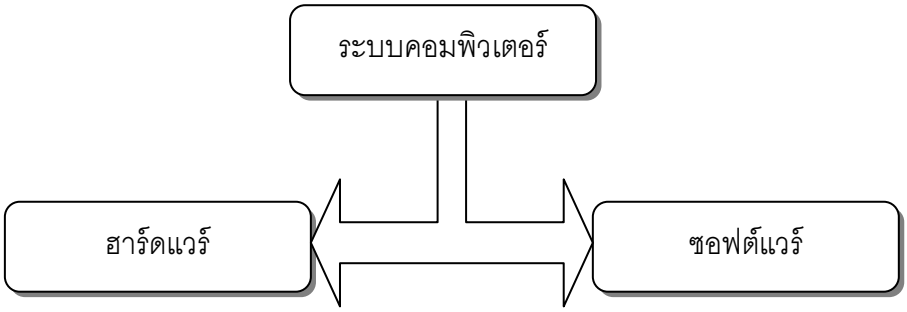
บทที่ 1
องค์ประกอบคอมพิวเตอร์

สังคมปัจจุบันคอมพิวเตอร์เข้ามามีบทบาทอย่างมาก ไม่ว่าจะเป็นการใช้ชีวิตประจำวัน หรือในการทำงานเนื่องด้วยคอมพิวเตอร์เป็นอุปกรร์ที่มีประสิทธิภาพสูง มีความรวดเร็ว และแม่นยำในการทำงาน อีกทั้งราคาสำหรับคอมพิวเตอร์ก็ไม่เกินกำลังความสามารถที่จะซื้อหามาใช้ ดังนั้นบุคลากรทางด้านวิศวกรรมศาสตร์ หรือ วิทยาศาสตร์ ควรเข้าใจเกี่ยวกับโครงสร้างพื้นฐานของคอมพิวเตอร์และการพัฒนาซอฟต์แวร์บนคอมพิวเตอร์เพื่อแก้ไขปัญหาที่มีซับซ้อนให้มีประสิทธิภาพมากขึ้น

1.1 ระบบคอมพิวเตอร์ (Computer systems)

คอมพิวเตอร์คืออุปกรณ์ทางอิเล็กทรอนิกส์ที่ทำงานตามคำสั่งในการจัดการกับข้อมูล ในปัจจุบันระบบคอมพิวเตอร์ได้ถูกแบ่งออกเป็น 2 องค์ประกอบได้แก่ฮาร์ดแวร์ (Hardware) และซอฟต์แวร์ (Software) ซึ่งฮาร์ดแวร์คอมพิวเตอร์จะเป็นอุปกรณ์ทางกายภาพที่ประกอบขึ้น ส่วนซอฟต์แวร์คือโปรแกรมที่ถูกสร้างขึ้นมาเพื่อสั่งให้ฮาร์ดแวร์ทำงานได้ตามที่ต้องการ รูปที่ 1.1 แสดงระบบคอมพิวเตอร์

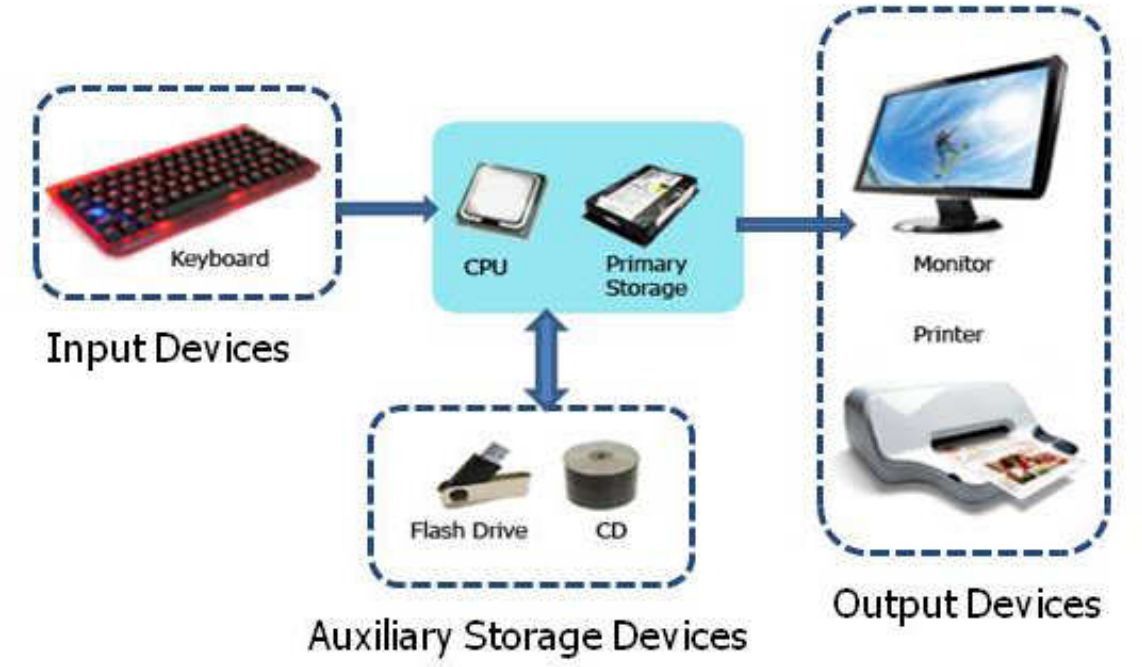


รูปที่ 1.1 ระบบคอมพิวเตอร์

1.2 ฮาร์ดแวร์คอมพิวเตอร์ (Computer hardware)

องค์ประกอบฮาร์ดแวร์ของระบบคอมพิวเตอร์ประกอบไปด้วย 4 ส่วนหลักได้แก่ดังรูปที่ 1.2

- ตัวประมวลผลกลาง (Central processing unit หรือเรียกสั้นๆว่า ซีพียู (CPU))
- หน่วยความจำ (Memory) ซึ่งรวม Primary storage และ Auxiliary storage
- อุปกรณ์อินพุต (Input Device)
- อุปกรณ์เอาต์พุต (Output Device)

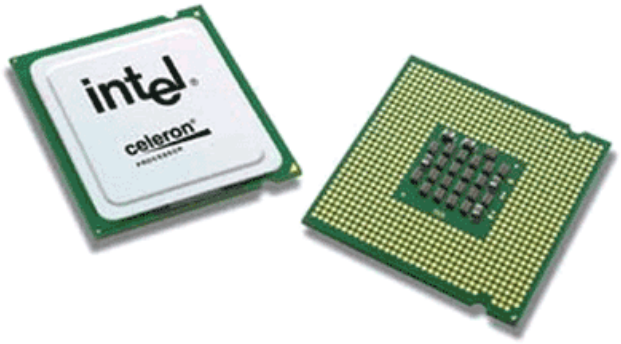


รูปที่ 1.2 องค์ประกอบฮาร์ดแวร์เบื้องต้น

1.2.1 ซีพียู

Central Processing Unit (CPU) เป็นหน่วยประมวลผลกลางซึ่งถือว่าเป็นหัวใจสำคัญของระบบ รูปร่างของซีพียูแสดงในรูปที่ 1.3 ซีพียูเป็นอุปกรณ์ที่ทำงานตามชุดคำสั่งที่กำหนดให้มีความสามารถในการคำนวณทางคณิตศาสตร์ ตรรกศาสตร์และสามารถติดต่อรวมทั้งควบคุมอุปกรณ์ต่างๆได้ จำนวนบิตของซีพียูจะเป็นตัวเลขที่บอกขนาดความยาวของรีจิสเตอร์ในการประมวลผล จำนวนบิตของซีพียูที่ใช้กันเป็นส่วนใหญ่คือ 32 บิต แต่ปัจจุบันซีพียูที่วางจำหน่ายส่วนใหญ่มีขนาด 64 บิต ซึ่งแนวโน้มในอนาคตซีพียู 32 บิตจะหมดไปเหลือแต่เพียงซีพียูที่มีขนาด 64 บิต

ในอดีตความเร็วของซีพียูจะบอกเป็นความเร็วของสัญญาณนาฬิกาที่ป้อนให้กับตัวซีพียู เช่น Pentium4 2GHz, Celeron 1.2 GHz เป็นต้น แต่เนื่องจากการเพิ่มความถี่ของสัญญาณนาฬิกาได้ไปถึงจุดอิ่มตัว เนื่องจากความร้อนและเป็นตัวก่อเกิดสัญญาณรบกวนในเครื่องคอมพิวเตอร์ ปัจจุบันจึงมีการเปลี่ยนแปลงสถาปัตยกรรมภายในของซีพียูโดยเพิ่มหน่วยประมวลผลที่เรียกว่า ALU (Arithmetic Logic Unit) มากกว่าหนึ่งหน่วยลงไปในซีพียู ปัจจุบันที่เรารู้จักกันว่า คอร์ (Core) เช่น Intel Core 2 duo หมายถึง ซีพียู ที่มีจำนวนคอร์ 2 คอร์ (2 ALU) ในซีพียู



รูปที่ 1.3 ลักษณะของซีพียู

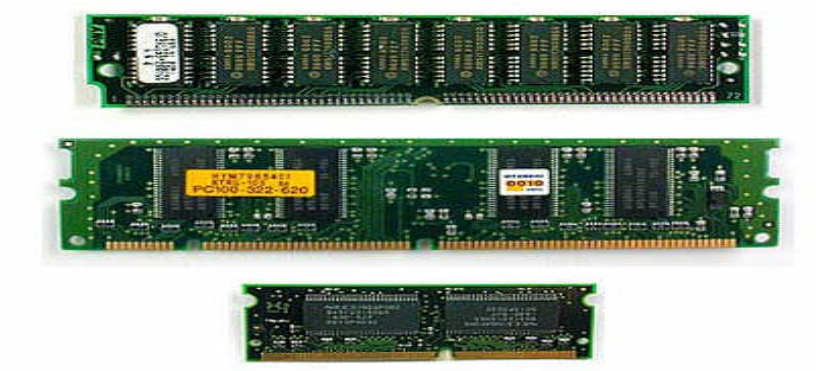
1.2.2 **หน่วยความจำ**

หน่วยความจำของคอมพิวเตอร์สามารถแบ่งออกเป็น 2 ชนิดคือ หน่วยความจำหลัก และหน่วยความจำสำรอง

1.2.2.1 **หน่วยความจำหลัก** (Primary Storage)

หน่วยความจำหลักในคอมพิวเตอร์มีหน้าที่เก็บชุดคำสั่งที่ใช้สั่งงานซีพียู และ เก็บข้อมูลต่างๆที่จะให้ซีพียูประมวลผลรวมทั้งเก็บผลลัพธ์ที่ได้จากการประมวลผล หน่วยความจำหลักในคอมพิวเตอร์เรียกว่า RAM (Random Access Memory) แสดงในรูปที่ 1.4 ซึ่งข้อมูลจะถูกจัดเก็บเป็นบิตซึ่งแต่ละบิตมีค่าเป็น '0' หรือ '1' เช่น 10111011 ซึ่งจะเรียกว่าเท่ากับ 8 บิต หน่วยที่ใช้เรียกขนาดความจุของหน่วยความจำมีดังนี้

1 ไบต์	เท่ากับ	8	บิต
1 กิโลไบต์ (1 K Byte)	เท่ากับ	1024	ไบต์
1 เมกะไบต์ (1 M Byte)	เท่ากับ	1024	กิโลไบต์
1 จิกกะไบต์ (1 G Byte)	เท่ากับ	1024	เมกะไบต์



รูปที่ 1.4 หน่วยความจำหลัก (RAM)

(ก)

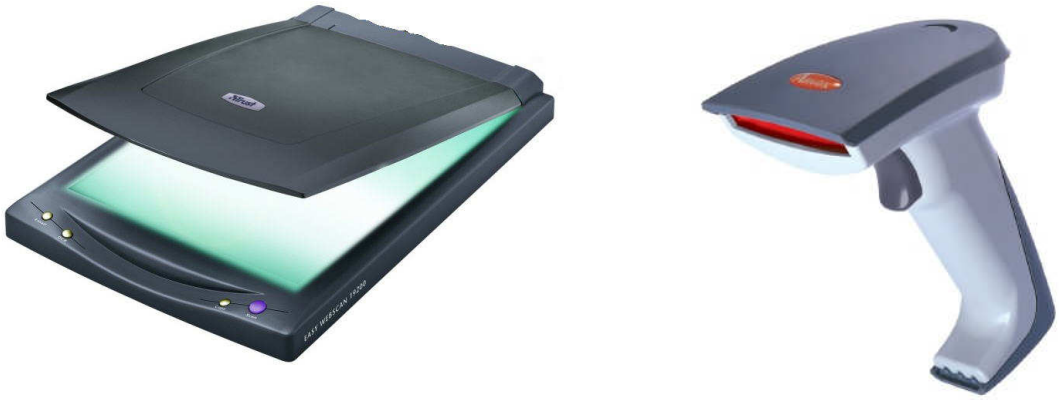
(ข)

(ค)

รูปที่ 1.5 หน่วยความจำสำรอง (ก) Hard disk, (ข) แผ่น CD และ (ค) Flash Drive

1.2.2.2 **หน่วยความจำสำรอง** (Auxiliary storage)

หน่วยความจำสำรองบางทีเรียกกันว่า Secondary memory หมายถึงหน่วยความจำหรือหน่วยเก็บที่ใช้เก็บข้อมูลนอกเครื่องคอมพิวเตอร์ (เพิ่มไปจากหน่วยความจำหลัก) เช่น จานบันทึก เมื่อใดที่ต้องการใช้ ก็ให้เครื่องคอมพิวเตอร์นำมาเก็บในเครื่อง จะแก้ไขเปลี่ยนแปลงเพิ่มเติม แล้วเก็บลงใหม่ก็ได้ ทั้งนี้เพื่อมิให้สิ้นเปลืองหน่วยความจำหลักในเครื่องคอมพิวเตอร์ ซึ่งสามารถใช้เก็บข้อมูลแบบถาวรได้ แต่การทำงานจะช้ากว่าหน่วยความจำหลักแต่มีราคาที่ถูกกว่า หน่วยความจำสำรองเช่น Hard disk, CD-ROM และ Flash drive แสดงในรูปที่ 1.5



รูปที่ 1.6 ตัวอย่างอุปกรณ์อินพุต Scanner (ซ้าย) และ Barcode Reader (ขวา)

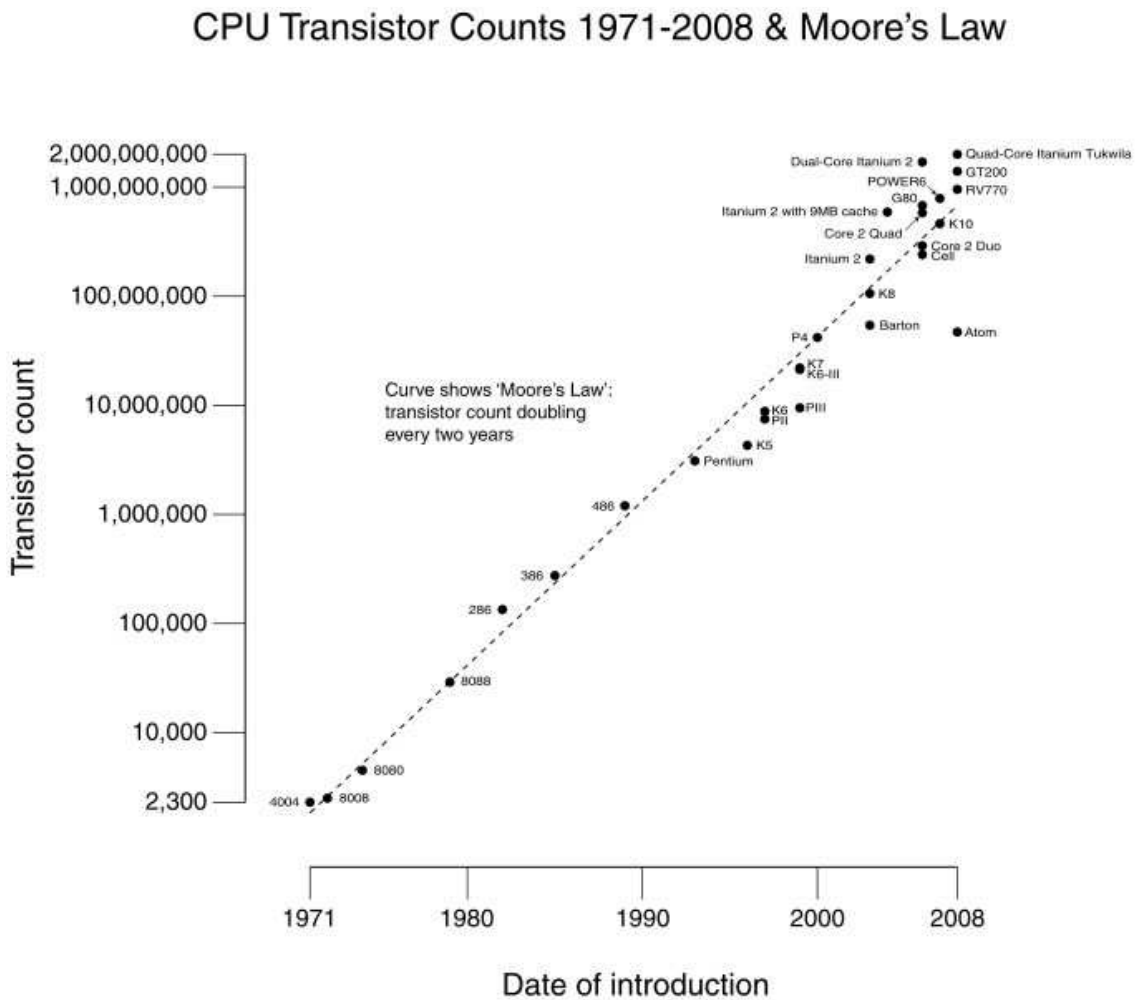
รูปที่ 1.7 ตัวอย่างอุปกรณ์เอาต์พุต Printer (ซ้าย) และ Speaker (ขวา)

1.2.3 อุปกรณ์อินพุต

เป็นอุปกรณ์ที่ใช้รับข้อมูลในรูปแบบต่างๆ จากภายนอกคอมพิวเตอร์ แล้วแปลงข้อมูลที่รับให้อยู่ในรูปแบบ Binary ซึ่งสามารถสื่อสารกับซีพียูได้ ตัวอย่างอุปกรณ์อินพุตแสดงในรูปที่ 1.6 ด้านซ้ายคือเครื่องแสกน(Scanner) และด้านขวาคือเครื่องอ่านบาร์โค้ด(Barcode Reader) แต่อุปกรณ์อินพุตยังมีอีกมากมายหลายชนิดเช่น คีย์บอร์ด(Keyboard), เมาส์(Mouse), Trackball, Joystick ฯลฯ

1.2.4 อุปกรณ์เอาต์พุต

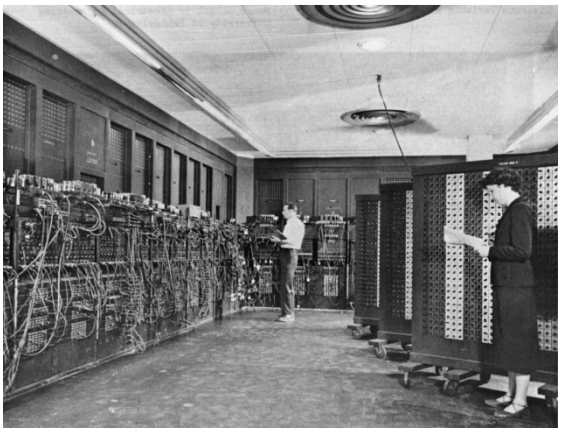
เป็นอุปกรณ์ที่แปลงข้อมูลแบบ Binary จากซีพียู/คอมพิวเตอร์ไปเป็นข้อมูลในรูปแบบอื่นหรือให้อุปกรณ์นั้นทำงานอย่างใดอย่างหนึ่งตามลักษณะข้อมูล ตัวอย่างอุปกรณ์เอาต์พุตแสดงในรูปที่ 1.7 ด้านซ้ายคือ เครื่องพิมพ์ (Printer) และด้านขวาคือ ลำโพง (Speaker) อุปกรณ์เอาต์พุตยังมีอีกมากมายหลายชนิด เช่น จอภาพ (Monitors) และเป็นต้น



รูปที่ 1.8 จำนวนทรานซิสเตอร์ที่เพิ่มขึ้นในซีพียูตามกฎของมัวร์

1.3 **วิวัฒนาการของคอมพิวเตอร์** (Computer Evolution)

วิวัฒนาการของคอมพิวเตอร์นั้น คอมพิวเตอร์จะมีขนาดที่เล็กลง ทำงานเร็วขึ้น ใช้พลังงานน้อยลง มีความจุมากขึ้น และ ราคาถูกลง ซึ่งเป็นไปตามกฎของมัวร์ (Moore's Law) ที่ว่า จำนวนของทรานซิสเตอร์ที่สามารถใส่ในตัว IC (Integrated Circuit) จะเพิ่มขึ้นเท่าตัวทุกๆ 2 ปี รูปที่ 1.8 แสดงจำนวนทรานซิสเตอร์ในหน่วยประมวลผลกลางที่เพิ่มขึ้นในแต่ละปีตามกฎซึ่งจะเห็นได้ว่าในปี ค.ศ. 1990 เมื่อซีพียูอยู่ในรุ่นของ 486 ในขณะนั้นมีจำนวนทรานซิสเตอร์ภายในประมาณ 1 ล้านตัว ในขณะที่ปัจจุบันซีพียู Core 2 quad มีจำนวนทรานซิสเตอร์ภายในถึงประมาณ 1 พันล้านตัว รูปที่ 1.9 (ก) แสดงคอมพิวเตอร์ที่เรียกกันว่าเป็นเครื่องคอมพิวเตอร์เครื่องแรกของโลก ENIAC ซึ่งต้องวางในห้องที่มีขนาด 63 ตารางเมตร และมีน้ำหนักมากถึง 27 ตัน ซึ่งในปัจจุบันมีวิวัฒนาการมาเป็นเครื่องคอมพิวเตอร์พกพาขนาดเล็ก (netbook) ที่มีขนาดเล็กและมีน้ำหนักไม่ถึง 1 กิโลกรัมดังรูปที่ 1.9 (ข)



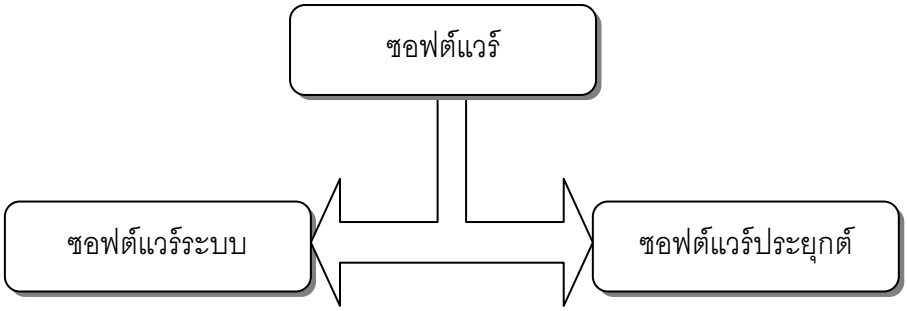
(ก)

(ข)

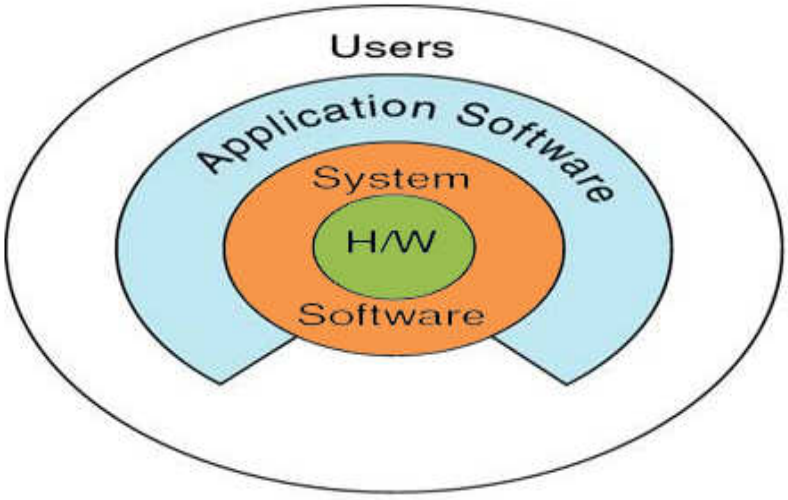
รูปที่ 1.9 เครื่องคอมพิวเตอร์ (ก) ENIAC และ (ข) Netbook

