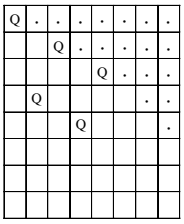
Is k=4 ? Yes !!!

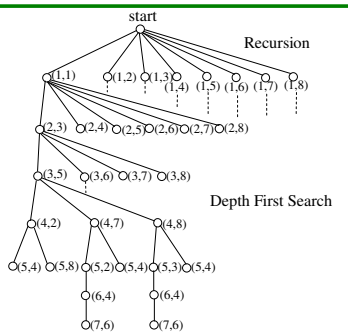
Why does Dijkstra's Algorithm Work ?



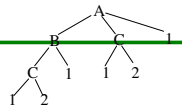
4. Traversing Graphs and Digraphs

8 Queen Problem





Depth First Search

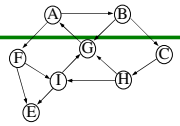


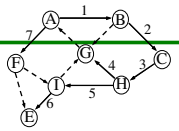
A
- B
  - C
    - 1
    - 2
  - 1
- C
  - 1
  - 2
- 1

Depth First Search

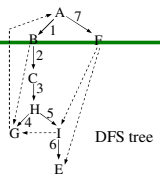
A
- B
- C
- 1
  - C
  - 1
  - 1
  - 2
    - 1
    - 2

Breadth First Search



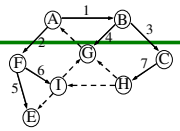


Depth First Search

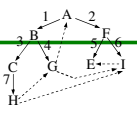


DFS tree

```
procedure DFS (AdjList : HeaderList;  v : VertexType)
    var
        s: Stack
        w: VertexType
    begin
        s:= Ø
        visit, mark, and stack v;
        while s!= Ø do
            while there are unmarked vertex w adjacent to Top(s) do
                    visit, mark, and stack w
            end{while};
            pop s;
    end{outer while}
end{DFS}
```

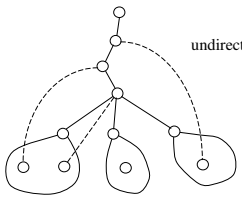
Breadth First Search



BFS tree

```
procedure BFS(AdjList : HeaderList;  v : VertexType)
    var
        Q: Queue
        w: VertexType
    begin
        Q:= Ø
        visit and mark v; insert v in Q;
        while Q≠ Ø do
            x := Front(Q);
            for each unmarked vertex w adjacent to x do
                    visit and mark w;
                    insert w in Q;
            end{for};
        end{while}
    end{BFS}
```

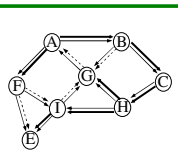
Depth First Search

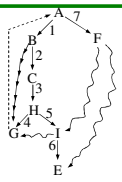


undirected graphs

Tree edges
Back edges
(no other types)

Depth First Search Tree (Directed Graph)





Tree edges
Back edges
Cross edges
Descendant edges
(forward)

why not in undirected graph?

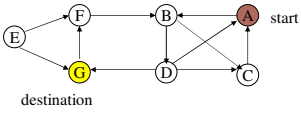
Note : (1) top → down
(2) left → right

Applications: Finding a Path

- Find path from source vertex s to destination vertex d
- Use graph search starting at s and terminating as soon as we reach d
  - Need to remember edges traversed
- Use depth – first search ?
- Use breath – first search?

DFS vs. BFS

DFS Process

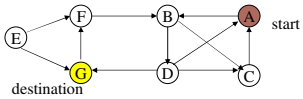


A	DFS on A	B A	DFS on B	C B A	DFS on C	D B A	Call DFS on D Return to call on B

G D B A	Call DFS on G	found destination - done! Path is implicitly stored in DFS recursion Path is: A, B, D, G

DFS vs. BFS

BFS Process



rear front	rear front	rear front	rear front
A	B	D C	D
Initial call to BFS on A Add A to queue	Dequeue A Add B	Dequeue B Add C, D	Dequeue C Nothing to add

rear front

G

Dequeue D
Add G

found destination - done!
Path must be stored separately

บรรณานุกรม

กฤษณะ ชินสาร, *เอกสารประกอบการสอนวิชา Computer Algorithm: Analysis and Design*, คณะวิทยาการสารสนเทศ มหาวิทยาบูรพา, 2553

ธีรวัฒน์ ประกอบผล, *คู่มือการเขียนโปรแกรม ภาษา C*, สำนักพิมพ์ ซิม พลิ ฟาย, 2553.

ธีรวัฒน์ ประกอบผล, *การวิเคราะห์และออกแบบระบบ*, สำนักพิมพ์ ซัค เซส มีเดีย, 2552

ประกาศสมาคมวิทยาศาสตร์แห่งประเทศไทยในพระบรมราชูปถัมภ์

ครั้งที่ 11/2553

เรื่อง รายชื่อคณะอนุกรรมการดำเนินงานโครงการอบรมครูวิทยาศาสตร์ วิชาคอมพิวเตอร์

****

เพื่อให้การดำเนินงานตามโครงการอบรมครูวิทยาศาสตร์ คณิตศาสตร์ คอมพิวเตอร์ และวิทยาศาสตร์โลก เป็นไปด้วยความเรียบร้อย และมีประสิทธิภาพ จึงขอแต่งตั้งคณะอนุกรรมการดำเนินงานโครงการอบรมครูวิทยาศาสตร์ วิชาคอมพิวเตอร์ ดังมีรายชื่อต่อไปนี้

1.	ศาสตราจารย์ ดร.ชิดชนก	เหลือสินทรัพย์	(จุฬาฯ)	ประธานคณะอนุกรรมการ
2.	รองศาสตราจารย์ ดร.อนงค์นาฏ	ศรีวิหค	(เกษตรศาสตร์)	อนุกรรมการ
3.	รองศาสตราจารย์ ดร.จันทนา	ผ่องเพ็ญศรี	(ศิลปากร)	อนุกรรมการ
4.	รองศาสตราจารย์ ไพบูลย์	พันธรักษ์พงษ์	(ลาดกระบัง)	อนุกรรมการ
5.	รองศาสตราจารย์ ธีรวัฒน์	ประกอบผล	(ลาดกระบัง)	อนุกรรมการ
6.	ผู้ช่วยศาสตราจารย์ ดร.กฤษณะ	ชินสาร	(บูรพา)	อนุกรรมการ
7.	ผู้ช่วยศาสตราจารย์ ดร.ชวลิต	ศรีสถาพรพัฒน์	(เกษตรศาสตร์)	อนุกรรมการ
8.	ดร.สุนิสา	ริมเจริญ	(บูรพา)	อนุกรรมการ
9.	อาจารย์ จิระ	จตุรานนท์	(บูรพา)	อนุกรรมการ

ทั้งนี้ตั้งแต่บัดนี้เป็นต้นไป

ประกาศ ณ วันที่ 5 มีนาคม 2553

(ศาสตราจารย์ ดร.ม.ร.ว.ชิษณุสรร สวัสดิวัตน์)

นายกสมาคมวิทยาศาสตร์แห่งประเทศไทยในพระบรมราชูปถัมภ์