1.4 ซอฟต์แวร์คอมพิวเตอร์ (Computer software)

ซอฟต์แวร์คอมพิวเตอร์ถูกแบ่งออกเป็น 2 ประเภทใหญ่ ได้แก่ ซอฟต์แวร์ระบบ (System software) และซอฟต์แวร์ประยุกต์ (Application software) ดังรูปที่ 1.10 ซึ่งซอฟต์แวร์ระบบมีหน้าที่บริหารจัดการกับระบบฮาร์ดแวร์ของคอมพิวเตอร์ ในขณะที่ซอฟต์แวร์ประยุกต์เป็นซอฟต์แวร์ที่ถูกสร้างขึ้นโดยเจาะจงลงไปให้เหมาะสมกับการนำไปใช้งาน รูปที่ 1.11 แสดงประเภทของซอฟต์แวร์คอมพิวเตอร์และความสัมพันธ์ระหว่างซอฟต์แวร์ระบบกับซอฟต์แวร์ประยุกต์ แกนชั้นในสุดคือฮาร์ดแวร์ (H/W) และมีแกนนอกสุดเป็นผู้ใช้ (Users) ซึ่งจะติดต่อกับฮาร์ดแวร์โดยผ่านซอฟต์แวร์ประยุกต์จากนั้นจะผ่านซอฟต์แวร์ระบบไปสั่งงานฮาร์ดแวร์



รูปที่ 1.10 ประเภทของซอฟต์แวร์



รูปที่ 1.11 ความสัมพันธ์ของผู้ใช้ ซอฟต์แวร์ และ ฮาร์ดแวร์

1.4.1 **ซอฟท์แวร์ระบบ** (System Software)

ซอฟต์แวร์ระบบที่เด่นชัดที่สุดคือ โปรแกรมระบบปฏิบัติการได้แก่ DOS, Windows, Linux และ Mac OS ฯลฯ ซึ่งจะทำหน้าที่ในการควบคุมอุปกรณ์ในระบบ เช่น คีย์บอร์ด (Keyboard), หรือ Hard disk ฯลฯ และรวมไปถึงการจัดสรรทรัพยากรและควบคุมการทำงานโปรแกรมประยุกต์ต่างๆที่ทำงานบนระบบคอมพิวเตอร์

1.4.2 **ซอฟต์แวร์ประยุกต์** (Application Software)

ซอฟต์แวร์ประยุกต์เป็นโปรแกรมที่ถูกพัฒนาขึ้นเพื่อการทำงานต่างๆเฉพาะด้าน ไม่ว่าจะเป็นงานด้านวิศวกรรม, ระบบบัญชี, ฐานข้อมูลบุคคล หรือในด้านบันเทิงต่างๆ เช่น ซอฟท์แวร์สำหรับฟังเพลง, ซอฟท์แวร์สำหรับเล่นเกม อีกทั้งยังมีซอฟต์แวร์ประยุกต์ที่เรียกว่า โปรแกรมช่วยงาน (Utilities Program) ซึ่งเป็นโปรแกรมที่ช่วยให้เราทำงานกับคอมพิวเตอร์ ได้ง่ายขึ้น และมีประสิทธิภาพ ตัวอย่างเช่น โปรแกรมจัดการเนื้อที่ Hard Disk, โปรแกรมป้องกันไวรัสคอมพิวเตอร์, โปรแกรมบีบอัดขนาดของแฟ้มข้อมูล เป็นต้น

1.5 การพัฒนาซอฟต์แวร์ (Software Development)

การพัฒนาซอฟต์แวร์หมายถึงขบวนการ การเขียนรหัส/โปรแกรมด้วยภาษาคอมพิวเตอร์ ซึ่งการพัฒนาซอฟต์แวร์แบ่งออกเป็นขั้นตอนต่างๆ ดังนี้

- การกำหนดความต้องการ (ปัญหาที่ต้องการใช้คอมพิวเตอร์ช่วยในแก้ไข)
- การวิเคราะห์ตัวปัญหา ได้แก่การกำหนดข้อมูลขาเข้าและรูปแบบผลลัพธ์ที่ต้องการได้
- การออกแบบ ได้แก่การออกแบบวางแผนถึงวิธีการแก้ไขปัญหาที่จะเกิดขึ้น
- การเขียนโปรแกรม ได้แก่การเขียนโปรแกรมด้วยภาษาทางคอมพิวเตอร์เพื่อแก้ไขปัญหา
- การทดสอบ ได้แก่การทดสอบเพื่อดูผลของการแก้ไขปัญหา
- การปรับปรุงแก้ไขโปรแกรม ได้แก่ในกรณีที่โปรแกรมยังไม่สามารถทำงานได้อย่างสมบูรณ์ ต้องนำไปแก้ไขปรับปรุงใหม่เพื่อให้ได้ผลลัพธ์ที่ถูกต้องตามที่ต้องการ

คำศัพท์เทคนิคที่ใช้ในการพัฒนาซอฟแวร์ในส่วนขั้นตอนการเขียนและทดสอบโปรแกรมมีดังนี้

- Edit คือการเขียนและแก้ไขโปรแกรมด้วยภาษาคอมพิวเตอร์
- Compile คือการแปลโปรแกรมทีเขียนด้วยภาษาคอมพิวเตอร์ให้เป็นภาษาเครื่อง
- Execute หรือ Run คือการสั่งงานภาษาเครื่องที่ได้ให้ทำงาน
- Debug คือการหาจุดผิดพลาดของโปรแกรม
- Integrated Development Environment (IDE) software เป็นเครื่องมือในการพัฒนาซอฟต์แวร์แบบเบ็ดเสร็จสำหรับวงจรการ “edit-compile-execute-debug”

1.6 ภาษาคอมพิวเตอร์ (Computer Languages)

ในการพัฒนาซอฟต์แวร์นั้นจะใช้ภาษาคอมพิวเตอร์(Computer Languages) หรือบางที่เรียกว่าภาษาโปรแกรม (Programming Language) เป็นเครื่องมือในการพัฒนาซึ่งภาษาคอมพิวเตอร์ที่ถูกพัฒนาเสร็จสิ้นแล้วนั้นจะถูกแปลงให้อยู่ในรูปของภาษาเครื่องเพื่อใช้สั่งงานซีพียูให้ทำงานตามที่กำหนด ซึ่งภาษาคอมพิวเตอร์นั้นสามารถแบ่งออกเป็นประเภทใหญ่ๆ 3 ประเภทคือ

- ภาษาโปรแกรมระดับต่ำ (Low-level Programming Language)
- ภาษาโปรแกรมระดับสูง (High-level Programming Language)
- ภาษาโปรแกรมระดับสูงมาก (Very High-level Programming Language)

1.6.1 ภาษาโปรแกรมระดับต่ำ

ภาษาโปรแกรมระดับต่ำนั้นส่วนใหญ่นิยมใช้ภาษาที่เรียกว่าภาษาแอสเซมบลี (Assembly) ในการพัฒนาโปรแกรมด้วยภาษาแอสเซมบลีผู้พัฒนาซอฟต์แวร์จำเป็นต้องมีความรู้เกี่ยวกับสถาปัตยกรรมของซีพียูว่าใช้ซีพียูเบอร์อะไรเพราะซีพียูแต่ละตัวจะมีคำสั่งควบคุมที่เป็นภาษาแอสเซมบลีที่แตกต่างกัน

ข้อดีของการพัฒนาซอฟต์แวร์ด้วยภาษาระดับต่ำคือซอฟต์แวร์ที่ได้จะมีการทำงานที่รวดเร็วและควบคุมอุปกรณ์ได้โดยตรง แต่ข้อเสียคือซอฟต์แวร์ที่พัฒนาขึ้นในซีพียูหนึ่งจะไม่สามารถทำงานได้ในซีพียูรุ่นอื่นที่สถาปัตยกรรมภายในไม่เหมือนกัน ทำให้ถ้าต้องการเปลี่ยนซีพียูจำเป็นต้องพัฒนาซอฟต์แวร์นั้นใหม่อีกครั้ง

1.6.2 ภาษาโปรแกรมระดับสูง

เป็นภาษาโปรแกรมที่มีโครงสร้างของคำสั่งที่เข้าใจง่ายกว่าภาษาระดับต่ำ โดยคำสั่งจะมีลักษณะเป็นคำในภาษาอังกฤษ ข้อดีของการเขียนโปรแกรมด้วยภาษาระดับสูง คือ

- ลักษณะของภาษานั้นใช้คำที่สื่อความหมายในภาษาอังกฤษแทนคำสั่งที่เข้าใจได้ยาก ซึ่งทำให้ผู้ใช้โปรแกรมเข้าใจโปรแกรมได้ง่ายกว่า
- ภาระดับสูงนั้นถูกออกแบบมาให้เป็นภาษาที่มีลักษณะโครงสร้าง และการทำงานที่ง่ายต่อผู้เขียนโปรแกรมในการจะเปลี่ยนจากวิธีการในการแก้โจทย์ปัญหามาเป็นคำสั่งของโปรแกรมคอมพิวเตอร์
- การพัฒนาโปรแกรมคอมพิวเตอร์ด้วยโปรแกรมภาษาระดับสูงนั้น ผู้เขียนโปรแกรมไม่จำเป็นที่จะต้องเข้าใจเกี่ยวกับฮาร์ดแวร์ของซีพียูในระบบคอมพิวเตอร์เท่ากับการเขียนด้วยภาษาระดับต่ำ

ซึ่งภาษาโปรแกรมระดับสูงที่นิยมใช้ในการพัฒนาซอฟต์แวร์ในปัจจุบันได้แก่ ภาษา C, ภาษา Basic, ภาษา Fortran และ ภาษา Java เป็นต้น

1.6.3 ภาษาโปรแกรมระดับสูงมาก

เป็นภาษาโปรแกรมแบบใหม่ที่ช่วยให้การพัฒนาซอฟต์แวร์มีความสะดวกมากขึ้น ส่วนใหญ่จะอยู่ในรูปของ Script โดยส่วนใหญ่ภาษาโปรแกรมระดับสูงมากผู้พัฒนาซอฟต์แวร์ไม่จำเป็นต้องเข้าใจเกี่ยวกับประเภทของข้อมูลที่ต้องการใช้งาน เหมาะสำหรับการพัฒนาเครื่องมือช่วยเหลือต่างๆ แต่อย่างไรก็ตามการทำงานของซอฟต์แวร์ที่ถูกพัฒนาจะทำงานได้ค่อนข้างช้า และการเข้าถึงหรือควบคุมอุปกรณ์ต่างๆในคอมพิวเตอร์ไม่สามารถทำได้สะดวกมากนัก ตัวอย่างของภาษาโปรแกรมระดับสูงมาก เช่น Yacc, Lex, ML เป็นต้น

บทที่ 2
แนะนำภาษาซี

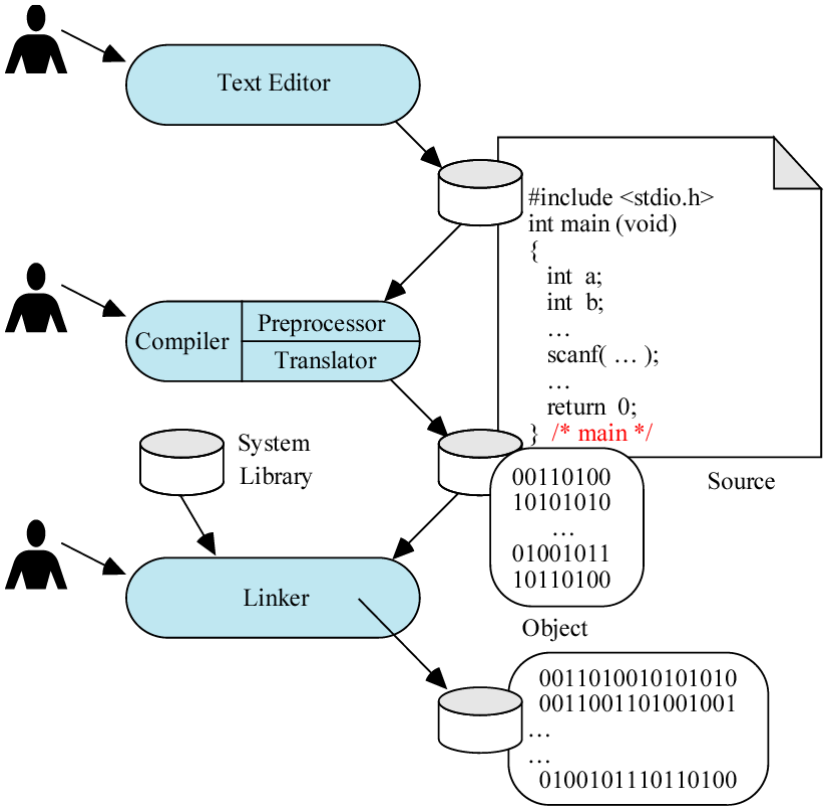
ภาษาซีได้ถูกพัฒนาขึ้นโดยนายเดนนิส ริทชี่ ณ ศูนย์วิจัยเบล ในสหรัฐอเมริกา เมื่อปี ค.ศ. 1972 จนกระทั่งปี ค.ศ. 1978 นายไบรอัน เคอร์นิแกน และนายเดนนิส ริทชี่ ได้จัดทำข้อกำหนดมาตรฐานของภาษาซี ซึ่งนิยมเรียกกันว่า K&R หลังจากนั้นปี ค.ศ. 1980 ภาษาซี ก็ได้รับความนิยมมากขึ้น มีการพัฒนา compiler ภาษาซีออกมามากมาย

2.1 ทำไมต้องภาษาซี

ภาษาซีเป็นภาษาที่มีความใกล้เคียงกับภาษาระดับต่ำ (Low-Level Language) จึงทำให้นักพัฒนาซอฟต์แวร์สามารถที่จะกำหนดรายละเอียดของโปรแกรมให้เข้าถึงการทำงานในส่วนต่าง ๆ ของคอมพิวเตอร์ให้มากที่สุด เพื่อให้เกิดความเร็วในการทำงานสูงสุด และในขณะเดียวกันภาษาซีก็ยังมีความเป็นภาษาระดับสูง (High-Level Language) ทำให้นักพัฒนาสามารถที่จะพัฒนาโปรแกรมได้ โดยเน้นไปที่การแก้ปัญหาที่ต้องการได้อย่างอิสระโดยไม่ต้องคำนึงถึง ฮาร์ดแวร์ใด ๆ ภาษาซีเป็นภาษาโปรแกรมคอมพิวเตอร์ที่ได้รับความนิยมและมีการพัฒนาอย่างต่อเนื่อง ซึ่งระบบปฏิบัติการยอดนิยมส่วนใหญ่ไม่ว่าจะเป็น Windows XP, Linux, UNIX, MAC OS X ก็ถูกพัฒนาขึ้นด้วยภาษาซี อีกทั้งภาษาซียังถูกใช้ในการพัฒนาภาษาระดับสูงอื่นๆ เช่น PHP, Perl, Ruby และ Python รวมถึงภาษาซีเป็นภาษาที่เรียกว่าเป็นต้นตระกูลของภาษายอดนิยมอื่นๆ เช่น C++, Java, PHP ซึ่งจะมีโครงสร้างและไวยากรณ์ของภาษาคล้ายคลึงกับภาษาซี ทำให้ผู้ที่มีความชำนาญกับภาษาซีสามารถทำความเข้าใจกับภาษาใหม่ๆ ได้อย่างรวดเร็ว

2.2 ขั้นตอนการพัฒนาโปรแกรมภาษาซี

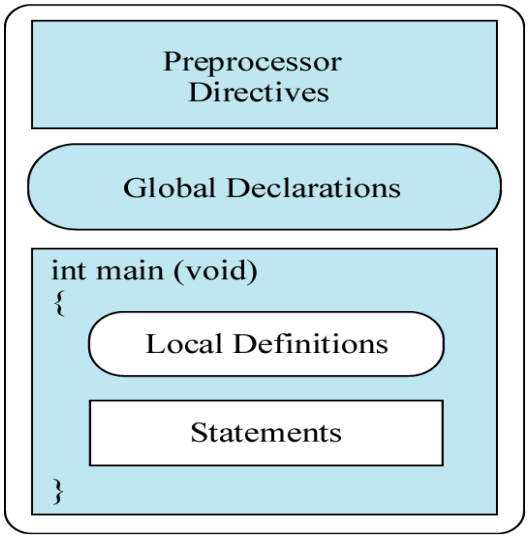
การพัฒนาโปรแกรมภาษาซีมีขั้นตอนดังรูปที่ 2.1 เริ่มต้นทำงานขั้นตอนแรกด้วยการเขียนโปรแกรมต้นฉบับด้วยภาษาซีลงบนเอดิเตอร์ (Editor) ที่เรียกว่า Source code แล้วทำการบันทึกไฟล์ เมื่อบันทึกเสร็จแล้วให้ทำการคอมไพล์โปรแกรมด้วยคอมไพเลอร์เพื่อแปลง Source Code ให้เป็น Object Code จากนั้นจะเรียกใช้ Linker เพื่อทำการรวม Object Code ที่ได้จากการคอมไพล์กับชุดคำสั่งเสริมที่อยู่ใน Library ของระบบถ้าไม่มีข้อผิดพลาด จะได้ไฟล์ที่มีส่วนขยายเป็น EXE (ในระบบปฏิบัติการ Windows) ซึ่งสามารถนำไปรันเพื่อใช้งานได้



รูปที่ 2.1 ขั้นตอนการพัฒนาโปรแกรมภาษาซี

2.3 การทำงานและโครงสร้างของภาษาซี

โปรแกรมภาษาซีมีองค์ประกอบที่สำคัญ 3 ส่วนคือไฟล์ส่วนเรียกใช้ฟังก์ชัน ส่วนกำหนด และส่วนของตัวโปรแกรม ดังรูปที่ 2.2



รูปที่ 2.2 โครงสร้างของภาษาซี

2.3.1 **ส่วนเรียกใช้ฟังก์ชัน** (Preprocessor Directives)

เป็นส่วนที่เรียกใช้ไฟล์ที่มีชื่อขยาย .h ซึ่งเป็นไฟล์เก็บฟังก์ชัน (คำสั่งของภาษาซีเป็นฟังก์ชันทั้งหมด) โดยจัดเป็นกลุ่มๆ ที่เหมือนกัน เช่น stdio.h เป็นไฟล์เก็บฟังก์ชันพื้นฐานทั่วไป ดังนั้นหากไม่มีการเรียกไฟล์นี้เข้ามาใช้งาน ก็จะไม่สามารถใช้คำสั่งหรือฟังก์ชันต่างๆ ที่เก็บไว้ใน ส่วนเรียกใช้ฟังก์ชัน (Preprocessor Directives) ได้ การเรียกใช้ไฟล์ header จะเรียกผ่าน #include ซึ่งเป็น preprocessor directives)

2.3.2 **ส่วนกำหนด** (Global Declarations)

เป็นส่วนที่ใช้ในการกำหนดตัวแปร ค่าคงที่ หรือฟังก์ชันใช้เอง โดยกำหนดชื่อและรายละเอียดการทำงานของฟังก์ชันไว้ทั้งหมดก่อนการเรียกใช้ฟังก์ชันในส่วนหลักของโปรแกรม ส่วนกำหนดวางไว้ตอนต้นเพื่อง่ายในการกลับขึ้นมาเปลี่ยนแปลงแก้ไข หากวางไว้ในตำแหน่งที่ใช้งานจะกระจัดกระจายลำบากในการค้นหามาแก้ไข ไม่ว่าตัวแปร ค่าคงที่ หรือฟังก์ชัน ส่วนประกอบส่วนนี้ในบางโปรแกรมไม่ต้องมีก็ได้ หากไม่มีการกำหนดตัวแปรร่วม หรือไม่มีการคำนวณ

2.3.3 **ส่วนของตัวโปรแกรม** (Main Function)

ภาษาซีจะกำหนดการทำงานทุกส่วนในรูปของฟังก์ชันทั้งสิ้น โดยมีฟังก์ชันหลักคือฟังก์ชัน Main ทุกโปรแกรมของภาษาซีต้องมีฟังก์ชันนี้เป็นหลักและอาจมีฟังก์ชันย่อยเพิ่มเติมตามความต้องการ วิธีการเขียนฟังก์ชัน main สามารถเขียนได้หลายรูปแบบ เช่น

- void main(void)
- int main(void)
- void main(int argc, char **argv)
- int main(int argc, char **argv)

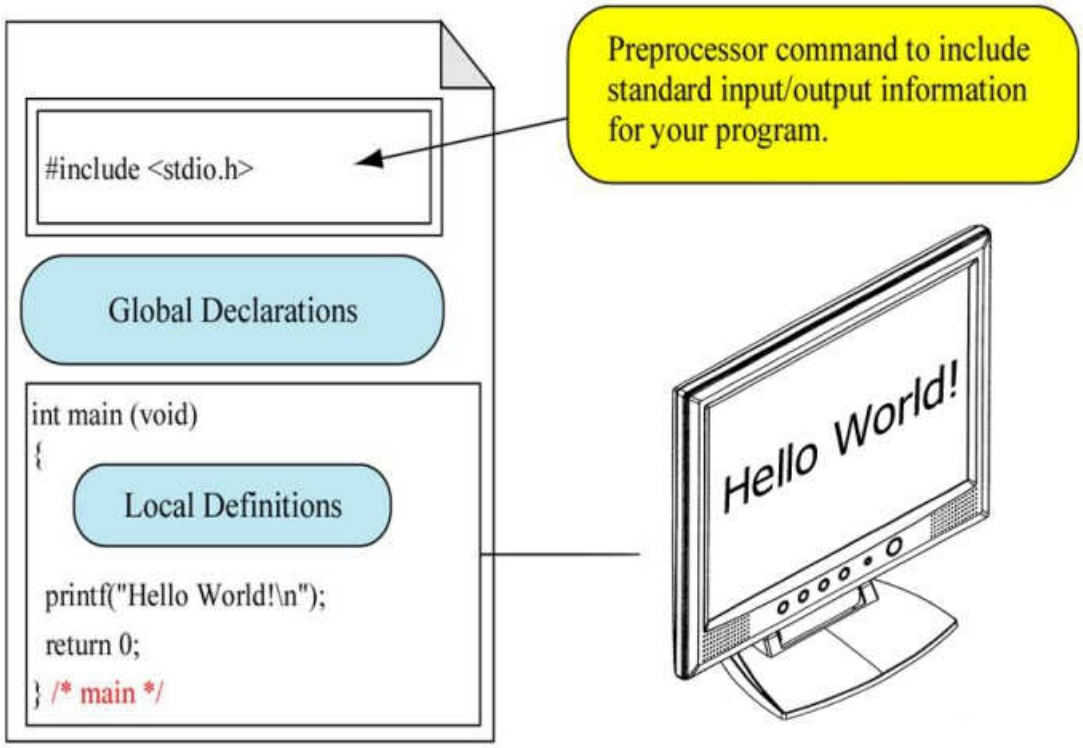
โดยที่ void หรือ int ที่อยู่หน้าคำว่า main หมายถึงประเภทข้อมูลที่ฟังก์ชัน main จะคืนค่า โดยที่ void หมายถึง ฟังก์ชัน main จะไม่มีการคืนค่า และ int หมายถึงฟังค์ชัน main จะคืนค่าจำนวนเต็ม ส่วน parameter ที่อยู่ในวงเล็บเป็นตัวบ่งบอกว่าฟังก์ชัน main จะสามารถรับค่าจากภายนอกโปรแกรมหรือไม่

ในภาษาซีสัญลักษณ์ * หน้าชื่อตัวแปรหมายถึงพอยน์เตอร์ซึ่งสามารถเขียนแทนด้วยเครื่องหมายวงเล็บก้ามปูเปิดปิดได้ [ ] ดังนั้นจึงสามารถเขียนฟังก์ชัน main เพิ่มได้อีกหลายแบบเช่น int main(int argc, char *argv[]) หรือ int main(int argc, char argv[][])

2.4 ไวยากรณ์ของภาษาซี และตัวอย่างโปรแกรม

ไวยากรณ์ของภาษาซีไม่ซับซ้อนกำกวม แต่ห้ามเขียนผิดแม้แต่นิดเดียว ภาษาซีมีการเขียนอยู่ในรูปแบบของบล็อก (Block) ที่เริ่มด้วยเครื่องหมายปีกกาเปิด “{“ และจบด้วยเครื่องหมาย ปีกกาปิด ”}” ทุกคำสั่งที่ไม่ได้ตามด้วยบล็อกจำเป็นต้องมีเครื่องหมายเซมิโคลอน “;” ปิดท้ายเสมอ ยกเว้นในส่วนของ Preprocessor directive และควรจำไว้ว่าชื่อของฟังก์ชันทั้งหมดในภาษาซีจะเป็นตัวเล็กทั้งหมด

รูปที่ 2.3 แสดงตัวอย่างการเขียนโปรแกรมภาษาซีที่ใช้ในการแสดงคำว่า “Hello World!” ออกทางหน้าจอ โดยจะมีส่วนของ Preprocessor Directive และ ส่วนของฟังก์ชัน main ในภาษาซีสามารถแทรกส่วนที่คอมไพเลอร์จะไม่สนใจในขั้นตอนคอมไพล์ได้ ส่วนนั้นเรียกว่า comment ซึ่งสามารถทำได้สองวิธี คือใช้เครื่องหมาย “//” ข้อความหลัง // จะไม่ถูกคอมไพเลอร์อ่าน หรือถ้าต้องการ comment แบบหลายบรรทัดให้ใช้ “/*” เป็นตัวเปิด แล้วปิดด้วย “*/” ข้อความทุกอย่างที่อยู่ระหว่าง /* และ */ จะไม่ถูกคอมไพเลอร์อ่าน



รูปที่ 2.3 ตัวอย่างการทำงานของภาษาซี

2.5 เครื่องมือที่ใช้ในการเรียน

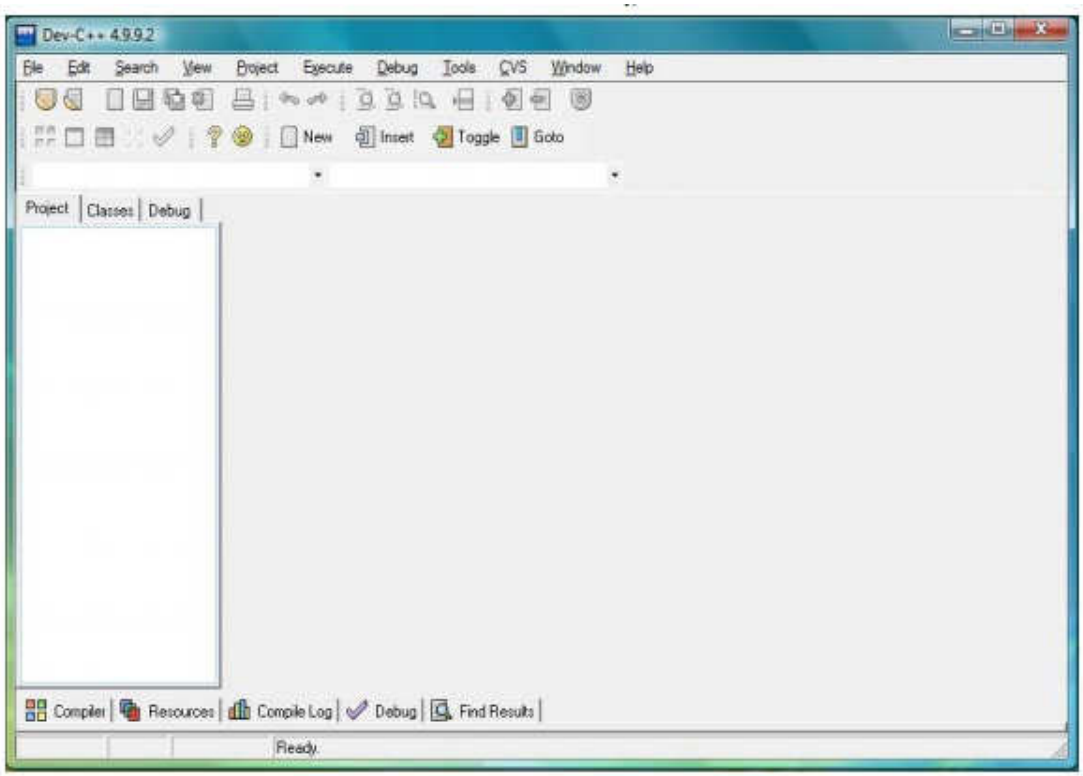
เครื่องมือที่ใช้ในการพัฒนาโปรแกรมด้วยภาษาซีในวิชานี้ จะใช้ตัว IDE ที่ชื่อ Dev-C++ เนื่องด้วยเป็น IDE ที่ฟรี และสามารถทำงานได้ครอบคลุมเนื้อหาทั้งหมดของวิชานี้ โดยจะใช้ Dev-C++ ในการเขียนโปรแกรมแบบ Console Application เท่านั้น

การเรียกใช้งานโปรแกรม Dev-C++

- ดับเบิ้ลคลิก shortcut Dev-C++ ดังรูปที่ 2.4 เพื่อเปิดโปรแกรม Dev-C++

รูปที่ 2.4 Shortcut ของ Dev-C++

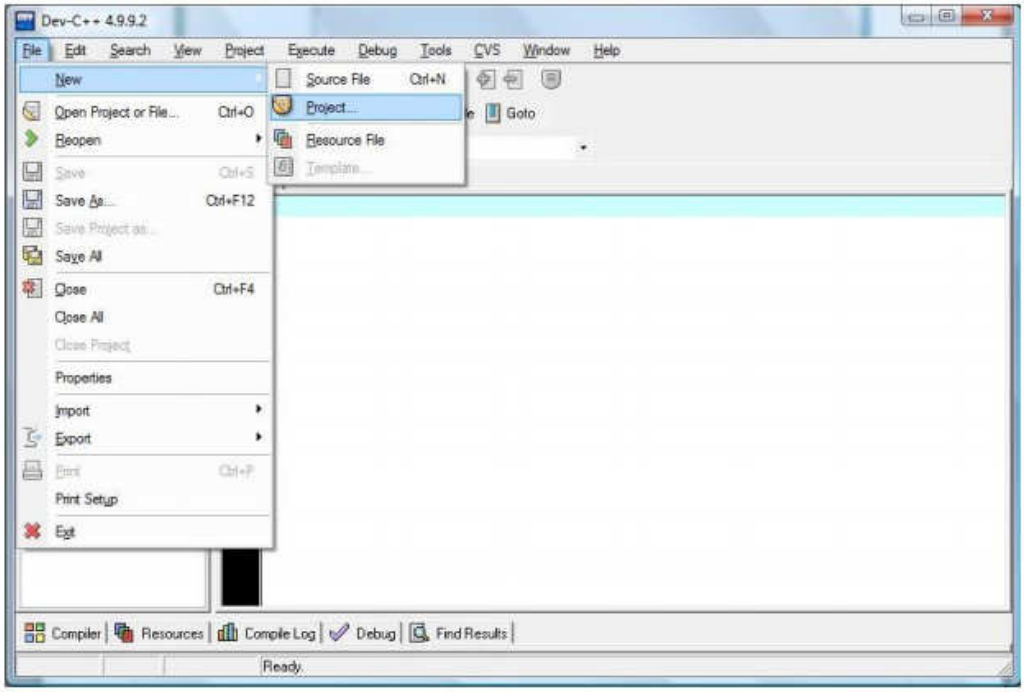
- หน้าต่างของโปรแกรม Dev-C++ จะปรากฏขึ้นดังรูปที่ 2.5



รูปที่ 2.5 หน้าต่างโปรแกรม Dev-C++

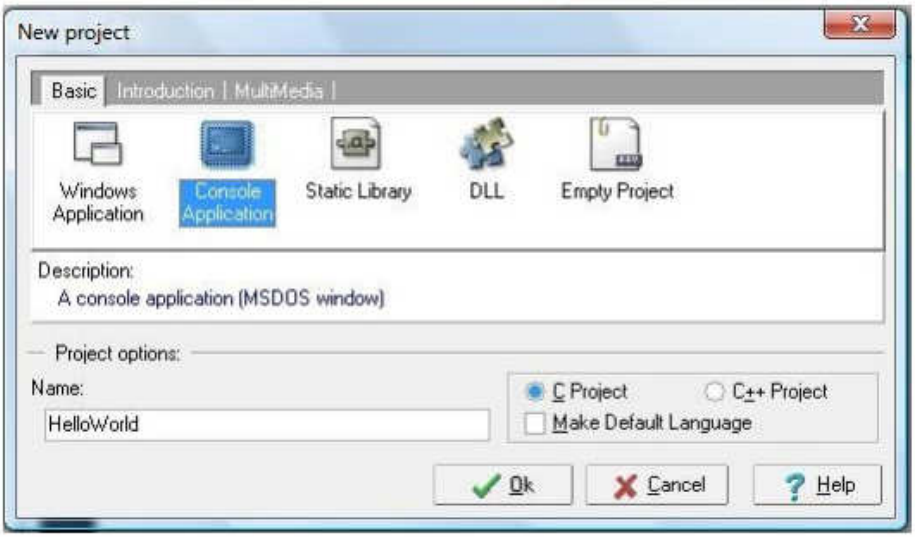
การสร้าง Project

- ทุกครั้งที่เริ่มพัฒนาโปรแกรมต้องสร้าง project ทำได้โดยการสั่งที่เมนู File → New→ Project ดังรูปที่ 2.6



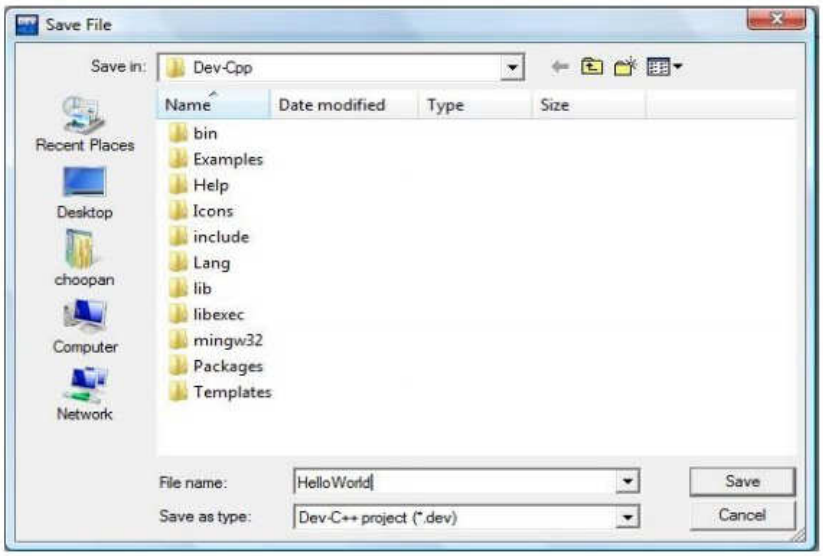
รูปที่ 2.6 การสร้าง project

- จากนั้นหน้าต่าง new project จะปรากฏขึ้นมา ให้เลือก console application, C project, และทำการใส่ชื่อ project จากนั้นกดปุ่ม Ok ดังรูปที่ 2.7



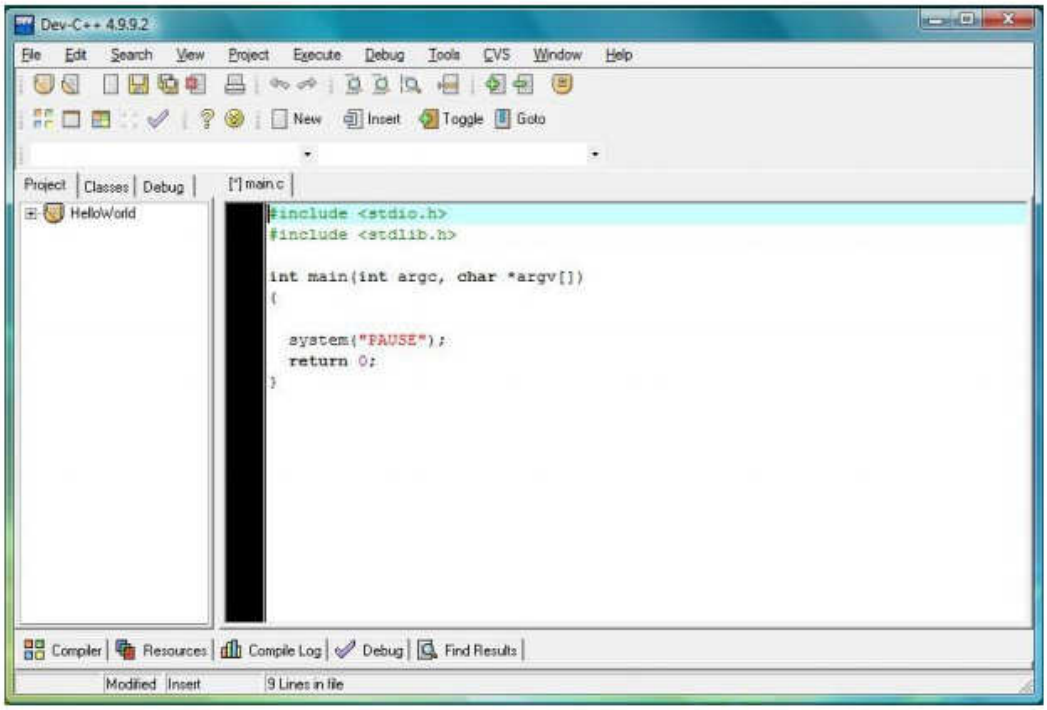
รูปที่ 2.7 หน้าต่าง new project

- เมื่อหน้าต่าง Save File ปรากฏขึ้นดังรูปที่ 2.8 ให้เลือกไปยังที่ที่ต้องการจะเก็บไฟล์โปรแกรม จากนั้นกดปุ่ม Save



รูปที่ 2.8 หน้าต่างสำหรับบันทึกที่เก็บแฟ้มข้อมูล

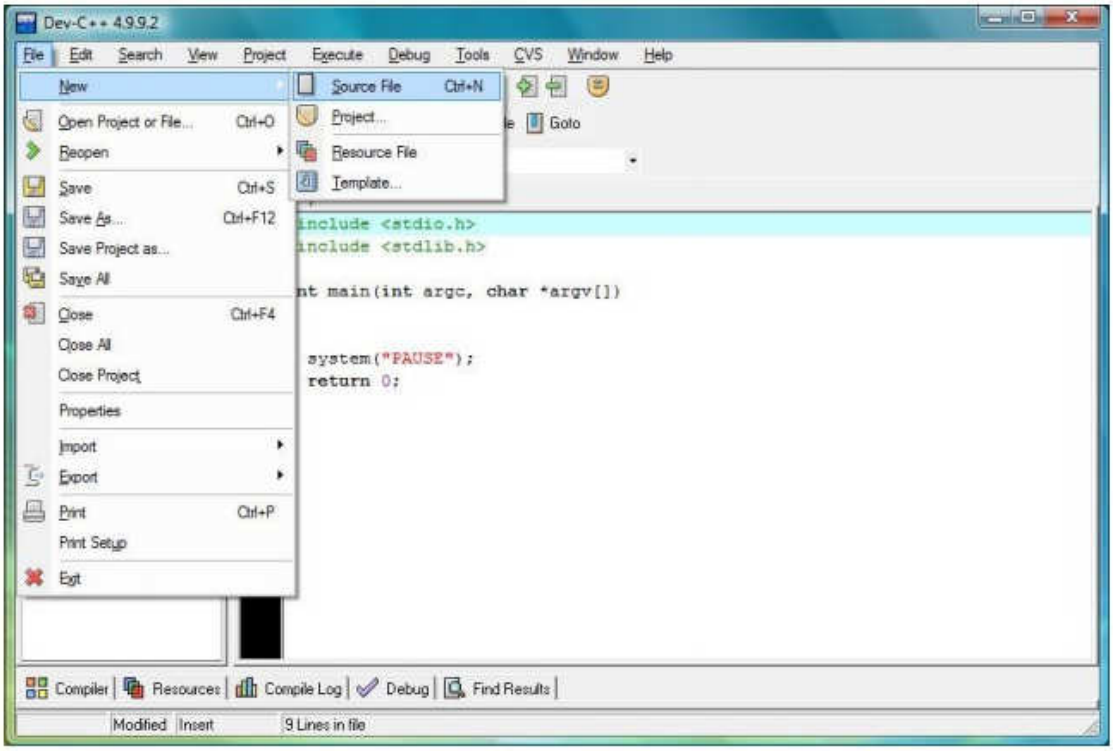
- Dev-C++ จะสร้างไฟล์ main.c เริ่มต้นมาให้และจะทำการ add เข้าไปใน project อัตโนมัติดังรูปที่ 2.9



รูปที่ 2.9 โปรแกรมต้นฉบับที่ถูกสร้างขึ้นเมื่อเปิด project

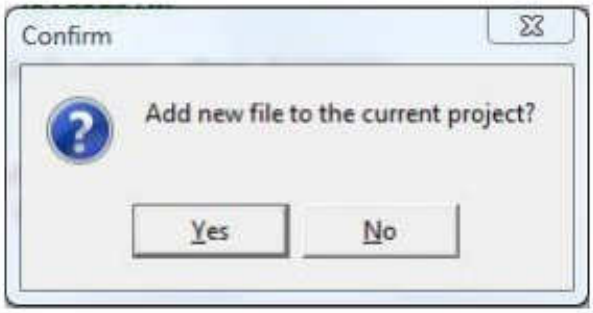
วิธีการเพิ่ม Source file ใน project

- ผู้พัฒนาโปรแกรมสามารถสร้างไฟล์ (Source file) โดยการสั่งที่เมนู File→ New→ Source file หรือ กด shortcut Ctrl+N ดังรูปที่ 2.10



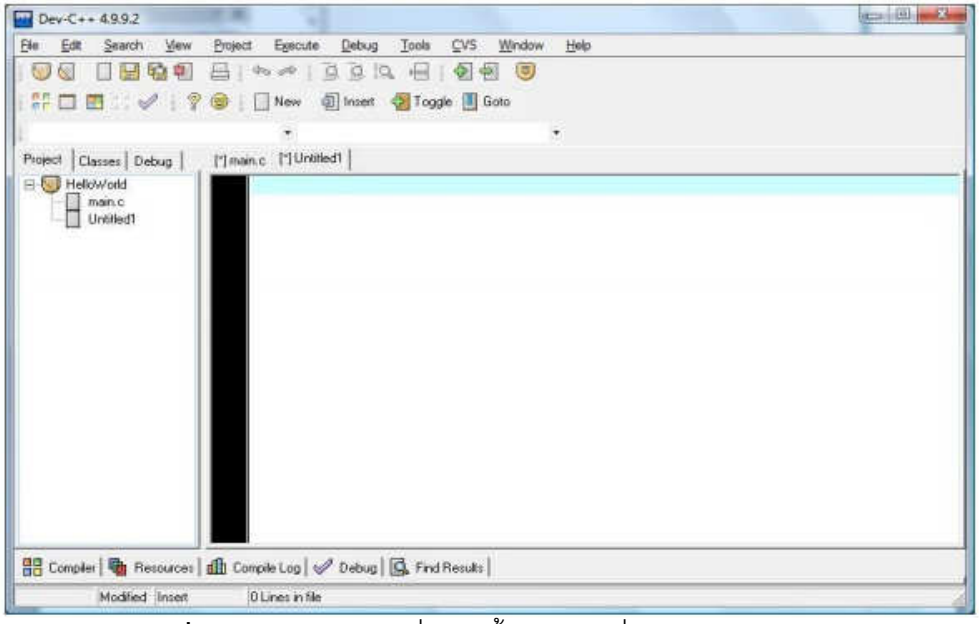
รูปที่ 2.10 การสร้าง source file ใหม่

- จากนั้นหน้าต่าง confirm จะปรากฏขึ้นดังรูปที่ 2.11 ถ้าผู้ใช้ต้องการเพิ่ม file เข้าไปใน project ให้ตอบ “Yes”



รูปที่ 2.11 หน้าต่างยืนยันการเพิ่ม source file เข้าไปใน project

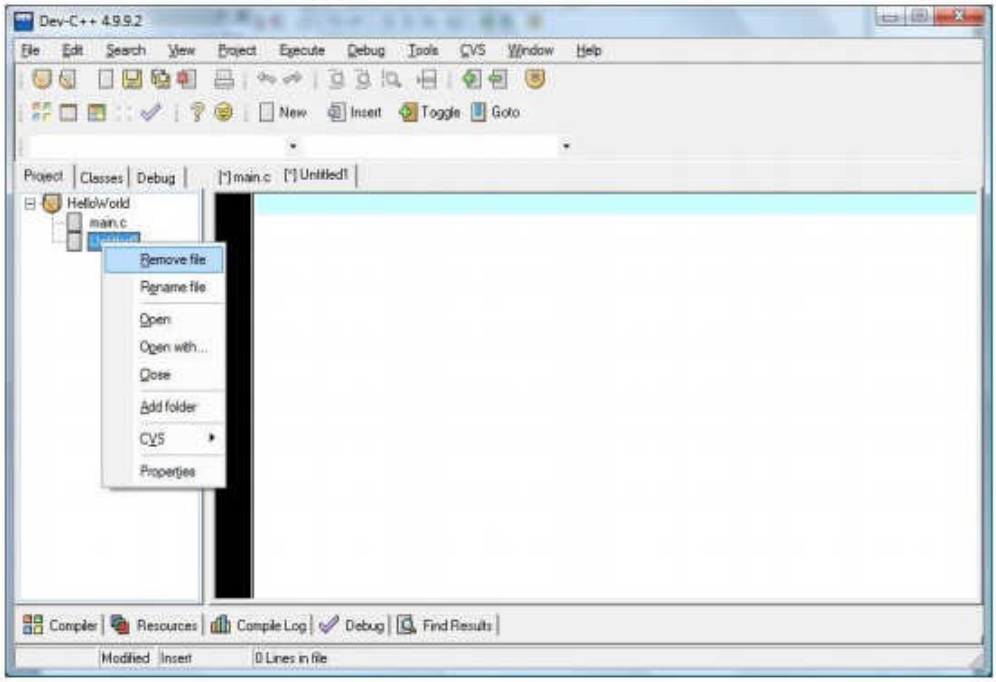
- ไฟล์ใหม่ที่ถูกเพิ่มเข้ามาจะอยู่ใน project สามารถดูได้ทางด้านซ้ายของโปรแกรม Dev-C++ ดังรูปที่ 2.12



รูปที่ 2.12 Source file ที่สร้างขึ้นใหม่ถูกเพิ่มเข้าไปใน project

การนำ Source file ออกจาก project

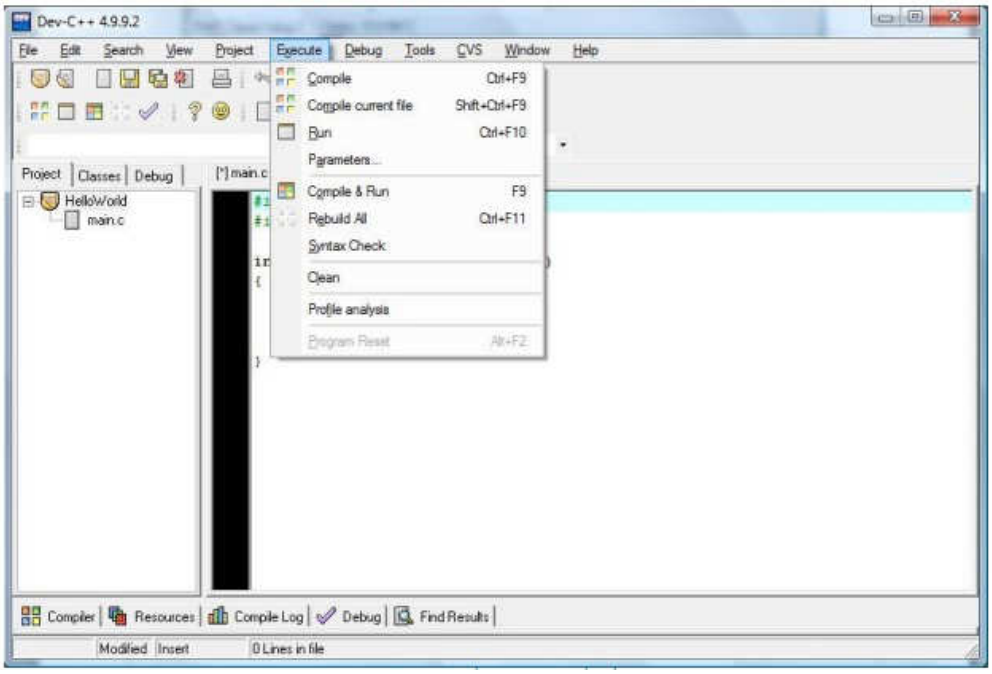
- กดคลิกขวาตรงชื่อ file ที่อยู่ทางด้านซ้ายของหน้าต่างแล้วเลือก remove file ดังรูปที่ 2.13



รูปที่ 2.13 การนำ source file ออกจาก project

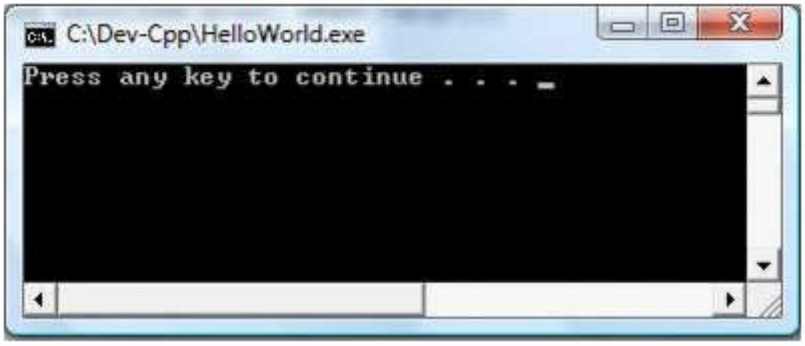
การ Compile และ Execute/Run โปรแกรม

- สามารถสั่งได้โดยเข้าเมนู Execute ดังรูปที่ 2.14 ข้างในเมนู Execute จะประกอบด้วยคำสั่งย่อย คำสั่งที่สำคัญได้แก่
    - Compile (Ctrl+F9) : ใช้ในการ Compile ไฟล์ที่อยู่ใน Project ทั้งหมด
    - Compile current file (Shift+Ctrl+F9) : Compile เฉพาะไฟล์ปัจจุบัน
    - Run (Ctrl+F10) : Run โปรแกรมที่ Compile ผ่านแล้ว
    - Compile & Run (F9) : Compile ไฟล์ใน project แล้ว Run โปรแกรมต่อ



รูปที่ 2.14 เมนูการ Compile และ Run โปรแกรม

- วิธีที่ง่ายที่สุดคือกด F9 เพื่อ Compile และ Run จะปรากฏหน้าจอดังรูปที่ 2.15



รูปที่ 2.15 หน้าจอการรันของโปรแกรม

บทที่ 3
ประเภทข้อมูลและการประกาศตัวแปร

การพัฒนาซอฟต์แวร์ด้วยภาษาซีนั้น จำเป็นต้องเข้าใจประเภทของข้อมูลที่และการประกาศใช้งานตัวแปร เนื่องด้วยประเภทข้อมูลแต่ละประเภทใช้ในการเก็บข้อมูล และใช้เนื้อที่ในหน่วยความจำที่ต่างกัน การจะเก็บข้อมูลในภาษาซีได้นั้นจำเป็นต้องมีการประกาศตัวแปรซึ่งก็คือการจองเนื้อที่ในหน่วยความจำขึ้นมาก่อน

3.1 ประเภทของข้อมูลในภาษาซี

ประเภทของข้อมูลในภาษาซีนั้นแบ่งออกเป็น 2 ประเภทใหญ่ๆ คือ ประเภทข้อมูลพื้นฐาน และประเภทข้อมูลแบบซับซ้อน (เช่น array, structure, และ union จะกล่าวถึงในบทที่ 9 และ บทที่ 13 ตามลำดับ) ซึ่งประเภทข้อมูลพื้นฐานสามารถแบ่งออกเป็น 7 ชนิด ที่ใช้สำหรับเก็บข้อมูล 3 รูปแบบ คือ

- จำนวนเต็ม ได้แก่ int, short, long
- จำนวนจริง ได้แก่ float, double, long double
- ตัวอักขระ ได้แก่ char

การเลือกใช้ประเภทข้อมูลในภาษาซีจำเป็นต้องเลือกใช้ให้ตรงกับความต้องการที่จะจัดเก็บข้อมูลประเภทนั้นๆ ถึงแม้จะเป็นการเก็บข้อมูลประเภทเดียวกันเช่น จำนวนเต็ม ผู้พัฒนาซอฟต์แวร์จำเป็นต้องเข้าใจขนาดหน่วยความจำที่ใช้ เนื่องจาก int, short, และ long สามารถเก็บข้อมูลได้ไม่เท่ากัน

3.1.1 จำนวนเต็ม

จำนวนเต็มคือจำนวนเต็มบวก ลบ หรือค่าศูนย์ จำนวนเต็มจะไม่มีจุดทศนิยม เช่น 1024, -50, 0 ในภาษาซีประเภทข้อมูลที่ใช้ในการเก็บจำนวนเต็มคือ int, short, และ long ข้อแตกต่างของประเภทข้อมูลทั้ง 3 ประเภทนี้คือ ขนาดของหน่วยความจำที่ใช้ในการจัดเก็บข้อมูลซึ่งจะส่งผลไปถึงขนาดของข้อมูลที่สามารถเก็บได้

ชนิดข้อมูลประเภท short, และ long จะใช้เนื้อที่ใช้การเก็บข้อมูล 2 ไบต์ และ 4 ไบต์ ตามลำดับ ในขณะที่ชนิดข้อมูลประเภท int จะขึ้นอยู่กับสถาปัตยกรรมของซีพียูและระบบปฏิบัติการซึ่งจะมีขนาด 2 ไบต์ในสถาปัตยกรรม 16-bits แต่ในปัจจุบันสถาปัตยกรรมของคอมพิวเตอร์ส่วนใหญ่เป็นแบบ 32-bits จึงส่งผลให้ชนิดข้อมูลประเภท int ใช้เนื้อที่ในการเก็บข้อมูล 4 ไบต์ ชนิดข้อมูลประเภท int เป็นประเภทของข้อมูลที่ซีพียูสามารถทำงานด้วยได้เร็วที่สุดเมื่อเทียบกับชนิดตัวแปรพื้นฐานประเภทอื่นๆ

ข้อมูลจำนวนเต็มที่จัดเก็บในหน่วยความจำ จะใช้บิตแรกสุดเป็นตัวเก็บเครื่องหมายบวก และลบ ซึ่งถ้าผู้พัฒนาซอฟต์แวร์ต้องการเก็บข้อมูลที่มีแต่จำนวนเต็มบวกเพียงอย่างเดียว แต่มีค่ามากกว่าชนิดข้อมูลนั้นจะเก็บได้ ผู้พัฒนาซอฟต์แวร์สามารถเพิ่มคำว่า unsigned เข้าไปหน้าประเภทข้อมูล เพื่อบ่งบอกว่า ข้อมูลที่ต้องการจะจัดเก็บมีแต่จำนวนเต็มบวก ซึ่งจะเป็นการขยายเนื้อที่ในการเก็บข้อมูลขึ้นมาอีก 1 บิต เนื่องจากไม่จำเป็นต้องใช้บิตแรกเพื่อเก็บเครื่องหมาย ตารางที่ 3-1 แสดงเนื้อที่ของหน่วยความจำ และค่าที่สามารถเก็บได้ในประเภทข้อมูลชนิดต่างๆ สำหรับข้อมูลที่เป็นจำนวนเต็ม

ตารางที่ 3.1 ขนาดหน่วยความจำ และค่าที่สามารถเก็บได้ในข้อมูลประเภทจำนวนเต็มแต่ละประเภท

ชนิดของข้อมูล	จำนวนบิตที่ใช้	ช่วงของข้อมูล	การกำหนดชนิดข้อมูล
เลขจำนวนเต็ม (Integer)	32	-2147483648 ถึง 2147843649	int
เลขจำนวนเต็มไม่คิดเครื่องหมาย (unsigned integer)	32	0 ถึง 4294967296	unsigned int
เลขจำนวนเต็มสั้น (short integer)	16	-32768 ถึง 32767	short int หรือ short
เลขจำนวนเต็มสั้น ไม่คิดเครื่องหมาย (unsigned short integer)	16	0 ถึง 65535	unsigned short int หรือ unsigned short
เลขจำนวนเต็มยาว (long integer)	32	-2147483648 ถึง 2147843649	long int หรือ long
เลขจำนวนเต็มยาว ไม่คิดเครื่องหมาย (unsigned long integer)	32	0 ถึง 4294967296	unsigned long int หรือ unsigned long

3.1.2 จำนวนจริง

จำนวนจริงคือจำนวนบวก ลบ หรือค่าศูนย์ ซึ่งสามารถมีจุดทศนิยมได้ เช่น 1024.05, -50.55, 0 ในภาษาซีประเภทข้อมูลที่ใช้ในการเก็บจำนวนจริงคือ float, double, และ long double ข้อแตกต่างของประเภทข้อมูลทั้ง 3 ประเภทนี้คือ ขนาดของหน่วยความจำที่ใช้ในการจัดเก็บข้อมูลซึ่งจะส่งผลไปถึงขนาดของข้อมูลที่สามารถเก็บได้

ตารางที่ 3.2 แสดงขนาดของหน่วยความจำและค่าของข้อมูลที่เก็บได้ในแต่ละชนิดของข้อมูล ซึ่งชนิดของข้อมูล float, double และ long double จะใช้ขนาดของหน่วยความจำในการจัดเก็บข้อมูล 32, 64 และ 80 บิตตามลำดับ

ตารางที่ 3.2 ขนาดหน่วยความจำ และ ค่าที่สามารถเก็บได้ในข้อมูลประเภทจำนวนจริงแต่ละประเภท

ชนิดของข้อมูล	จำนวนบิตที่ใช้	ช่วงของข้อมูล	การกำหนดชนิดข้อมูล
เลขจำนวนจริง (floating point)	32	3.4×10^{-38} ถึง 3.4×10^{38}	float
เลขจำนวนจริงละเอียด 2 เท่า (double precision floating point)	64	1.7×10^{-308} ถึง 1.7×10^{308}	double
เลขจำนวนจริงยาวละเอียด 2 เท่า (long double precision floating point)	80	3.4×10^{-4932} ถึง 1.1×10^{4932}	long double

3.1.3 ตัวอักขระ

ถ้าต้องการจัดเก็บข้อมูลในรูปแบบตัวอักษร หรืออักขระต่างๆในภาษาซี ควรเลือกใช้ชนิดข้อมูลประเภท char ชนิดข้อมูลประเภทนี้จะใช้เนื้อที่ในหน่วยความจำ 8 bits เนื่องด้วยหน่วยความจำหลักและการเก็บข้อมูลของคอมพิวเตอร์อยู่ในรูปบิต หรือเลขฐานสองที่มีค่า “0” และ “1” ดังนั้นการเก็บตัวอักษรในระบบคอมพิวเตอร์นั้นจำเป็นต้องมีรหัสที่ใช้ในการแปลงค่าตัวอักขระเป็นเลขฐานสองรหัสที่ใช้ในการจัดเก็บข้อมูลตัวอักขระลงในคอมพิวเตอร์นั้นใช้ รหัสแอสกี้ (ASCII : American Standard Code for Information Interchange) รูปที่ 3.1 แสดงตารางรหัสแอสกี้และเลขฐาน 2 สำหรับตัวอักขระต่างๆ อีกทั้งชนิดข้อมูลประเภท char สามารถเก็บรหัสควบคุมพิเศษได้อีกด้วยดังตารางที่ 3.3

USASCII code chart

Bits $b_7 b_6 b_5$ →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ Row / Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

รูปที่ 3.1 ตารางรหัสแอสกี้ (ASCII Table)

ตารางที่ 3.3 รหัสควบคุมพิเศษในภาษาซี

สัญลักษณ์	ความหมาย
\b	เลื่อนเคอร์เซอร์ไปทางซ้าย
\f	ขึ้นหน้าใหม่
\n	ขึ้นบรรทัดใหม่
\r	เลื่อนเคอร์เซอร์ไปทางซ้ายสุด
\t	Tab ในแนวนอน
\”	แสดงเครื่องหมาย ”
\’	แสดงเครื่องหมาย ’
\\	แสดงเครื่องหมาย \
\a หรือ \007	เสียงบีป

3.2 การประกาศตัวแปรในภาษาซี

ตัวแปรในภาษาซีใช้ในการเก็บข้อมูลชั่วคราวระหว่างการทำงานของโปรแกรม โดยที่ตัวแปรทุกตัวจะต้องมีการประกาศชื่อและประเภทของข้อมูลที่ต้องการเก็บ และจำเป็นต้องประกาศก่อนที่จะนำมาใช้งานทุกครั้ง การนำค่าในตัวแปรมาใช้งานจะไม่ทำให้ค่านั้นหายไป แต่สามารถนำข้อมูลใหม่มาเก็บทับได้ หลังจากการประกาศใช้งานตัวแปรค่าที่เก็บในตัวแปรจะเรียกว่า ค่าขยะ ซึ่งจะเป็นค่าที่อยู่ในหน่วยความจำเก่าและจะมีค่าไม่เหมือนกันในแต่ละครั้งที่สั่งโปรแกรมทำงาน จึงควรมีการใส่ค่าที่ต้องการให้ถูกต้องก่อนนำตัวแปรนั้นไปใช้งาน

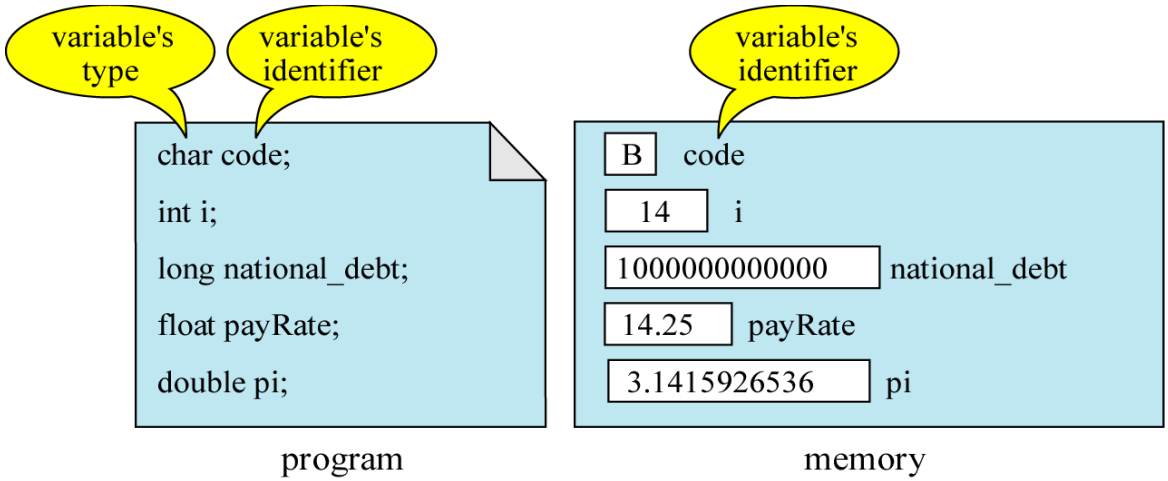
การประกาศตัวแปรในภาษาซีจะอยู่ในรูปแบบที่ตายตัวคือ ประเภทของตัวแปร ตามด้วยชื่อของตัวแปร และสามารถกำหนดค่าเริ่มต้นให้กับตัวแปรได้ซึ่งจะไม่กำหนดก็ได้ รูปที่ 3.2 มีการประกาศตัวแปรชื่อ grade ที่จะใช้เก็บข้อมูลประเภทตัวอักขระซึ่งกำหนดค่าเริ่มต้นเป็นตัวอักษร A, ประกาศตัวแปรชื่อ PI เก็บข้อมูลประเภทจำนวนจริงมีค่าเริ่มต้นเป็น 3.14159, และตัวแปรชื่อ salary ที่จะเก็บข้อมูลประเภทจำนวนเต็มซึ่งไม่กำหนดค่าเริ่มต้น

```
char  grade = 'A';
float PI = 3.14159;
int   salary;
```

รูปที่ 3.2 การประกาศตัวแปรในภาษาซี

```
char  grade = 'A', fall = 'F' ;
float x1 = 1.2, x2 = 2.4, x3;
int   y1, y2;
```

รูปที่ 3.3 การประกาศตัวแปรหลายตัวที่มีการเก็บข้อมูลประเภทเดียวกัน



รูปที่ 3.4 การประกาศตัวแปรในภาษาซีและการเก็บข้อมูลในหน่วยความจำ

ถ้าต้องการจะประกาศตัวแปรประเภทเดียวกันหลายตัวแปรสามารถทำได้ ดังรูปที่ 3.3 ตัวแปร grade และ fall ใช้เก็บข้อมูลประเภทตัวอักขระโดยมีการกำหนดค่าเริ่มต้นคือ 'A' และ 'F' ตามลำดับ ตัวแปร x1, x2, x3 เก็บข้อมูลประเภทจำนวนจริงโดยที่ x1 มีการกำหนดค่าเริ่มต้นคือ 1.2 และ x2 มีการกำหนดค่าเริ่มต้นคือ 2.4 ในขณะที่ y1 และ y2 เก็บข้อมูลประเภทจำนวนเต็มที่ไม่มีการกำหนดค่าเริ่มต้น และรูปที่ 3.4 แสดงการประกาศตัวแปรในภาษาซีและข้อมูลที่ถูกจัดเก็บในหน่วยความจำของเครื่องคอมพิวเตอร์ ของประเภท char, int, long, float และ double ตามลำดับ

3.3 กฎการตั้งชื่อตัวแปร

การตั้งชื่อตัวแปรในภาษาซี มีกฎระเบียบต่างๆดังนี้

- ชื่อตัวแปรจะต้องประกอบด้วย ตัวอักษร ตัวเลข และ ตัวอักขระ $ และ _ เท่านั้น
- ชื่อตัวแปรห้ามขึ้นต้นด้วยตัวเลข
- ชื่อตัวแปรห้ามยาวเกิน 32 ตัวอักษรในคอมไพเลอร์เก่าๆ แต่คอมไพเลอร์ใหม่ๆ สามารถรองรับได้ในความยาวไม่จำกัด
- ตัวอักษรตัวใหญ่ไม่เท่ากับตัวอักษรตัวเล็ก
- ชื่อตัวแปรห้ามซ้ำกับคำสงวนในภาษาซี ดังรูปที่ 3.5
- ชื่อตัวแปรห้ามซ้ำกับชื่ออื่นๆในโปรแกรม เช่น ชื่อฟังก์ชัน

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

รูปที่ 3.5 คำสงวนในภาษาซี

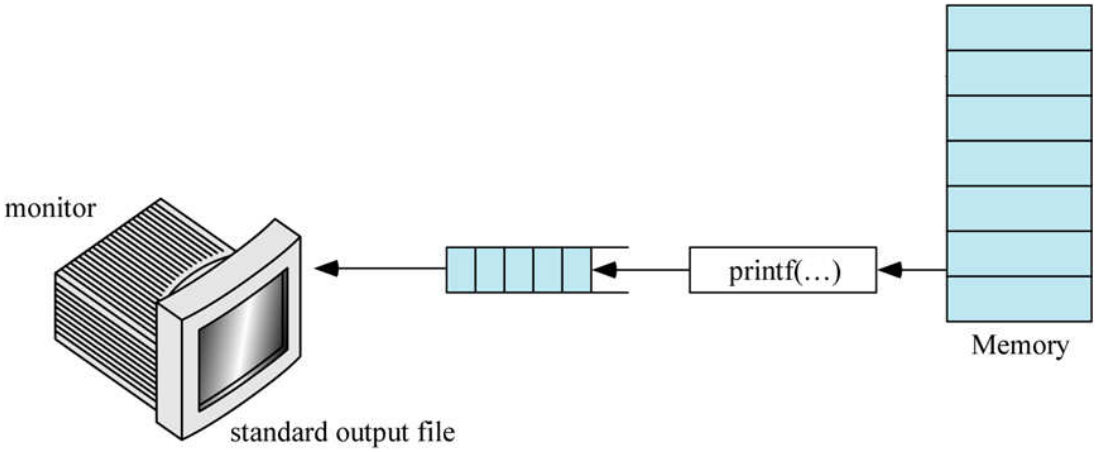
ตัวอย่างการตั้งชื่อตัวแปรที่ผิดเช่น

- 12monkey เนื่องจากขึ้นต้นด้วยตัวเลข
- I..Love..U เนื่องจากมีตัวขระ "." ที่ภาษาซีไม่ยอมให้ใช้ในการตั้งชื่อตัวแปร
- ohOh! เนื่องจากมีตัวขระ "!" ที่ภาษาซีไม่ยอมให้ใช้ในการตั้งชื่อตัวแปร

หลักการตั้งชื่อตัวแปรในภาษาซีมีมากมายหลายแบบ แต่ส่วนใหญ่จะมีธรรมเนียมคือ หลีกเลี่ยงการตั้งชื่อตัวแปรที่ขึ้นต้นด้วยตัวอักขระ _ และ $ เนื่องจากฟังก์ชันภายในของภาษาซี จะใช้ตัวอักขระสองตัวนี้ในการตั้งชื่อ อาจจะทำให้ชื่อตัวแปรที่ตั้งไปซ้ำกับชื่อฟังก์ชันภายในของภาษาซี และควรตั้งชื่อตัวแปรให้สื่อกับค่าที่จะจัดเก็บในตัวแปรเช่น ตั้งชื่อตัวแปรว่า score เพื่อใช้ในการเก็บค่าคะแนนสอบ ทั้งนี้เนื่องจากเมื่อพัฒนาซอฟต์แวร์ที่ใหญ่อาจทำให้ผู้พัฒนาลืมว่าตัวแปรนี้ใช้เก็บค่าอะไร

บทที่ 4
การแสดงผลในภาษาซี

เมื่อต้องการแสดงค่าของตัวแปรหรือพิมพ์ข้อความใดๆ ด้วยภาษาซีจะต้องใช้ฟังก์ชัน printf ซึ่งจะทำให้ข้อความภายในเครื่องหมายคำพูด (“……”) แสดงผลบนจอภาพ หลักการทำงานของฟังก์ชัน printf นั้นจะเป็นการดึงข้อมูลที่เก็บในหน่วยความจำหลักมาแสดงบนจอภาพดังรูปที่ 4.1



รูปที่ 4.1 การทำงานของฟังก์ชัน printf

ฟังก์ชัน printf มีรูปแบบการใช้งานดังนี้

```
printf (“ control  string ” , variable  list);
```

- control string อาจจะเป็นตัวอักษร ข้อความ หรือตัวกำหนดประเภทของการแสดงผลตัวแปรที่ต้องการแสดงออกทางจอภาพ
- variable list ถ้า control string เป็นตัวกำหนดประเภทของการแสดงผลตัวแปรจำเป็นต้องใส่ชื่อตัวแปรในส่วนนี้ ถ้า control string เป็นข้อความปกติส่วนของ variable list ไม่จำเป็นต้องมี

ตัวอย่างการแสดงผลข้อความคำว่า Hello World ออกทางหน้าจอด้วยคำสั่ง printf

```
printf (“Hello World”);
```

ถ้าต้องการแสดงข้อมูลที่เก็บในตัวแปรออกทางหน้าจอนั้น ภาษาซีจะใช้ชุดอักขระพิเศษที่เรียกว่า Format Code ซึ่งจะเป็นตัวกำหนดให้ข้อมูลที่อยู่ในตำแหน่งหลังเครื่องหมาย “,” แสดงออกมา โดยจะมีความหมายดังตารางที่ 4.1

ตารางที่ 4.1 รหัส Format Code สำหรับฟังก์ชัน printf

รหัสรูปแบบ	ความหมาย
%c	ใช้กับตัวแปรที่เก็บค่าเป็นตัวอักษรเพียงตัวเดียว
%s	ใช้กับตัวแปรที่เก็บค่าเป็นข้อความที่เก็บในตัวแปรชุด
%d	ใช้กับตัวแปรที่เก็บค่าเป็นเลขจำนวนเต็ม
%u	ใช้กับตัวแปรที่เก็บค่าเป็นเลขจำนวนเต็มบวก
%f	ใช้กับตัวแปรที่เก็บค่าที่เป็นเลขทศนิยม
%e	ใช้กับตัวแปรที่เก็บค่าที่เป็นเลขทศนิยมในรูป e ยกกำลัง
%x	ใช้กับตัวแปรที่เก็บค่าที่เป็นเลขฐานสิบหก
%o	ใช้กับตัวแปรที่เก็บค่าที่เป็นเลขฐานแปด
%p	ใช้กับตัวแปรที่เก็บค่าที่เป็นตัวชี้ตำแหน่ง (pointer)

การแทนค่าตัวแปรจาก format code จะแทนค่าตามลำดับของตัวแปรที่กำหนดไว้ในส่วนของ variable list เช่น printf(“%d %d”, A , B); ข้อความที่แสดงบนหน้าจอจะแทนที่ตำแหน่ง %d ตัวแรกด้วยค่าที่เก็บในตัวแปร A ในรูปเลขฐาน 10 และ ตำแหน่ง %d ตัวที่ 2 ด้วยค่าที่เก็บในตัวแปร B ในรูปของเลขฐาน 10

ตัวอย่างโปรแกรมที่ 4.1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(int argc,  char **argv) {
4       int A = 5;
5      float B = 10.05 ;
6      printf(“%f     %d”, B, A);
7      system(“PAUSE”);
8  }
```

ผลการรัน

```
10.05      5
```

จากโปรแกรมตัวอย่าง 4.1 ในบรรทัดที่ 4 และ 5 มีการประกาศตัวแปร A ที่เก็บค่าจำนวนเต็มมีการกำหนดค่าเริ่มต้นคือ 5 และประกาศตัวแปร B ที่เก็บค่าจำนวนจริงมีการกำหนดค่าเริ่มต้นคือ 10.05 ตามลำดับ ในบรรทัดที่ 6 เป็นการเรียกฟังก์ชัน printf เพื่อแสดงผลออกทางจอภาพ โดย control list ประกอบไปด้วย %f และ %d ซึ่งจะเป็นการดึงค่าในตัวแปรที่อยู่ในส่วนของ variable list ออกมาแสดงตามลำดับ ดังนั้นค่าในตัวแปร B จะถูกแสดงก่อนตามด้วยค่าในตัวแปร A

ตัวอย่างโปรแกรมที่ 4.2

1	#include <stdio.h>	**ผลการรัน**
2	#include <stdlib.h>	
3	int main(int argc, char **argv) {	A-ANT B-Bird
4	char X = 'A', Y = 'B';	
5	printf("%c-ANT \n%c-Bird", X, Y);	
6	system("PAUSE");	
7	}	

ตัวอย่างโปรแกรมที่ 4.2 แสดงการใช้งานของ Format code ร่วมกันกับข้อความปกติและตัวควบคุมพิเศษ '\n' สำหรับการขึ้นบรรทัดใหม่ ซึ่ง %c ตัวแรกในฟังก์ชัน printf จะถูกแทนที่ด้วยค่า ในตัวแปร X ในที่นี้คือ 'A' ที่ได้ถูกประกาศในบรรทัดที่ 4 และ %c ตัวที่ 2 จะถูกแทนที่ตัวค่าในตัวแปร Y ในที่นี้คือ 'B'

ฟังก์ชัน printf ยังมีความสามารถในการจัดตำแหน่งข้อความในการแสดงผลผ่านจอภาพได้อีก ซึ่งสามารถแบ่งออกเป็น 3 ประเภทใหญ่ๆ คือ

- ตัวเลขบวกตามหลัง % เช่น %10d หมายถึงการเว้นช่องว่างไว้ 10 ช่องแล้วใส่ตัวเลขจากตำแหน่งขวาไปยังตำแหน่งซ้าย ตามตัวอย่าง

คำสั่ง	printf("=%5d=", 12);								
ผลการรัน	=				1	2	=		

- ตัวเลขลบตามหลัง % เช่น %-10d หมายถึงการเว้นช่องว่างไว้ 10 ช่องแล้วใส่ตัวเลขจากตำแหน่งซ้ายไปยังตำแหน่งขวา ตามตัวอย่าง

คำสั่ง	printf("=%-5d=", 12);								
ผลการรัน	=	1	2				=		

- จุดทศนิยม ซึ่งจะใช้กับ %f เช่น %.2f หมายถึงการแสดงค่าของตัวแปรให้อยู่รูปของจุดทศนิยม 2 ตำแหน่ง ซึ่งสามารถใช้ควบคู่ไปกับค่าบวกและลบได้ด้วย ตามตัวอย่าง

คำสั่ง	printf("=%.2f=", 10.1234);								
ผลการรัน	=	1	0	.	1	2	=		

คำสั่ง	printf("=%7.2f=", 10.1234);								
ผลการรัน	=			1	0	.	1	2	=

คำสั่ง	printf("=%-7.2f=", 10.1234);								
ผลการรัน	=	1	0	.	1	2			=

ตัวอย่างโปรแกรมที่ 4.3

```
1  int main(int argc,  char **argv) {
2        int    a;
3        float  b;
4        a = 25;
5        b = 15.281;
6        printf("A = %d\n", a);
7        printf("B = %f\n", b);
8        printf("A = %8d\n", a);
9        printf("B = %8.2f\n", b);
10       system("PAUSE");
11 }
```

ผลการรัน

```
A = 25
B = 15.281000
A =       25
B =    15.28
```

จากตัวอย่างโปรแกรมที่ 4.3 ค่าของตัวแปร a และ b จะถูกพิมพ์โดยใช้ตัวกำหนดชนิดข้อมูล %d และ %f ในบรรทัดที่ 6 และ บรรทัดที่ 7 ตามลำดับ ฟังก์ชัน printf( ) ในบรรทัดที่ 8 จะพิมพ์ค่าของตัวแปร a โดยมีความยาว 8 ตำแหน่ง ตำแหน่งว่างหน้าตัวเลขจะแทนด้วยช่องว่าง ในบรรทัดที่ 9 จะพิมพ์ค่าของตัวแปร b โดยมีความยาวทั้งหมด 8 ตำแหน่ง ซึ่งจะมีจำนวนทศนิยม 2 ตำแหน่ง ซึ่งจะเห็นว่าผลลัพธ์ของคำสั่ง printf ในบรรทัดที่ 8 และ บรรทัดที่ 9 จะได้ตัวเลขชิดขวาในตำแหน่งเดียวกัน

ตัวอย่างโปรแกรมที่ 4.4

1	int main(int argc, char **argv) {	**ผลการรัน**
2	char tab;	
3	tab = '\t';	1234567890
4	printf("1234567890\n");	Exam
5	printf("%c Exam\n", tab);	
6	system("PAUSE");	
7	}	

ตัวอย่างโปรแกรมที่ 4.4 ในบรรทัดที่ 4 จะเป็นการพิมพ์ข้อความปกติแล้วขึ้นบรรทัดใหม่จากตัวควบคุมพิเศษ '\n' จากนั้นในบรรทัดที่ 5 จะเป็นการแสดงค่าที่ผ่าน %c ซึ่งคือแสดงค่าในตัวแปร tab ในที่นี้มีค่าเท่ากับ \t นั่นคือเรากำหนดให้ตัวแปร tab มีการเลื่อน (Tab) ไปในแนวนอน การนั้นพิมพ์ช่องว่างและคำว่า Exam

ตัวอย่างโปรแกรมที่ 4.5

1	int main(int argc, char **argv) {	**ผลการรัน**
2	char ch = 'A';	
3	printf("ASCII value of A = %d\n", ch);	ASCII value of A = 65
4	system("PAUSE");	
5	}	

ตัวอย่างโปรแกรมที่ 4.5 จะแสดงค่าของรหัสแอสกีของตัวอักษรโดยจะต้องพิมพ์ด้วย %d ในบรรทัดที่ 3 ทำให้ผลการรันออกมาเป็นค่า 65 ซึ่งก็คือรหัสแอสกีของตัวอักษร A

ตัวอย่างโปรแกรมที่ 4.6

1	int main(int argc, char **argv) {	**ผลการรัน**
2	printf("12345678901234567890\n");	12345678901234567890
3	printf("=%s=\n", "C language");	=C language=
4	printf("=%-15s=\n", "C language");	=C language =
5	printf("=%15s=\n", "C language");	= C language=
6	system("PAUSE");	
7	}	

ตัวอย่างโปรแกรมที่ 4.6 เป็นการพิมพ์ข้อความหรือสตริงก์ซึ่งจะต้องใช้ตัวกำหนดชนิดข้อมูล %s ตัวกำหนดชนิดข้อมูล %s สามารถใช้ร่วมกับตัวเลขบวกและลบได้ บรรทัดที่ 3 เป็นการพิมพ์ข้อความออกทางหน้าจอแบบธรรมดา บรรทัดที่ 4 มีการกำหนดให้ชิดขอบซ้ายโดยจะเห็นว่ามีช่องว่างด้านหลังข้อความก่อนจะแสดงตัวอักขระ “=” ส่วนบรรทัดที่ 5 เป็นการกำหนดให้ข้อความที่แสดงชิดขอบขวา

บทที่ 5
คณิตศาสตร์ในภาษาซี

ภาษาคอมพิวเตอร์แทบทุกภาษา จะต้องมีเครื่องหมายคณิตศาสตร์เพื่อใช้ในการคำนวณ ใน บทนี้จะได้กล่าวถึงการใช้เครื่องหมายคณิตศาสตร์ต่างๆ และการเขียนนิพจน์คณิตศาสตร์ (Arithmetic expression) หรือนิพจน์ตัวเลข (Numeric expression)

5.1 เครื่องหมายคณิตศาสตร์พื้นฐาน

ภาษาซีมีเครื่องหมายคณิตศาสตร์พื้นฐานซึ่งจะแบ่งออกเป็น 2 ประเภท

- เครื่องหมายทางคณิตศาสตร์ที่ใช้กับจำนวนเต็ม และ
- เครื่องหมายทางคณิตศาสตร์ที่ใช้กับจำนวนจริง

5.1.1 เครื่องหมายทางคณิตศาสตร์ที่ใช้กับจำนวนเต็ม

เครื่องหมายทางคณิตศาสตร์ที่ใช้กับจำนวนเต็มเมื่อกระทำกับจำนวนเต็มจะให้ผลลัพธ์ที่เป็นจำนวนเต็ม เครื่องหมายทั้งหมดแสดงในตารางที่ 5.1 เมื่อนำข้อมูลจำนวนเต็ม 2 ตัว มาหารกันโดยใช้เครื่องหมาย “/” ผลลัพธ์ที่ได้จะเป็นตัวเลขจำนวนเต็มโดยส่วนที่เป็นเศษจะถูกตัดทิ้งไป เช่น 10 / 3 จะได้ 3 สำหรับการหารที่คิดเฉพาะเศษโดยใช้เครื่องหมาย % ผลลัพธ์จากการหารจะได้เศษที่เกิดจากการหาร เช่น 10 % 3 จะมีค่าเป็น 1

ตารางที่ 5.1 เครื่องหมายคณิตศาสตร์พื้นฐานสำหรับจำนวนเต็ม

เครื่องหมาย	ความหมาย
–	ลบ
+	บวก
*	คูณ
/	หาร(คิดเฉพาะเลขจำนวนเต็ม)
%	หาร (คิดเฉพาะเศษ)

5.1.2 เครื่องหมายทางคณิตศาสตร์ที่ใช้กับจำนวนจริง

เครื่องหมายทางคณิตศาสตร์ที่ใช้กับจำนวนจริงจะเหมือนกับที่ใช้กับจำนวนจริงเกือบทุกอย่าง ยกเว้นเครื่องหมายหาร “/” ผลลัพธ์ที่ได้จะเป็นจำนวนเต็ม เช่น 10 / 3 จะได้ 3.3333 และจะไม่สามารถใช้เครื่องหมายหารที่เอาแต่เศษได้ (%) เครื่องหมายคณิตศาสตร์พื้นฐานสำหรับจำนวนจริงแสดงในตารางที่ 5.2

ตารางที่ 5.2 เครื่องหมายคณิตศาสตร์พื้นฐานสำหรับจำนวนจริง

เครื่องหมาย	ความหมาย
-	ลบ
+	บวก
*	คูณ
/	หาร(แบบมีเศษ)

5.2 เครื่องหมาย ++ และ --

เครื่องหมาย ++ และ -- อาจจะอยู่หน้าหรือหลังตัวแปร เครื่องหมาย ++ จะหมายถึง เพิ่มค่าของตัวแปรขึ้น 1 ค่า สำหรับเครื่องหมาย -- หมายถึงลดค่าของตัวแปรลงไป 1ค่า นั่นคือ

Y = x + 1

จะเท่ากับ

Y = x++ หรือ ++x

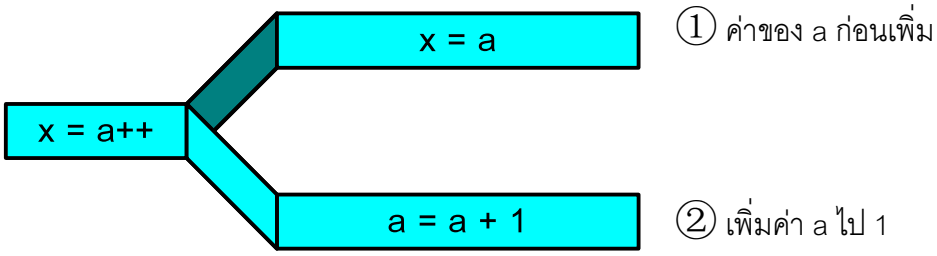
และ

Y = x – 1

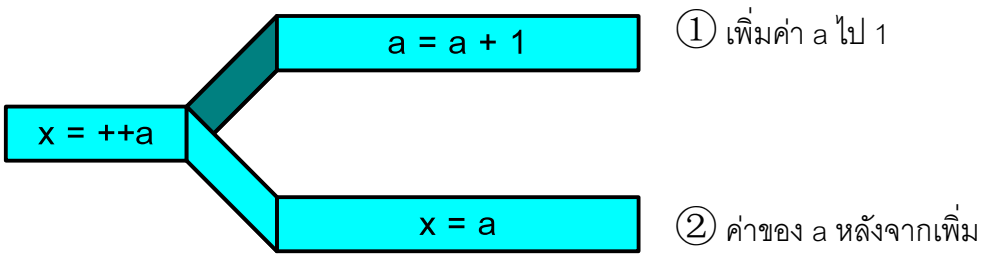
จะเท่ากับ

Y = x-- หรือ --x

เครื่องหมาย ++ และ -- สามารถนำไปประยุกต์ใช้ได้กับทั้งจำนวนเต็มและจำนวนจริง การใช้เครื่องหมาย ++ หรือ -- ก่อนหรือหลังตัวแปรจะให้ผลลัพธ์เท่ากัน ยกเว้นในกรณีที่มีการใช้ตัวแปรอื่นมารับค่า ยกตัวอย่างเช่น x = a++ ในรูปที่ 5.1 การทำงานจะแบ่งเป็น 2 ส่วน คือ จะให้ x = a ก่อน จากนั้นค่อยเพิ่มค่าของ a ไป 1



รูปที่ 5.1 การใช้เครื่องหมาย ++ หลังตัวแปร



รูปที่ 5.2 การใช้เครื่องหมาย ++ หน้าตัวแปร

กลับกันถ้าคำนวณ x = ++a ขั้นตอนการทำงานภายในจะเป็นดังรูปที่ 5.2 คือจะทำการเพิ่มค่า a ไป 1 ก่อนหลังจากนั้นค่อยนำค่า a ที่เปลี่ยนแปลงแล้วไปให้ x การทำงานของเครื่องหมาย – ก็ทำงานในลักษณะเดียวกันกับเครื่องหมาย ++ ตัวอย่างโปรแกรมที่ 5.1 และ 5.2 แสดงการทำงานและผลที่ได้จากการใช้เครื่องหมาย ++ หน้าและหลังตัวแปรตามลำดับ

ตัวอย่างโปรแกรมที่ 5.1

```
1  int main(int argc, char **argv) {
2      int x,y;
3      x = 5;
4      y = ++x;
5      printf("X = %d\n", x);
6      printf("Y = %d\n", x);
7      system("PAUSE");
8  }
```

ผลการรัน

```
X = 6
Y = 6
```

ตัวอย่างโปรแกรมที่ 5.2

```
1  int main(int argc, char **argv) {
2      int x,y;
3      x = 5;
4      y = x++;
5      printf("X = %d\n", x);
6      printf("Y = %d\n", x);
7      system("PAUSE");
8  }
```

ผลการรัน

```
X = 6
Y = 5
```

5.3 นิพจน์คณิตศาสตร์

นิพจน์คณิตศาสตร์ประกอบด้วยค่าคงที่ ตัวแปร ฟังก์ชัน หรือการรวมกันของค่าคงที่ ตัวแปร ฟังก์ชัน ตัวปฏิบัติการทางคณิตศาสตร์ และเครื่องหมายวงเล็บเปิดปิด นิพจน์คณิตศาสตร์จะคล้ายกับสมการคณิตศาสตร์ เช่น การหาพื้นที่ (Area) ของรูปสามเหลี่ยม

Area = 0.5 x base x height

เมื่อเขียนเป็นนิพจน์คณิตศาสตร์จะได้ดังนี้

Area = 0.5 * base * height

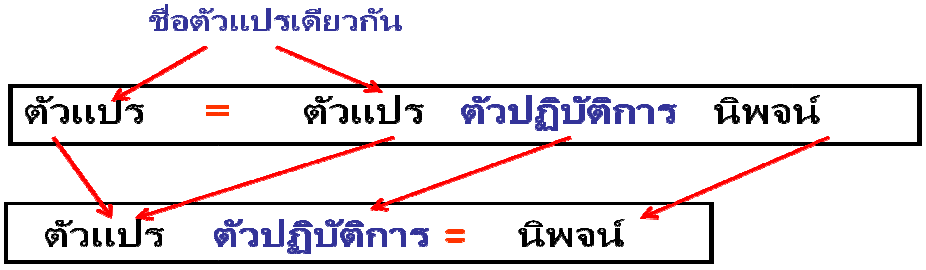
เครื่องหมายคณิตศาสตร์แต่ละชนิดจะมีลำดับในการประมวลผลต่างกันดังในตารางที่ 5.3 นอกจากนี้อาจจะใช้วงเล็บคร่อมนิพจน์เพื่อให้ถูกประมวลผลก่อนและถ้ามีวงเล็บหลายชั้น นิพจน์ที่อยู่ชั้นในสุดจะถูกประมวลผลก่อน เครื่องหมายคณิตศาสตร์ที่มีลำดับเดียวกันจะถูกประมวลผลจากซ้ายไปขวา

ตารางที่ 5.3 ลำดับการประมวลผลของเครื่องหมายคณิตศาสตร์

เครื่องหมายคณิตศาสตร์	ลำดับการประมวลผล
++ , --	1
- (เครื่องหมายลบหน้าตัวเลข)	2
* , / , %	3
+ , -	4

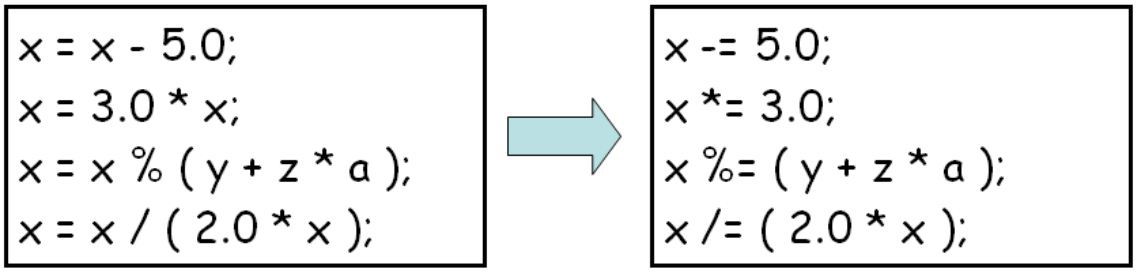
ตัวอย่างเช่น

(2+3)*5	=	25	เอา 2 บวก 3 ได้ 5 แล้วคูณด้วย 5 ได้ 25
2+3*5	=	17	เอา 3 คูณ 5 ได้ 15 แล้วบวกด้วย 2 ได้ 17
10/2*3	=	15	เอา 10 หาร 2 ได้ 5 แล้วคูณด้วย 3 ได้ 15
(7+3)*((10-2))	=	80	เอา 10 ลบด้วย 2 ได้ 8 จากนั้นนำ 10 คูณด้วย 8 ได้ 80
(5+2)*15%4	=	1	เอา 5 บวกกับ 2 ได้ 7 แล้วคูณด้วย 15 ได้ 105 จากนั้นหารด้วย 4 เหลือเศษ 1



รูปที่ 5.3 การเปลี่ยนนิพจน์ให้อยู่ในรูปย่อ

ในภาษาซีสามารถเขียนนิพจน์คณิตศาสตร์ให้อยู่ในรูปย่อเพื่อทำให้โปรแกรมต้นฉบับสั้นลง รูปที่ 5.3 แสดงการย่อนิพจน์ให้สั้นลง เมื่อมีตัวแปรเดียวกันอยู่ทั้งทางด้านซ้าย และทางด้านขวาของเครื่องหมาย “=” เราสามารถลบชื่อตัวแปรนั้นที่อยู่ทางด้านขวาของเครื่องหมาย “=” ออกได้และนำเอาตัวปฏิบัติการทางคณิตศาสตร์มาไว้ข้างหน้าเครื่องหมาย “=” แต่จะต้องไม่มีช่องว่างใดๆ ระหว่างตัวปฏิบัติการและเครื่องหมาย “=” รูปที่ 5.4 แสดงตัวอย่างของการเปลี่ยนนิพจน์ให้อยู่ในรูปย่อ ตัวอย่างโปรแกรมที่ 5.3 เปรียบเทียบการทำงานของการใช้นิพจน์ในรูปแบบเต็มและรูปแบบย่อ ซึ่งจะได้ผลลัพธ์เท่ากัน



รูปที่ 5.4 ตัวอย่างการเปลี่ยนนิพจน์ให้อยู่ในรูปย่อ

ตัวอย่างโปรแกรมที่ 5.3

```
1  int main(int argc, char **argv) {
2      int x,y;
3      x = 5;   y= 5;
4      x += 10;  y = y + 10;
5      printf(“X = %d\n”, x);
6      printf(“Y = %d\n”, x);
7      system(“PAUSE”);
8  }
```

ผลการรัน

```
X = 15
Y = 15
```

5.4 การกำหนดค่าให้ตัวแปร

พิจารณาโปรแกรมที่ 5.4 มีการกำหนดค่าให้กับตัวแปรโดยใช้เครื่องหมายเท่ากับ “=” การกำหนดค่าในลักษณะนี้จะเป็นการนำค่าที่อยู่ด้านขวาของเครื่องหมายเท่ากับ ไปเก็บไว้ในตัวแปร ตัวแปรดังกล่าวจะต้องอยู่ด้านซ้ายเสมอ ตัวแปร base, height ถูกกำหนดให้มีค่าเป็น 5 และ 10 จากบรรทัดที่ 4 และ 5 ตามลำดับ ส่วน area มีค่าเท่ากับผลคูณของ 0.5 x base x height มีค่าเท่ากับ 25

ตัวอย่างโปรแกรมที่ 5.4

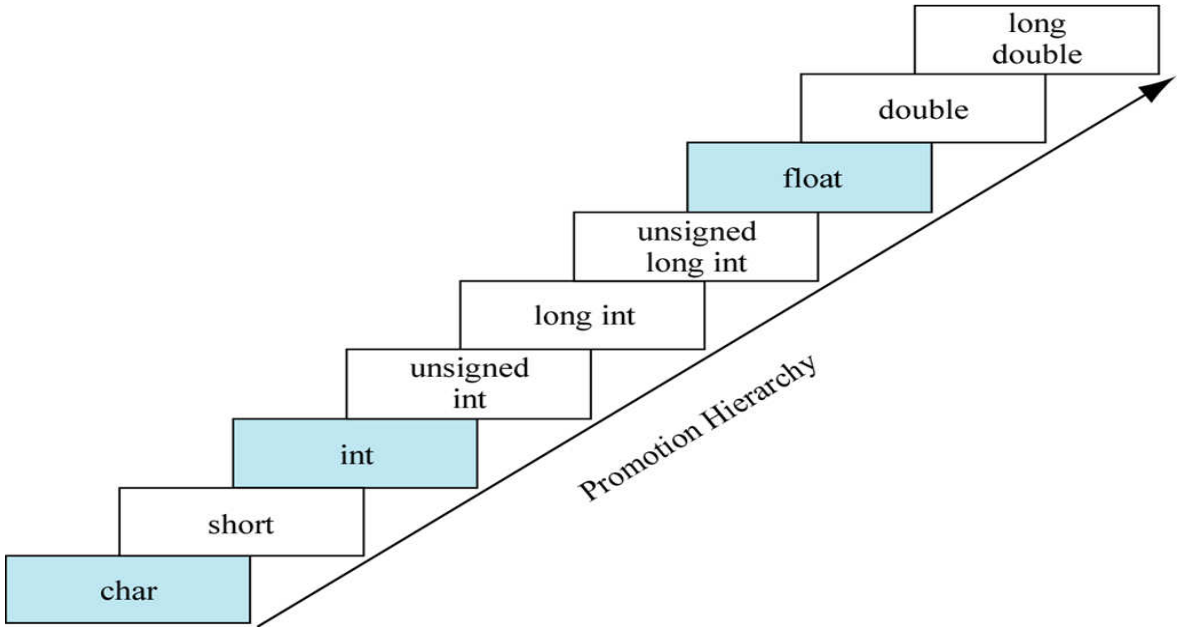
1	int main(int argc, char **argv) {	**ผลการรัน**
2	float base, height;	
3	float area;	
4	base = 5.0;	
5	height = 10.0;	Area of triangle = 25.000000
6	area = 0.5 * base * height;	
7	printf(“Area of triangle = %f\n”, area);	
8	system(“PAUSE”);	
9	}	

การกระทำทางคณิตศาสตร์ส่วนใหญ่ไม่ใช่เป็นเพียงการกระทำของข้อมูลเดียวกัน บางครั้งจะเป็นการกระทำร่วมกันระหว่างจำนวนเต็ม และ จำนวนจริง ดังนั้นผลลัพธ์ที่ได้จะมีประเภทข้อมูลเป็นชนิดใด ผู้พัฒนาซอฟต์แวร์ควรจะทราบเพื่อจะได้ประกาศตัวแปรที่มีประเภทข้อมูลได้ถูกต้องเพื่อเก็บผลลัพธ์ที่ได้ การที่ผลลัพธ์เปลี่ยนแปลงชนิดของข้อมูลได้เราเรียกว่า การเปลี่ยนประเภทของข้อมูล ซึ่งในภาษาซีมีทั้งหมด 2 ประเภท คือ

- การเปลี่ยนประเภทของข้อมูลโดยอัตโนมัติ
- การเปลี่ยนประเภทของข้อมูลโดยตรง

5.4.1 การเปลี่ยนประเภทของข้อมูลโดยอัตโนมัติ

เมื่อมีการกระทำทางคณิตศาสตร์ของข้อมูลต่างประเภทกัน ภาษาซีจะทำการเปลี่ยนประเภทข้อมูลของผลลัพธ์โดยอัตโนมัติ ภาษาซีเลือกที่จะเปลี่ยนประเภทข้อมูลของผลลัพธ์ตามรูปที่ 5.5 ซึ่งหมายถึงประเภทข้อมูลที่อยู่สูงกว่าจะเป็นประเภทข้อมูลที่จะเป็นผลลัพธ์ เช่น ถ้ามีตัวแปรประเภท int กระทำทางคณิตศาสตร์กับ ตัวแปรประเภท float ผลลัพธ์จะเป็นตัวแปรประเภท float เนื่องจาก float อยู่สูงกว่า int



รูปที่ 5.5 ลำดับการเปลี่ยนประเภทข้อมูลแบบอัตโนมัติในภาษาซี

5.4.2 การเปลี่ยนประเภทของข้อมูลโดยตรง

บางครั้งผู้พัฒนาซอฟต์แวร์ต้องการจะเปลี่ยนแปลงประเภทของข้อมูลชั่วคราว ผู้พัฒนาซอฟต์แวร์สามารถกำหนดประเภทข้อมูลของผลลัพธ์ได้โดยตรง การเปลี่ยนประเภทข้อมูลประเภทนี้จะถูกใช้บ่อยเมื่อต้องการผ่านค่าให้กับฟังก์ชันเพื่อหลีกเลี่ยง warning ที่จะเกิดตอนคอมไพล์โปรแกรมต้นฉบับเมื่อมีการกระทำทางคณิตศาสตร์ของข้อมูลต่างประเภทกันในภาษาซี

การเปลี่ยนประเภทของข้อมูลโดยตรงจะใช้อยู่ในรูปของวงเล็บเปิดปิด ข้างในจะเป็นประเภทข้อมูลที่ต้องการ ไว้ข้างหน้านิพจน์ที่ต้องการเปลี่ยนประเภทของข้อมูล การทำแบบนี้เรียกว่า Casting

(ประเภทข้อมูลที่ต้องการ) นิพจน์

ตัวอย่างการใช้งาน

- (int) (5.0 /2.0) ผลลัพธ์ที่ได้จะเป็น 2 แทนที่จะเป็น 2.5 เนื่องจากเราบังคับให้ผลลัพธ์ อยู่ในรูปของ int ซึ่งเป็นจำนวนเต็ม
- (long)5 ผลลัพธ์ที่ได้จะเป็น 5 แต่จะเป็นประเภทข้อมูลแบบ long เนื่องด้วยบางครั้งฟังก์ชันต้องการรับค่าเป็น long แต่เลข 5 จะเป็นประเภทข้อมูลแบบ int โดยปริยาย ถ้าไม่มีการเปลี่ยนประเภทข้อมูลก่อนจะทำให้เกิด warning ตอนคอมไพล์

ตัวอย่างโปรแกรมที่ 5.5

1	int main(int argc, char **argv) {	**ผลการรัน**
2	int base, height; float area1, area2, area3;	
3	int area4;	
4	base = 3;	
5	height = 3;	Area1 = 4.500000
6	area1 = 0.5 * base * height;	Area2 = 0.000000
7	area2 = (1 / 2) * base * height;	Area3 = 4.500000
8	area3 = (1.0 / 2) * base * height;	Area4 = 4
9	area4 = 0.5 * base * height;	
10	printf("Area1 = %f\n", area1);	
11	printf("Area2 = %f\n", area2);	
12	printf("Area3 = %f\n", area3);	
13	printf("Area4 = %d\n", area4);	
14	system("PAUSE");	
15	}	

จากตัวอย่างโปรแกรมที่ 5.5

- ในบรรทัดที่ 6 จะเห็นว่า 0.5 เป็นจำนวนจริงโดยปริยายคือ float ซึ่งกระทำทางคณิตศาสตร์กับจำนวนเต็ม(int) 2 ตัวคือ base และ height ผลลัพธ์ที่ได้จึงถูก เปลี่ยนเป็นประเภท float ซึ่ง area1 ถูกประกาศเป็น float อยู่แล้วจึงสามารถเก็บค่าได้อย่างถูกต้อง
- ในบรรทัดที่ 7 จะเห็นว่า (1 / 2) เป็นจำนวนเต็มหารกับจำนวนเต็มดังนั้นผลที่ได้จะ ไม่มีการเปลี่ยนประเภททางข้อมูล ดังนั้นผลที่ได้มีค่าเท่ากับ 0 เมื่อทำไปคูณกับ base และ height ก็จะให้ผลลัพธ์เป็น 0 ซึ่งจะให้ผลลัพธ์ที่ผิดจากที่ต้องการ
- ในบรรทัดที่ 8 จะเห็นว่า (1.0 / 2) เป็นการหารกับระหว่างจำนวนจริง 1.0 และจำนวนเต็ม 2 ซึ่งผลลัพธ์ที่ได้จะถูกเปลี่ยนเป็นประเภทข้อมูลแบบจำนวนจริงซึ่งก็คือ 0.5 เมื่อไปคูณกับ base และ height ก็จะได้ผลลัพธ์ที่ถูกต้องตามที่ต้องการเหมือนกับการชุดคำสั่งในบรรทัดที่ 7
- ในบรรทัดที่ 9 ผลลัพธ์ที่ได้คือ 4.5 แต่เนื่องจากตัวแปรที่ใช้ในการจัดเก็บข้อมูลรับได้เฉพาะจำนวนเต็ม ทำให้ค่าที่สามารถเก็บได้คือ 4 ทำให้ผลลัพธ์ที่ได้ไม่ถูกต้องตามที่ต้องการ และเป็นข้อควรระวังอย่างยิ่งเนื่องจากคอมไพเลอร์ภาษาซีจะไม่มีการเตือนใดๆ ทั้งสิ้น

ตัวอย่างโปรแกรมที่ 5.6

1	int main(int argc, char **argv) {	**ผลการรัน**
2	float a;	
3	int b, c, d, e;	
4		
5	a = (int)5.5 + (float)10.4;	a = 15.400000
6	b = (int)5.5 + 10.4;	b = 15
7	c = (int)(5.5 / 1.1);	c = 5
8	d = (int)5.5 / 1.1;	d = 4
9	e = 5.5 / (int)1.1;	e = 5
10	printf("a = %f\n", a);	
11	printf("b = %d\n", b);	
12	printf("c = %d\n", c);	
13	printf("d = %d\n", d);	
14	printf("e = %d\n", e);	
15	system("PAUSE");	
16	}	

จากตัวอย่างโปรแกรมที่ 5.6 มีการประกาศตัวแปร a เป็นประเภทข้อมูลแบบจำนวนจริง (float) และ b, c, d, e เป็นประเภทข้อมูลแบบจำนวนเต็ม (int)

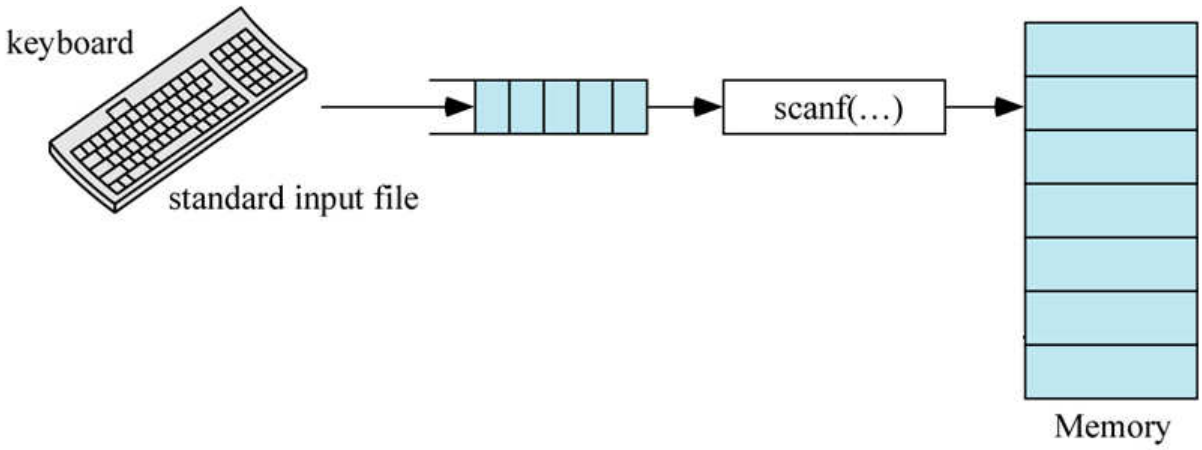
- บรรทัดที่ 5 (int)5.5 จะให้ค่า 5 และ (float)10.4 จะให้ค่า 10.4 ดังนั้นเมื่อบวกกันภาษาซีจะเปลี่ยนประเภทข้อมูลของผลลัพธ์เป็น float ซึ่งทำให้ได้ค่า 15.4 เก็บไว้ในตัวแปร a
- บรรทัดที่ 6 (int)5.5 จะให้ค่า 5 เมื่อบวกกับ 10.4 ทำให้ได้ค่า 15.4 ภาษาซีเปลี่ยนประเภทของข้อมูลผลลัพธ์เป็นจำนวนจริงแต่อย่างไรก็ตามตัวแปร b ที่ใช้เก็บค่าผลลัพธ์ไม่สามารถ เก็บจุดทศนิยมได้ จึงเก็บได้เพียงค่า 15
- บรรทัดที่ 7 ผลลัพธ์ที่ได้จาก 5.5/1.1 คือ 5 เมื่อผ่านการ casting (int) ก็ได้ 5 เก็บในตัวแปร c
- บรรทัดที่ 8 (int)5.5 จะให้ค่า 5 เมื่อนำไปหารกับ 1.1 ทำให้ได้ค่า 4.545454 จากนั้นผ่านการ casting (int) ทำให้ค่าที่เก็บในตัวแปร d คือ 4
- บรรทัดที่ 9 (int)1.1 จะให้ค่า 1 เมื่อนำไปหารกับ 5.5 จะได้ผลลัพธ์คือ 5.5 ตัวแปร e สามารถเก็บค่าได้เฉพาะจำนวนเต็มดังนั้นตัวแปร e จึงเก็บค่า 5

บทที่ 6
การป้อนข้อมูล

การป้อนข้อมูลทางแป้นพิมพ์เป็นวิธีหนึ่งที่ใช้ในการกำหนดค่าให้กับตัวแปรในโปรแกรมต้นฉบับ ซึ่งการทำงานนั้นก็คือการนำค่าที่ได้รับจากแป้นพิมพ์ไปเก็บไว้ในหน่วยความจำโดยตำแหน่งของหน่วยความจำนั้นก็คือตำแหน่งที่ตัวแปรใช้ในการเก็บข้อมูล ในภาษาซีมีฟังก์ชันสำหรับป้อนข้อมูลที่นิยมใช้คือ scanf ( ) , getchar ( ) , getche ( ), getch () และ gets ( )

6.1 ฟังก์ชัน scanf( )

ฟังก์ชัน scanf ( ) มีรูปแบบการทำงานดังรูปที่ 6.1 กล่าวคือเมื่อโปรแกรมทำงานถึงฟังก์ชันนี้จะหยุดเพื่อให้ป้อนข้อมูล โดยข้อมูลที่ป้อนจะแสดงบนจอภาพ เมื่อป้อนข้อมูลเสร็จกด Enter ข้อมูลจะผ่านตัวฟังก์ชัน scanf เพื่อแปลงค่าที่ได้ไปเก็บไว้ยังหน่วยความจำหลัก



รูปที่ 6.1 การทำงานของฟังก์ชัน scanf

การใช้งานฟังก์ชัน scanf อย่างง่ายมีรูปแบบดังนี้

```
scanf ("format  code" , &var);
```

- Format code จะมีลักษณะการใช้งานคล้ายกับ format code ที่ใช้กับฟังก์ชัน printf ซึ่งเป็นตัวกำหนดประเภทของข้อมูลที่ต้องการรับผ่านแป้นพิมพ์ ตารางที่ 6.1 แสดง format สำหรับฟังก์ชัน scanf
- var คือชื่อตัวแปรที่ต้องการใช้ในการเก็บข้อมูลที่รับจากแป้นพิมพ์

ตารางที่ 6.1 รหัส Format code ของฟังก์ชัน scanf

รหัสรูปแบบ	ความหมาย
%c	ใช้กับตัวแปรที่เก็บค่าเป็นตัวอักษรเพียงตัวเดียว
%s	ใช้กับตัวแปรที่เก็บค่าเป็นข้อความที่เก็บในตัวแปรชุด
%d	ใช้กับตัวแปรที่เก็บค่าเป็นเลขจำนวนเต็ม
%u	ใช้กับตัวแปรที่เก็บค่าเป็นเลขจำนวนเต็มบวก
%f	ใช้กับตัวแปรที่เก็บค่าที่เป็นเลขทศนิยม (float)
%lf	ใช้กับตัวแปรที่เก็บค่าที่เป็นเลขทศนิยม (double)

ตัวอย่างโปรแกรมที่ 6.1

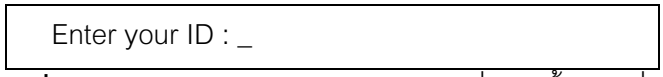
```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      int   x;                                  // กำหนดตัวแปร x ที่เป็นชนิด integer
4      printf("Enter your ID: ");
5      scanf("%d", &x);                          // รับข้อมูลเก็บไว้ใน x
6      printf(" Your ID = %d\n", x);             // พิมพ์จำนวนเต็ม x
7      system("pause");
8      return 0;
9  }
```

ตัวอย่างโปรแกรมที่ 6.1 เป็นตัวอย่างของการใช้งานของฟังก์ชัน printf และ scanf

- บรรทัดที่ 1 เป็นการประกาศส่วนของ Preprocessor Directive (#include) เพื่อที่จะใช้งานฟังก์ชัน printf และ scanf
- บรรทัดที่ 2 เป็นการประกาศฟังก์ชัน main ให้กับโปรแกรมต้นฉบับ
- บรรทัดที่ 3 ประกาศตัวแปร x ที่เก็บประเภทข้อมูลชนิดจำนวนเต็ม
- บรรทัดที่ 4 เป็นการเรียกใช้ฟังก์ชัน printf เพื่อแสดงคำว่า “Enter your ID” ออกทางจอภาพ
- บรรทัดที่ 5 เป็นการเรียกใช้ฟังก์ชัน scanf เพื่อเก็บค่าจำนวนเต็มที่ผู้ใช้ป้อนผ่านแป้นพิมพ์ไปเก็บไว้ในตัวแปร x
- บรรทัดที่ 6 เป็นการเรียกใช้ฟังก์ชัน printf อีกครั้งเพื่อแสดงคำว่า “Your ID = “ ตามด้วยค่าที่เก็บไว้ในตัวแปร x

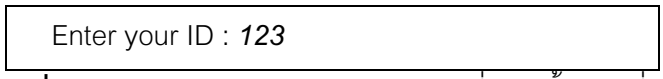
ผลการทำงานของตัวอย่างโปรแกรมที่ 6.1 แบ่งออกได้เป็น 3 ขั้นตอน

- ขั้นตอนที่ 1 ดังรูปที่ 6.2 กล่าวคือหน้าจอจะแสดงคำว่า Enter your ID : และโปรแกรมจะหยุด เพื่อรอการป้อนข้อมูลจากผู้ใช้



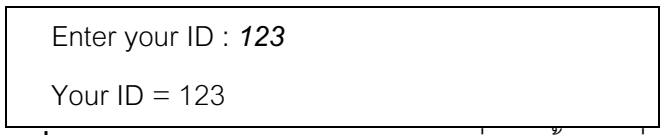
รูปที่ 6.2 ผลการรันของตัวอย่างโปรแกรมที่ 6.1 ขั้นตอนที่ 1

- ขั้นตอนที่ 2 ผู้ใช้จำเป็นต้องป้อนจำนวนเต็มเข้าไปด้วยการพิมพ์ผ่านแป้นพิมพ์ ดังรูปที่ 6.3 กำหนดให้ผู้ใช้ ป้อนตัวเลข 123 แล้วกดปุ่ม Enter



รูปที่ 6.3 ผลการรันของตัวอย่างโปรแกรมที่ 6.1 ขั้นตอนที่ 2

- ขั้นตอนที่ 3 โปรแกรมจะแสดงข้อความ “Your ID = “ แล้วตามด้วยค่าที่ผู้ใช้พิมพ์ในขั้นตอนที่ 2 ดังรูปที่ 6.4



รูปที่ 6.4 ผลการรันของตัวอย่างโปรแกรมที่ 6.1 ขั้นตอนที่ 3

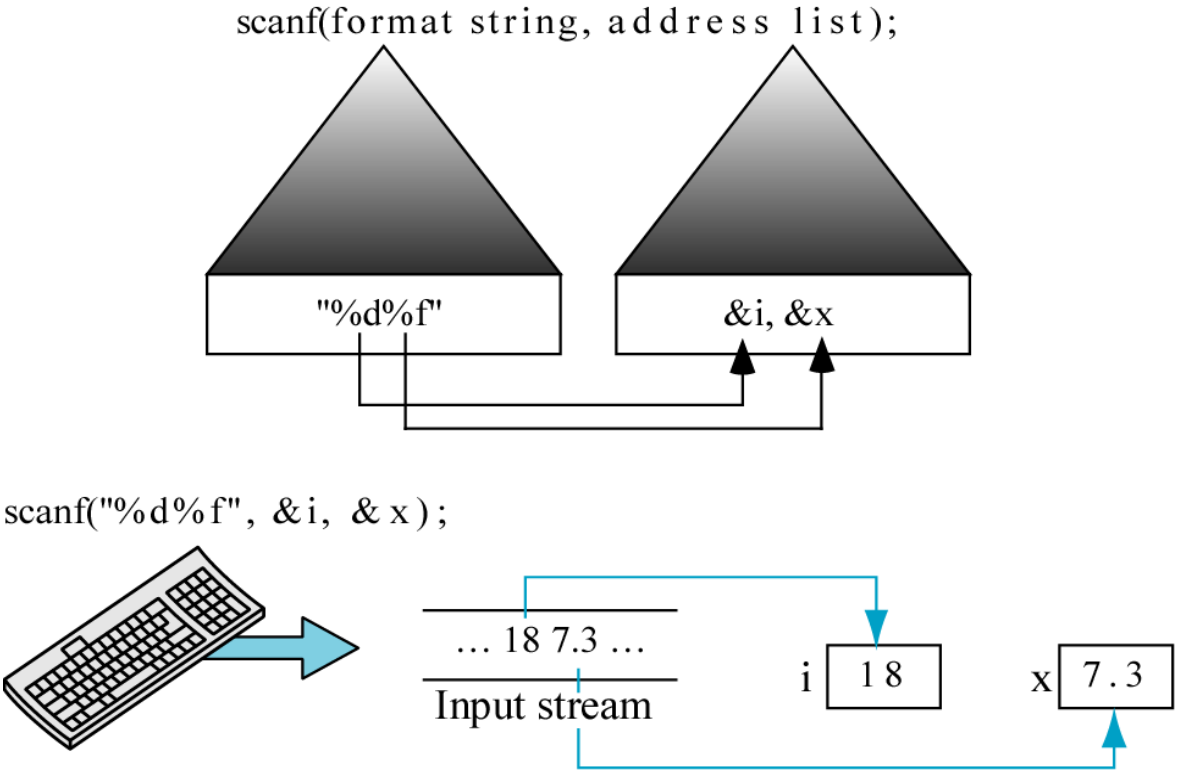
ตัวอย่างโปรแกรมที่ 6.2

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      char ch;  float f;  double d;
4      printf("Enter a character: ");
5      scanf("%c", &ch);
6      printf("Enter a floating point number: “);
7      scanf(“%f”, &f);
8      printf(“Enter a double precision floating point number: ”);
9     scanf(“%lf”, &d);
10 }
```

ตัวอย่างโปรแกรมที่ 6.2 แสดงการใช้งานฟังก์ชัน scanf กับตัวแปรประเภทตัวอักขระ (char), ตัวแปรประเภทจำนวนจริง (float), และ ตัวแปรประเภทจำนวนจริงที่มีความแม่นยำสูง (double) ในบรรทัดที่ 5, 7 และ 9 ตามลำดับ

6.2 รูปแบบการใช้ scanf เก็บค่ามากกว่าหนึ่งตัวแปร

ฟังก์ชัน scanf นอกจากจะใช้เพื่อเก็บค่าให้ตัวแปร 1 ตัวแล้ว ฟังก์ชัน scanf ยังสามารถใช้เพื่อเก็บค่ามากกว่าหนึ่งตัวแปรได้อีกด้วย ดังรูปที่ 6.5 แสดงการทำงานของฟังก์ชัน scanf เพื่อเก็บค่าให้กับตัวแปร 2 ตัว คือ i และ x โดยที่ i เป็นตัวแปรประเภทจำนวนเต็ม (int) และ x เป็นตัวแปรประเภทจำนวนจริง (float) การเก็บข้อมูลจะเก็บตามลำดับของ format code และชื่อตัวแปร



รูปที่ 6.5 การใช้ฟังก์ชัน scanf เพื่อรับค่าให้กับ 2 ตัวแปร

เพื่อให้เข้าใจหลักการทำงานของการใช้ฟังก์ชัน scanf ในการเก็บค่ามากกว่าหนึ่งตัวแปรมากขึ้น และการรับข้อมูลแบบพิเศษ ขอยกตัวอย่างโปรแกรมต่างๆ ตั้งแต่ตัวอย่างโปรแกรมที่ 6.3 ถึง ตัวอย่างโปรแกรมที่ 6.6

ตัวอย่างโปรแกรมที่ 6.3

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      int  a,  b,  c;
4      printf("Enter three integer number: ");
5      scanf("%d %d %d", &a, &b, &c);
6      printf("a = %d  b  = %d  c = %d\n", a, b, c);
7  }
```

ตัวอย่างโปรแกรมที่ 6.3 แสดงการใช้ฟังก์ชัน scanf ในการรับค่าตัวแปรประเภทจำนวนเต็ม 3 ตัวคือ a, b, และ c การป้อนค่าจะต้องป้อนค่าเป็นจำนวนเต็ม 3 ตัว และเว้นช่องว่างระหว่างตัวแปรทั้ง 3 ดังรูปที่ 6.6 ผู้ใช้ป้อนตัวเลข 123 เว้นวรรค 456 เว้นวรรค 78 ค่าที่ถูกป้อนมานั้นจะถูกเก็บไว้ในตัวแปร a, b, และ c ตามลำดับ

```
Enter three integer number :  123  456  78
a = 123  b = 456  c = 78
```

รูปที่ 6.6 ผลการรันของตัวอย่างโปรแกรมที่ 6.3

ตัวอย่างโปรแกรมที่ 6.4

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      int  a,  b,  c;
4      printf("Enter three integer number: ");
5      scanf("%3d %d %d", &a, &b, &c);
6      printf("a = %d  b  = %d  c = %d\n", a, b, c);
7  }
```

ตัวอย่างโปรแกรมที่ 6.4 แสดงการใช้ฟังก์ชัน scanf ในการรับค่าตัวแปรประเภทจำนวนเต็ม 3 ตัวคือ a, b, และ c แต่มีการกำหนด %3d เพื่อเก็บค่าเข้าตัวแปร a ซึ่งเป็นการกำหนดให้เก็บค่าแค่ 3 หลัก รูปที่ 6.7 แสดงผลการรันของโปรแกรมจะเห็นว่าตัวแปร a เก็บค่า 123 ทั้งๆที่ผู้ใช้ป้อนตัวเลข 1234

```
Enter three integer number :  1234  456  78
a = 123  b = 456  c = 78
```

รูปที่ 6.7 ผลการรันของตัวอย่างโปรแกรมที่ 6.4

ตัวอย่างโปรแกรมที่ 6.5

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      int  a,  b,  c;
4      printf("Enter three integer number: ");
5      scanf("%d/%d/%d", &a, &b, &c);
6      printf("a = %d  b  = %d  c = %d\n", a, b, c);
7  }
```

ตัวอย่างโปรแกรมที่ 6.5 แสดงการใช้ฟังก์ชัน scanf ในการรับค่าตัวแปรประเภทจำนวนเต็ม 3 ตัวคือ a, b, และ c แต่รูปแบบของ scanf ระหว่าง format code %d จะมีเครื่องหมาย '/' คั่นอยู่ ดังนั้นการป้อนข้อมูลให้กับตัวแปร จำเป็นต้องใส่เครื่องหมาย '/' เข้าไปด้วยดังรูปที่ 6.8 ที่แสดงผลการรันของโปรแกรม

```
Enter three integer number :  123/456/78
a = 123  b = 456  c = 78
```

รูปที่ 6.8 ผลการรันของตัวอย่างโปรแกรมที่ 6.5

ตัวอย่างโปรแกรมที่ 6.6

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      int  a,  b;
4      char  c;
5      printf("Enter code: ");
6      scanf("%d %c %d", &a, &c, &b);
7      printf("a = %d  b  = %d  c = %c\n", a, b, c);
   }
```

ตัวอย่างโปรแกรมที่ 6.6 แสดงการใช้ฟังก์ชัน scanf ในการรับค่าตัวแปรประเภทจำนวนเต็มร่วมกับการรับค่าตัวแปรประเภทอักขระ โดยจาก format code ของฟังก์ชัน scanf นี้คือ “%d %c %d” หมายถึงตัวแรกจะรับจำนวนเต็ม ตัวที่ 2 จะรับตัวอักขระ และตัวที่ 3 จะรับจำนวนเต็ม ดังนั้นค่าที่รับจึงถูกส่งไปเก็บที่ตัวแปร a, c, และ b ตามลำดับ รูปที่ 6.9 แสดงผลการรันของโปรแกรมโดยการพิมพ์ค่าที่อยู่ใน a, b และ c ตามลำดับ

```
Enter three integer number : 123 X 78
a = 123  b = 78  c = X
```

รูปที่ 6.9 ผลการรันของตัวอย่างโปรแกรมที่ 6.6

การใช้ฟังก์ชัน scanf มีข้อควรระวังเมื่อใช้ในการรับข้อมูลประเภทสตริงก์ เนื่องจาก scanf จะตัดค่าแค่เครื่องหมายเว้นวรรคทำให้บางครั้งส่งผลให้โปรแกรมทำงานผิดพลาดตัวอย่างโปรแกรมที่ 6.7 แสดงการใช้งานฟังก์ชัน scanf กับข้อมูลประเภทสตริงก์ซึ่งผลการรันที่ได้จะเห็นได้ว่าผู้ใช้ป้อนคำว่า “Hello World” แต่ scanf สามารถรับได้แค่คำว่า Hello เพื่อเก็บค่าไว้ในตัวแปร str ดังนั้นผู้พัฒนาโปรแกรมส่วนใหญ่จะนิยมใช้ฟังก์ชัน gets มากกว่า scanf ในการจัดการกับสตริงก์ ซึ่ง gets จะกล่าวถึงต่อไปในภายหลัง

ตัวอย่างโปรแกรมที่ 6.7

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      char str[60];
4      printf("Enter message: ");
5      scanf("%s", str);
6      printf("You typed -> %s\n", str);
7  }
```

ผลการรัน

```
Enter message: Hello World
You typed -> Hello
```

6.3 ฟังก์ชัน getchar( )

ฟังก์ชัน getchar( ) ใช้สำหรับการรับค่าจากการป้อนตัวอักษรผ่านทางแป้นพิมพ์ โดยจะรับตัวอักษรเพียง 1 ตัวเท่านั้น และแสดงตัวอักษรบนจอภาพ

```
ch = getchar( );
```

เมื่อโปรแกรมทำงานถึงคำสั่งนี้จะหยุดเพื่อให้ป้อนตัวอักษร 1 ตัว หลังจากนั้นกด Enter ตัวอักษรที่ป้อนจะถูกเก็บไว้ในตัวแปร ch ซึ่งเป็นชนิดตัวอักษรและเคอร์เซอร์จะขึ้นบรรทัดใหม่ฟังก์ชัน getchar( ) การทำงานของ getchar() นั้นจะเหมือนกับการใช้งาน scanf("%c", ...) ทุกประการ และจำเป็นต้องกำหนด Preprocessor Directive #include<stdio.h> เช่นเดียวกับฟังก์ชัน scanf( )

ตัวอย่างโปรแกรมที่ 6.8

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      char ch;
4      printf("Enter a character: ");
5      ch = getchar( );
6      printf("You typed -> %c\n", ch);
7  }
```

ผลการรัน

```
Enter a character: X
You typed -> X
```

ตัวอย่างโปรแกรมที่ 6.8 แสดงการใช้งานของฟังก์ชัน getchar( ) ในบรรทัดที่ 5 ซึ่งจะเป็นการรอรับค่าตัวอักษร 1 ตัวจากผู้ใช้ผ่านทางแป้นพิมพ์ จากนั้นนำค่าที่ได้รับนั้นเก็บไว้ในตัวแปร ch และแสดงค่าของตัวแปร ch ผ่านฟังก์ชัน printf( ) ในบรรทัดที่ 6

ตัวอย่างโปรแกรมที่ 6.9 จะให้ผลลัพธ์เหมือนกับตัวอย่างโปรแกรมที่ 6.8 ทุกประการแต่เปลี่ยนจากการใช้ฟังก์ชัน printf ในการแสดงผลมาใช้ฟังก์ชัน putchar( ) ในการแสดงผลแทน putchar( ) เป็นฟังก์ชันที่ใช้แสดงค่าตัวแปรที่เป็นตัวอักษร 1 ตัวออกทางหน้าจอ ซึ่งมีหลักการทำงานเหมือนกับ printf("%c" , ...) ทุกประการ

ตัวอย่างโปรแกรมที่ 6.9

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      char ch;
4      printf("Enter a character: ");
5      ch = getchar( );
6      printf("You typed -> “);   putchar(ch);
7  }
```

ผลการรัน

```
Enter a character: X
You typed -> X
```

6.4 ฟังก์ชัน getche( ) และ getch( )

ฟังก์ชัน getche( ) และ getch( ) มีรูปแบบการใช้งาน ดังนี้

```
ch1  =  getche( );
ch2  =  getch( );
```

6.4.1 ฟังก์ชัน getche( )

จะรับตัวอักษร 1 ตัวที่ป้อนทางแป้นพิมพ์ และจะแสดงตัวอักษรบนจอภาพ เมื่อป้อนข้อมูลเสร็จไม่ต้องกด Enter และเคอร์เซอร์จะไม่ขึ้นบรรทัดใหม่ ดังตัวอย่างโปรแกรมที่ 6.10

ตัวอย่างโปรแกรมที่ 6.10

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      char ch;
4      printf("Enter a character: ");   ch = getche( );
5      printf("\nYou typed -> “);        putchar(ch);
6  }
```

ผลการรัน

```
Enter a character: X
You typed -> X
```

6.4.2 ฟังก์ชัน getch( )

การทำงานของฟังก์ชัน getch( ) จะคล้ายกับฟังก์ชัน getche( ) ต่างกันตรงที่จะไม่แสดงตัวอักษรขณะป้อนข้อมูล ดังตัวอย่างโปรแกรมที่ 6.11

ตัวอย่างโปรแกรมที่ 6.11

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      char ch;
4     printf("Enter a character: ");
5     ch = getch( );
6     printf("\nYou typed -> “);
7     putchar(ch);
8  }
```

ผลการรัน

```
Enter a character:
You typed -> X
```

6.5 ฟังก์ชัน gets( )

ฟังก์ชัน gets( ) ใช้สำหรับข้อมูลชนิดสตริงก์หรือข้อความซึ่งป้อนทางแป้นพิมพ์โดยมีรูปแบบดังนี้

```
gets(str);
```

เมื่อโปรแกรมทำงานถึงคำสั่งนี้จะหยุดเพื่อให้ป้อนข้อความ เมื่อป้อนเสร็จแล้วกด Enter ข้อความทั้งหมดจะถูกเก็บไว้ในอาร์เรย์สตริงก์ str ฟังก์ชัน gets( ) จะทำให้เคอร์เซอร์ขึ้นบรรทัดใหม่หลังจากกด Enter และจะต้องกำหนด Preprocessor Directive #include<stdio.h>

ตัวอย่างโปรแกรมที่ 6.12 แสดงการใช้งานฟังก์ชัน gets โดยจะเห็นได้ว่า gets( ) สามารถรับสตริงก์ที่มีขนาดยาวได้และรับค่าช่องว่างระหว่างสตริงก์ได้ในขณะที่ ฟังก์ชัน printf( ) ไม่สามารถทำได้ ในการแสดงข้อความสตริงก์ออกมานั้นนอกจากใช้ฟังก์ชัน printf แล้วยังมีฟังก์ชัน puts( ) ซึ่งจะทำงานได้เหมือนกับ printf ทุกประการ

ตัวอย่างโปรแกรมที่ 6.12

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      char str[60];
4     printf("Enter message: ");
5     gets(str);
6     printf("\nYou typed -> “);
7     puts(str);
8  }
```

ผลการรัน

```
Enter message: Hello World
You typed -> Hello World
```

บทที่ 7
คำสั่งเงื่อนไข

ในบทที่ผ่านมาเป็นการเขียนโปรแกรมโดยเรียกใช้คำสั่งประกาศสร้างตัวแปร กำหนดค่าให้กับตัวแปร คำสั่งคำนวณประเภทต่างๆ และมีการเรียกใช้ฟังก์ชัน โดยโปรแกรมจะทำงานเรียงลำดับตั้งแต่คำสั่งแรกไปจนถึงคำสั่งสุดท้าย ซึ่งในบางครั้งอาจจะไม่ต้องการให้เป็นเช่นนั้น ในบทนี้เป็นการแนะนำให้รู้จักวิธีการเขียนโปรแกรมเพื่อควบคุมทิศทางการทำงานของโปรแกรมโดยใช้คำสั่งเงื่อนไข เพื่อให้โปรแกรมทำงานในแบบที่ต้องการได้ ในทางปฏิบัตินั้นสภาพของปัญหาที่ต้องเขียนโปรแกรมขึ้นมาเพื่อแก้ไขความซับซ้อน ซึ่งคงจะไม่ใช่โปรแกรมที่ทำงานเรียงกันไปตั้งแต่ต้นจนจบโปรแกรม แต่ควรจะเป็นโปรแกรมที่ควบคุมทิศทางของการทำงานได้ เช่น ถ้าข้อมูลที่รับเข้ามาเป็นเลขคู่ให้ทำงานแบบหนึ่ง แต่ถ้าข้อมูลที่รับเข้ามาเป็นเลขคี่ให้ทำงานอีกแบบหนึ่ง ซึ่งการควบคุมทิศทางการทำงานจะทำให้โปรแกรมทำงานได้อย่างมีประสิทธิภาพมากขึ้น

7.1 เครื่องหมายเปรียบเทียบ

ในภาษาซีมีเครื่องหมายเปรียบเทียบ (Relational operator) ดังแสดงในตารางที่ 7.1 ผลลัพธ์การเปรียบเทียบจะเป็นจริง (True) หรือเท็จ (False) อย่างใดอย่างหนึ่งเสมอ ค่าจริงหรือเท็จบางครั้งเรียกว่าค่าคงที่บูลีน (Boolean constant) ในภาษาซีค่าจริงและเท็จที่เกิดจากการใช้เครื่องหมายเปรียบเทียบจะมีค่าเป็น 1 และ 0 ตามลำดับ

ตารางที่ 7.1 เครื่องหมายเปรียบเทียบและความหมาย

เครื่องหมาย	ความหมาย
= =	เท่ากับ
!=	ไม่เท่ากับ
>	มากกว่า
>=	มากกว่าหรือเท่ากับ
<	น้อยกว่า
<=	น้อยกว่าหรือเท่ากับ

ในการเปรียบเทียบตัวเลขจะใช้เลขฐานสิบเป็นหลัก กล่าวคือ 1 < 2 < 3 < 4 <.... สำหรับตัวเลขติดลบที่น้อยกว่าจะมีค่ามากกว่าเลขติดลบที่มากกว่า เช่น

10 > 8	ค่าคงที่บูลีน 1 (จริง)
-9 <-3	ค่าคงที่บูลีน 1 (จริง)
13 != 3	ค่าคงที่บูลีน 1 (จริง)
2 < -5	ค่าคงที่บูลีน 0 (เท็จ)

สำหรับการเปรียบเทียบข้อมูลตัวอักษรและสตริงก์จะใช้ค่ารหัสแอสกีเป็นหลัก โดยอักษรที่มีรหัสแอสกีมากกว่าจะมีค่ามากกว่าอักษรที่มีรหัสแอสกีน้อยกว่า สำหรับการเปรียบเทียบสตริงก์จะเริ่มเปรียบเทียบตั้งแต่อักษรตัวแรก ถ้าอักษรตัวแรกมีค่ารหัสแอสกีเท่ากัน ก็จะเปรียบเทียบตัวอักษรถัดไปจนกระทั่งหมดข้อความ ตัวอย่างเช่น

C < c	C มีรหัสแอสกี = 67 , c มีรหัสแอสกี = 99
P > M	P มีรหัสแอสกี = 80 , M มีรหัสแอสกี = 77
PQ < RU	P มีรหัสแอสกี = 80 , R มีรหัสแอสกี = 82
Somsak < Somsri	อักษร 4 ตัวแปรมีค่ารหัสแอสกีเท่ากัน เมื่อเปรียบเทียบถึงอักษรตัวที่ 5 พบว่า a < s เนื่องจาก a มีหรัสแอสกีเป็น 97 s มีรหัสแอสกีเป็น 115

7.2 เครื่องหมายเปรียบเทียบเชิงตรรก

เครื่องหมายเปรียบเทียบเชิงตรรก (Logical operator) ใช้เชื่อมเงื่อนไข 2 เงื่อนไขหรือมากกว่า เพื่อให้การเปรียบเทียบมีความละเอียดมากขึ้น ภาษาซีมีเครื่องหมายเปรียบเทียบเชิงตรรก 3 ชนิด ดังตารางที่ 7.2 ในการเปรียบเทียบผลลัพธ์จะเป็นจริง (มีค่าเป็น 1) หรือเท็จ (มีค่าเป็น 0) ขึ้นกับชนิดเครื่องหมายเปรียบเทียบเชิงตรรกแต่ละชนิด ค่าความเป็นจริงหลังจากการใช้เครื่องหมายเปรียบเทียบเชิงตรรกแสดงในตารางที่ 7.3

ตารางที่ 7.2 เครื่องหมายเปรียบเทียบเชิงตรรกะ

เครื่องหมาย	ความหมาย
!	NOT
&&	AND
||	OR

! จะให้ค่าจริงเมื่อเงื่อนไขหลัง ! เป็นเท็จ และจะให้ค่าเท็จเมื่อเงื่อนไขหลัง ! เป็นจริง

&& จะให้ค่าจริงเมื่อเงื่อนไขทั้งสองเป็นจริงและให้ค่าเท็จเมื่อเงื่อนไขทั้งสองหรือเงื่อนไขใดเงื่อนไขหนึ่งเป็นเท็จ

|| จะให้ค่าจริงเมื่อเงื่อนไขทั้งสองหรือเงื่อนไขใดเงื่อนไขหนึ่งเป็นจริง และจะให้ค่าเท็จเมื่อเงื่อนไขทั้งสองเป็นเท็จ

ตารางที่ 7.3 การใช้เครื่องหมาย ! , && และ ||

เงื่อนไข 1	เงื่อนไข 2	! เงื่อนไข 1	! เงื่อนไข 2	เงื่อนไข 1 && เงื่อนไข 2	เงื่อนไข 1 || เงื่อนไข 2
1	1	0	0	1	1
1	0	0	1	0	1
0	1	1	0	0	1
0	0	1	1	0	0

7.3 ลำดับการประมวลผลของคำสั่งเงื่อนไข

ในคำสั่งเงื่อนไขซึ่งจะกล่าวในหัวข้อถัดไป อาจจะประกอบด้วยเครื่องหมายคณิตศาสตร์ เครื่องหมายเปรียบเทียบและเครื่องหมายเปรียบเทียบเชิงตรรก ในภาษาซีได้จัดลำดับการประมวลผลในคำสั่งเงื่อนไข ดังตารางที่ 7.4

ตารางที่ 7.4 ลำดับการประมวลผลในคำสั่งเงื่อนไข

โอเปอเรเตอร์	ลำดับ
เครื่องหมายคณิตศาสตร์	1
!	2
> , > =, < , < =	3
== , !=	4
&&	5
||	6

7.4 คำสั่ง if

การเลือกทำโดยใช้คำสั่ง if จะใช้ในกรณีที่มีทางเลือกให้ทำงานอยู่เพียงทางเลือกเดียว ผลจากการตรวจสอบเงื่อนไขคือ ทำหรือไม่ทำตามคำสั่งนั้น รูปแบบการเขียนคำสั่ง if แสดงดังต่อไปนี้

```
if (condition) statement;
```

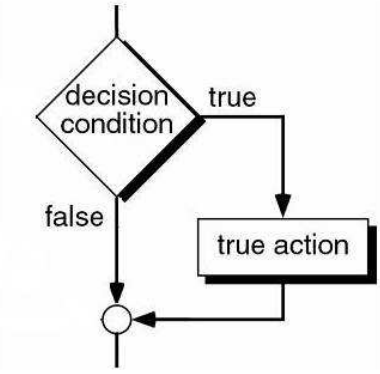
- condition : เงื่อนไขที่กำหนดขึ้น เพื่อใช้พิจารณาว่าจะทำหรือไม่ตามคำสั่ง โดยเงื่อนไขอาจจะอยู่ในรูปของนิพจน์การคำนวณ และเปรียบเทียบ หรือเป็นค่าของตัวแปรก็ได้ และจะต้องเขียนไว้ภายในเครื่องหมาย ( )
- statement : คำสั่งที่จะให้ทำงานถ้าผลการตรวจสอบเงื่อนไขออกมาเป็นจริง (true)

หรือ

```
if (condition)
{
    statement-1;
    statement-2;
    statement-3;
       …
    statement-n;
}
```

- condition : เงื่อนไขที่กำหนดขึ้น เพื่อใช้พิจารณาว่าจะทำหรือไม่ตามคำสั่ง
- statement-1, statement-2, statement-3, …, statement-n : ถ้าคำสั่งที่จะให้ทำงานมีมากกว่าหนึ่งคำสั่งให้เขียนคำสั่งทั้งหมดนั้นไว้ภายในเครื่องหมาย { }

แผนผังจำลองการทำงานของคำสั่ง if แสดงดังรูป 7.1



รูปที่ 7.1 แผนผังจำลองการทำงานของคำสั่ง if

ตัวอย่างโปรแกรมที่ 7.1

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int point;
5      printf("Enter your examination point :   ");
6      scanf("%d", &point);
7      if (point >= 50)
8          printf("You passed, congratulation");
9  }
```

จากโปรแกรมที่ 7.1 จะได้ผลการรันดังนี้

```
Enter your examination point : _
```

ถ้าคะแนนที่ป้อนเข้าไปในโปรแกรมมากกว่าหรือเท่ากับ 50 ซึ่งทำให้เงื่อนไขของคำสั่ง if เป็นจริง โปรแกรมก็จะแสดงข้อความ You passed, congratulation ขึ้นมาบนหน้าจอ

```
Enter your examination point : 80
You passed, congratulation
```

ตัวอย่างโปรแกรมที่ 7.2

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int x, y;
5      printf("Enter 2 numbers :    ");
6      scanf("%d %d" , &x, &y);
7      if (y = = 0)
8      {
9          printf("Error divided by zero.\n");
10         printf("Please try again.\n");
11     }
12 }
```

เมื่อสั่งรันโปรแกรม ทดลองป้อนจำนวนที่ทำให้เงื่อนไขของคำสั่ง if เป็นจริง โดยจำนวนที่สองให้กำหนดเป็น 0

```
Enter 2 numbers :  18    0
```

ในกรณีที่เงื่อนไขของคำสั่ง if เป็นจริงซึ่งก็คือ y = 0 โปรแกรมจะทำงานตามคำสั่งของ if ซึ่งปรากฏผลดังนี้

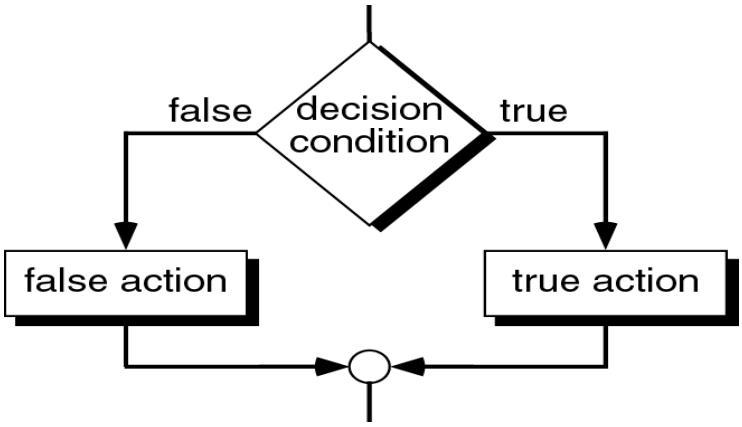
```
Error divided by zero.
Please try again.
```

7.5 คำสั่ง if-else

คำสั่ง if-else จะใช้กรณีที่มีทางเลือกให้ทำงาน 2 ทางเลือก โดยรูปแบบการเรียกใช้งานคำสั่ง if-else แสดงดังต่อไปนี้

```
if (condition)
    statement;
else
    statement;
```

การทำงานของคำสั่ง if-else จะเริ่มจากการตรวจสอบเงื่อนไข ถ้าผลออกมาจริง (true) ตัวแปรภาษาซีจะทำงานตามคำสั่งของ if แต่ถ้าผลการตรวจสอบเงื่อนไขเป็นเท็จ (false) คำสั่งของ else จะถูกทำงานแทน ดังแผนผังจำลองการทำงานของคำสั่ง if-else ดังรูปที่ 7.2



รูปที่ 7.2 แผนผังจำลองการทำงานของคำสั่ง if-else

ตัวอย่างโปรแกรมที่ 7.3

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4       int point;
5       printf("Enter your examination point  :     ");
6       scanf("%d", &point);
7       if (point >= 50)
8            printf("You passed, Congratulation\n");
9       else
10           printf("Sorry, you didn't pass\n");
11 }
```

ผลการรัน (1)

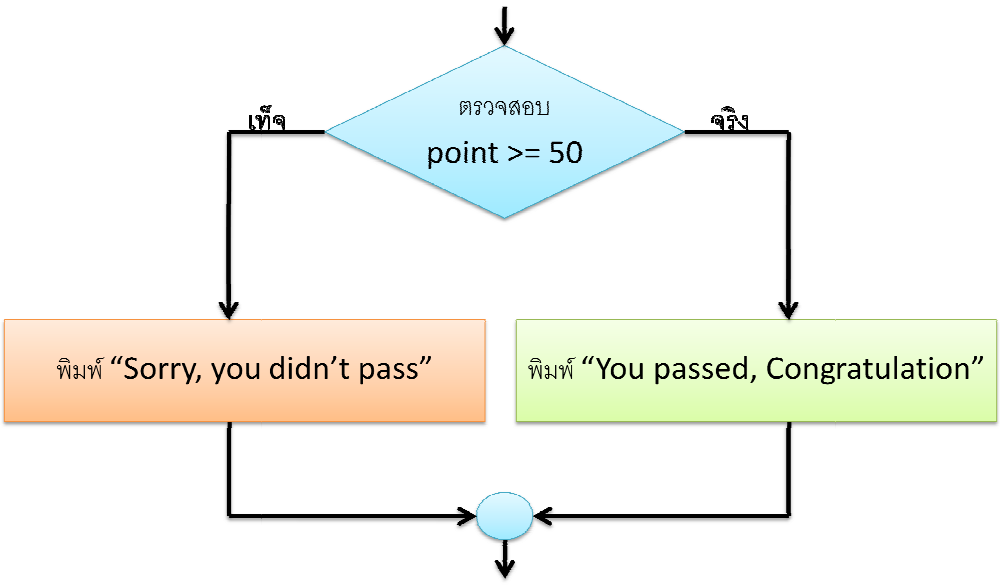
Enter your examination point : *32*

Sorry, you didn't pass

ผลการรัน (2)

Enter your examination point : *80*

You passed, Congratualation



รูปที่ 7.3 ส่วนของแผนผังการทำงานของตัวโปรแกรมที่ 7.3

จากตัวอย่างโปรแกรมที่ 7.3 แสดงการใช้งานคำสั่ง if-else โดยในส่วนของเงื่อนไขมีการเปรียบเทียบค่าในตัวแปร point ซึ่งรับมาจากผู้ใช้ผ่านทางแป้นพิมพ์ กับ ค่าคงที่ 50 ถ้าค่าในตัวแปร point มากกว่าหรือเท่ากับ 50 โปรแกรมจะแสดงคำว่า “You passed, Congratulation” แต่ถ้าค่าในตัวแปร point น้อยกว่า 50 โปรแกรมจะแสดงคำว่า “Sorry, you didn’t pass” แผนผังการทำงานของตัวอย่างโปรแกรมที่ 7.3 แสดงในรูปที่ 7.3

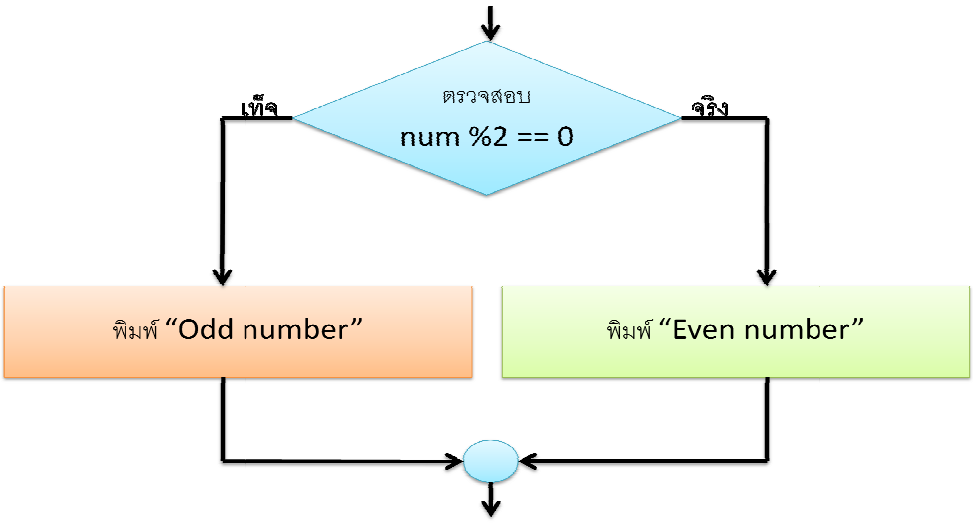
ตัวอย่างโปรแกรมที่ 7.4

```
1  #include <stdio.h>
2  int main(int  argc,  char **argv)  {
3      int num;
4      printf(“Enter number :    ”);
5      scanf(“%d”, &num);
6      if (num % 2 == 0)
7          printf(“Even number\n”);
8      else
9          printf(“Odd number\n”);
10 }
```

ผลการรัน

Enter number : ***17***

Odd number



รูปที่ 7.4 แผนผังการทำงานของตัวอย่างโปรแกรมที่ 7.4

จากตัวอย่างโปรแกรมที่ 7.4 แสดงการใช้งานคำสั่ง if-else โดยในส่วนของเงื่อนไขมีการเปรียบเทียบค่าในตัวแปร num ซึ่งรับมาจากผู้ใช้ผ่านทางแป้นพิมพ์มาหารเอาแต่เศษกับค่าคงที่ 2 กับ ค่าคงที่ 0 ถ้าค่า num % 2 เหลือเศษ 0 หมายถึงค่าในตัวแปร num เป็นเลขคู่โปรแกรมจะแสดงคำว่า “Even number” แต่ถ้าค่า num % 2 เหลือเศษ 1 หมายถึงค่าในตัวแปร num เป็นเลขคี่โปรแกรมจะแสดงคำว่า “Odd number” แผนผังการทำงานของตัวอย่างโปรแกรมที่ 7.4 แสดงในรูปที่ 7.4

ตัวอย่างโปรแกรมที่ 7.5

```
1  #include <stdio.h>
2  #define  PI  3.14156
3  int  main(int  argc,  char **argv)
4  {
5      char choice;
6      float radius, circum, area;
7      printf(“1. Circumference of the circle\n”);
8      printf(“2. Area of the circle\n”);
9      printf(“Enter your choice 1 or 2 :   ”);
10     choice = getchar();
11     printf(“Enter radius of the circle :   ”);
12     scanf(“%f”, &radius);
13     if (choice == ‘1’)  {
14         circum = 2 * PI * radius;
15         printf(“Circumference of the circle = %.2f\n”, circum);
16     }
17     else  {
18         area = PI * radius * radius;
19         printf(“Area of the circle = %.2f\n”, area);
20     }
21 }
```

ผลการรัน (1)

1. Circumference of the circle

2. Area of the circle

Enter your choice 1 or 2 : ***1***

Enter radius of the circle : ***3***

Circumference of the circle = 18.85

ผลการรัน (2)

1. Circumference of the circle

2. Area of the circle

Enter your choice 1 or 2 : ***2***

Enter radius of the circle : ***3***

Area of the circle = 28.27

ตัวอย่างโปรแกรมที่ 7.5 แสดงให้เห็นถึงการประยุกต์ใช้คำสั่งเงื่อนไขในโปรแกรม โดยเริ่มต้นโปรแกรมจะแสดงข้อความเพื่อให้เลือกการคำนวณ โดยถ้ากด '1' โปรแกรมจะทำการคำนวณเส้นรอบวงของวงกลม ถ้ากดข้อมูลอื่นที่ไม่ใช่ '1' โปรแกรมจะทำการคำนวณหาพื้นที่ของวงกลม

เมื่อผู้ใช้กดข้อความใดๆ โปรแกรมจะแสดงคำว่า "Enter radius of the circle: " เพื่อให้ผู้ใช้ป้อนค่ารัศมีของวงกลม เนื่องด้วยการคำนวณทั้ง 2 ไม่ว่าจะเป็น การหาเส้นรอบวง หรือพื้นที่ของวงกลม ก็จำเป็นจะต้องใช้ค่ารัศมีของวงกลมด้วย

บรรทัดที่ 13 จะเห็นได้ว่าโปรแกรมมีการเปรียบเทียบระหว่างค่าที่ผู้ใช้ป้อนเข้ามากับค่า '1' ซึ่งถ้าเป็นจริง หมายความว่าผู้ใช้ต้องการจะคำนวณหาเส้นรอบวงของวงกลม โปรแกรมจะทำการคำนวณตามสูตร circum = 2 * PI * radius; และแสดงค่าของเส้นรอบวงออกหน้าจอ

แต่ถ้าเงื่อนไขในบรรทัดที่ 13 เป็นเท็จ หมายความว่าผู้ใช้ต้องการคำนวณหาพื้นที่ของวงกลม ซึ่งจะคำนวณจากสูตร area = PI * radius * radius; และแสดงค่าของพื้นที่ของวงกลมออกหน้าจอ

จากผลการรันที่ 1 จะเห็นว่าผู้ใช้ป้อน 1 คือต้องการคำนวณหาขนาดเส้นรอบวง โปรแกรมจะถามถึงรัศมีของวงกลม และผู้ใช้ป้อน 3 จากการคำนวณตามสูตรจะได้ circum = 2 * 3.14156 * 3 = 18.84936 การแสดงจะปัดเศษเอา 2 ตำแหน่งดังนั้นค่าที่ได้คือ 18.85

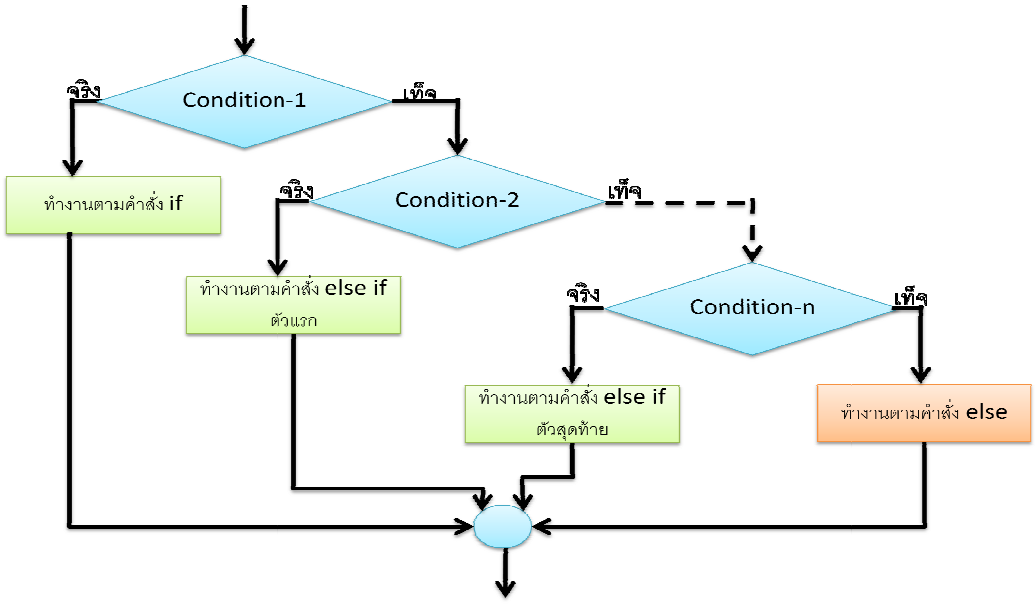
ในทางกลับกันจากผลการรันที่ 2 จะเห็นว่าผู้ใช้ป้อน 2 ก็คือต้องการหาพื้นที่วงกลม และผู้ใช้ป้อนรัศมีวงกลมคือ 3 ซึ่งการคำนวณตามสูตร area = 3.14156 * 3 * 3 = 28.27404 การแสดงปัดเศษเหลือ 2 ตำแหน่งดังนั้นค่าที่ได้คือ 28.27

7.6 คำสั่ง if-else if-else

คำสั่ง if-else if-else จะใช้ในกรณีที่มีทางเลือกให้ทำงานมากกว่า 2 ทางเลือก โดยเฉพาะทางเลือกมีเงื่อนไขต่างกัน ดังนั้น จึงต้องเรียกใช้คำสั่ง if หลายครั้ง เพื่อกำหนดเงื่อนไขสำหรับแต่ละทางเลือก โดยรูปแบบการเรียกใช้งานคำสั่ง if-else if-else แสดงดังต่อไปนี้

```
if (condition-1)
  statement;
else if (condition-2)
  statement;
…
else if (condition-n)
  statement;
else
  statement;
```

การทำงานจะเริ่มตั้งแต่ตัวแปลภาษาซี ตรวจสอบเงื่อนไขแรก ถ้าผลออกมาเป็นจริงจะทำงานตามคำสั่งของ if ถ้าผลออกมาเป็นเท็จ ตัวแปลภาษาซี จะตรวจสอบเงื่อนไขที่ 2 ซึ่งถ้าผลออกมาเป็นจริงก็จะทำงานตามคำสั่งของ else if นั้น ถ้าเป็นเท็จ ตัวแปลภาษาซี ก็จะไปตรวจสอบเงื่อนไขอื่นๆ เรียงตามลำดับต่อไปจนครบทุกเงื่อนไข ถ้าผลยังคงเป็นเท็จ ตัวแปลภาษาซี จะทำงานตามคำสั่งที่กำหนดไว้ใน else ดังแผนผังจำลองการทำงานของคำสั่ง if-else if-else ในรูปที่ 7.5



รูปที่ 7.5 แผนผังการทำงานของคำสั่ง if-else if-else

ตัวอย่างโปรแกรมที่ 7.6

```
1   #include <stdio.h>
2   int  main(int  argc,  char  **argv)  {
3        int point;
4        printf(“Enter your points :   ”);
5        scanf(“%d”, &point);
6        if ((point <= 100) && (point >= 80))
7             printf(“Grade  A\n”);
8        else if ((point < 80) && (point >= 70))
9             printf(“Grade  B\n”);
10       else if ((point < 70) && (point >= 60))
11            printf(“Grade  C\n”);
12       else if ((point < 60) && (point >= 50))
13            printf(“Grade  D\n”);
14       else
15            printf(“Grade  F\n”);
16  }
```

ผลการรัน

```
Enter your points :  79
Grade B
```

ตัวอย่างโปรแกรมที่ 7.6 แสดงให้เห็นการใช้งานของคำสั่ง if-else if-else โดยตัวโปรแกรมจะรับค่าจำนวนเต็มจากผู้ใช้ 1 ค่า นำมาเก็บไว้ในตัวแปร point จากนั้นจากเปรียบเทียบเงื่อนไขครั้งแรก อยู่ในบรรทัดที่ 7 เพื่อเปรียบเทียบค่าในตัวแปร point ว่ามีค่าอยู่ในช่วง 80 – 100 หรือไม่ ถ้าเป็นจริงให้แสดงข้อความ “Grade A” ออกทางหน้าจอ แต่ถ้าไม่เป็นจริงโปรแกรมจะทำการตรวจสอบเงื่อนไขที่ 2 นั่นก็คือค่าในตัวแปร point อยู่ช่วง 70-79 หรือไม่ถ้าใช่ให้แสดงข้อความ “Grade B” ถ้าไม่ใช่โปรแกรมก็จะทำการตรวจสอบกับเงื่อนไขที่ 3 คือค่าในตัวแปร point อยู่ในช่วง 60-69 หรือไม่ถ้าใช่แสดงข้อความ “Grade C” ถ้าไม่ใช่จะทำการตรวจสอบเงื่อนไขที่ 4 คือ point อยู่ในช่วง 50-59 หรือไม่ถ้าใช่แสดงข้อความ “Grade D” แต่ถ้าไม่ใช่โปรแกรมจะทำงานในส่วนของ else คือแสดงข้อความ “Grade F” ออกทางหน้าจอ

ตัวอย่างโปรแกรมที่ 7.7

```
1   #include <stdio.h>
2   int  main(int   argc,  char **argv)
3   {
4         char choice;
5         float    c,  f,   k;
6
7         printf("Change temperature program\n");
8         printf("1.Celsius -> Fahrenheit\n");
9         printf("2.Celsius -> Kelvin\n");
10        printf("Choose the menu 1 or 2 :    ");
11        choice = getchar();
12         if(choice == '1')
13        {
14                printf("Enter temperature in Celsius :     ");
15                scanf("%f", &c);
16                f = c / 5 * 9 + 32;
17                printf("Temperature in Farenheit = %.2f\n",f);
18        }
19        else if (choice == '2')
20        {
21                printf("Enter temperature in Celsius :     ");
22                scanf("%f", &c);
23                k = c  + 273.15;
24                printf("Temperature in Kelvin = %.2f\n", k);
25        }
26       else {
27                printf("You should type either 1 or 2\n");
28        }
29  }
```

ผลการรัน (1)

```
Change temperature program
1.Celsius -> Fahrenheit
2.Celsius -> Kelvin
Choose the menu 1 or 2 : 1
Enter temperature in Celsius :  20
Temperature in Farenheit = 68.00
```

ผลการรัน (2)

```
Change temperature program
1.Celsius -> Fahrenheit
2.Celsius -> Kelvin
Choose the menu 1 or 2 : 2
Enter temperature in Celsius :  20
Temperature in Farenheit = 293.15
```

ผลการรัน (3)

```
Change temperature program
1.Celsius -> Fahrenheit
2.Celsius -> Kelvin
Choose the menu 1 or 2 : 5
You should type either 1 or 2
```

ตัวอย่างโปรแกรมที่ 7.7 แสดงการประยุกต์ใช้งานคำสั่ง if-else if-else โดยโปรแกรมนี้จะทำการคำนวณ 2 อย่างคือ การแปลงอุณหภูมิจากองศา Celsius เป็น องศา Fahrenheit และ องศา Kelvin โดยโปรแกรม จะรอรับข้อมูลจากผู้ใช้ ถ้าผู้ใช้ป้อน 1 จะเป็นการแปลงจาก Celsius เป็น Fahrenheit หรือถ้าผู้ใช้ป้อน 2 จะเป็นการแปลงจาก Celsius เป็น Kelvin ซึ่งการเปรียบเทียบค่าของผู้ใช้นี้จะทำด้วยเงื่อนไข if ในบรรทัดที่ 12 และ else if ที่บรรทัดที่ 19 ตามลำดับ แต่ถ้าผู้ใช้ป้อนค่าอื่นที่ไม่ใช่ ค่า 1 และ 2 โปรแกรมจะไปทำงานที่คำสั่งภายใน else ซึ่งก็คือแสดงข้อความว่า “You should type either 1 or 2” โดยจะไม่มีการรับค่าองศา Celsius และการทำงานใดๆ เกิดขึ้น

7.7 คำสั่ง if ซ้อน if

คำสั่ง if ซ้อน if จะใช้กับปัญหาที่มีเงื่อนไขซับซ้อน ตัวอย่างเช่น กำหนดว่าต้องเป็นจำนวนคู่และมีค่าไม่เกิน 50 หรือเงื่อนไขต้องเป็นเพศชายอายุไม่เกิน 20 ปี เป็นต้น ซึ่งคำสั่ง if ซ้อน if ก็คือคำสั่ง if ตามปกติ แต่นำมาเขียนซ้อนกันมากกว่า 1 ชั้น ซึ่งตัวอย่างการใช้งานแสดงในตัวอย่างโปรแกรมที่ 7.8 โดยรูปแบบการเรียกใช้งานคำสั่ง if ซ้อน if แสดงดังต่อไปนี้

```
if (condition-1) {
        if (condition-2) {
                …
                if (condition-n)
                        statement;
        }
}
```

ตัวอย่างโปรแกรมที่ 7.8

```
1   #include <stdio.h>
2   int main(int argc, char **argv) {
3       int num;
4       printf("Enter your number :  ");
5       scanf("%d", &num);
6       if (num >= 30)  {
7           if (num <= 40)  {
8               if (num % 2 == 1)  printf("Right number\n");
9               else  printf("Wrong number\n");
10          }  else  printf("Large number\n");
11      }  else  printf("Small number\n");
12  }
```

ผลการรัน

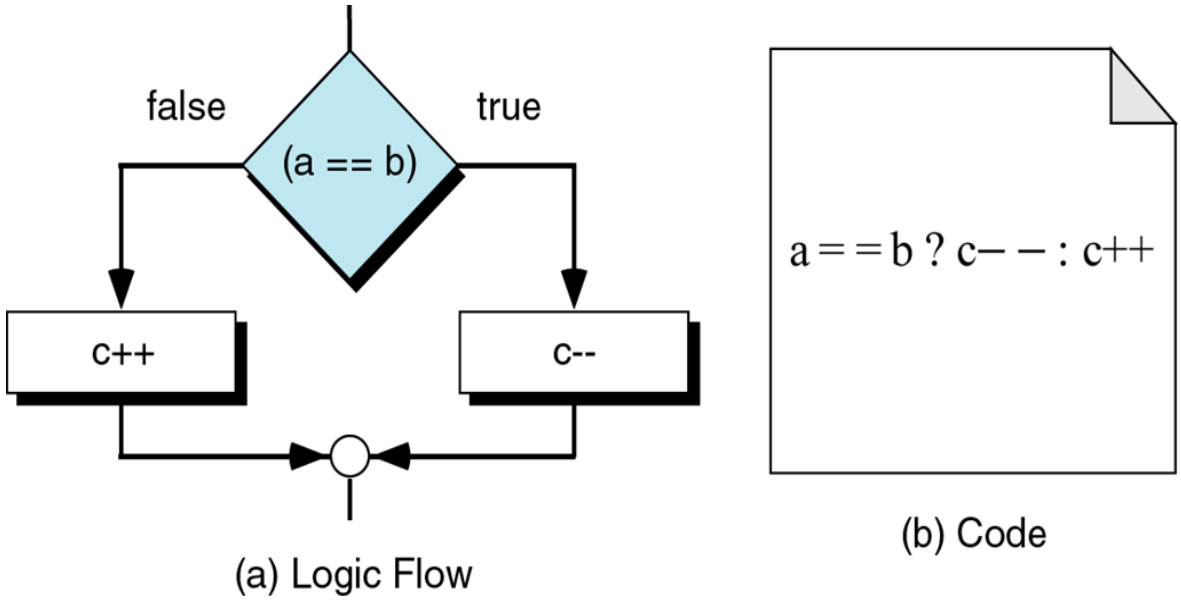
```
Enter your number : 38
Wrong number
```

7.8 ตัวดำเนินการเงื่อนไขแบบย่อ

ในภาษาซีนอกจากคำสั่ง if , if-else, และ if-else if-else แล้วยังสามารถเขียนอยู่ในรูปของตัวดำเนินการย่อได้อีกด้วย ซึ่งจะมีรูปแบบดังนี้

condition ? expr1 : expr2
• condition : เงื่อนไขที่จะทำการเปรียบเทียบ • expr1 : ชุดคำสั่งที่จะทำถ้าเงื่อนไขเป็นจริง • expr2 : ชุดคำสั่งที่จะทำถ้าเงื่อนไขเป็นเท็จ

รูปที่ 7.6 แสดงแผนผังการทำงานของตัวดำเนินการเงื่อนไขแบบย่อและตัวอย่างโปรแกรมอย่างง่ายโดยเงื่อนไขคือการเปรียบเทียบค่าในตัวแปร a และตัวแปร b ถ้าค่าเท่ากันโปรแกรมจะทำคำสั่งหลังเครื่องหมาย “?” คือ c-- แต่ถ้าค่าในตัวแปรทั้งสองตัวไม่เท่ากันโปรแกรมจะทำคำสั่งหลังเครื่องหมาย “:” ซึ่งในโปรแกรมนี้คือ c++



รูปที่ 7.6 แผนผังและตัวอย่างโปรแกรมตัวดำเนินการเงื่อนไขแบบย่อ

ตัวอย่างโปรแกรมที่ 7.9

```
1  #include <stdio.h>
2  int main(int argc, char **argv) {
3      float total, discount;
4      printf("Enter total payment : ");
5      scanf("%f", &total);
6      if( total > 2000)
7          discount = total * 0.1;
8      else
9          discount  = 0;
10     printf("Discount = %.2f\n", discount);
11 }
```

ผลการรัน (1)

Enter total payment : ***100***

Discount = 0.00

ผลการรัน (2)

Enter total payment : ***3500***

Discount = 350.00

ตัวอย่างโปรแกรมที่ 7.9 เป็นโปรแกรมที่ใช้คำสั่งเงื่อนไข if-else เพื่อหาส่วนลดโดยถ้าผู้ใช้ป้อนตัวเลขที่มากกว่า 2000 จะได้ส่วนลด 10% แต่ถ้าผู้ใช้ป้อนตัวเลขน้อยกว่าหรือเท่ากับ 2000 จะไม่มีส่วนลด ซึ่งจะสามารถเขียนลดรูปโดยใช้ตัวดำเนินการเงื่อนไขแบบย่อได้ดังตัวอย่างโปรแกรมที่ 7.10

ตัวอย่างโปรแกรมที่ 7.10

```
1  #include <stdio.h>
2  int main(int argc, char **argv) {
3      float total, discount;
4      printf("Enter total payment : ");
5      scanf("%f", &total);
6      printf("Discount = %.2f\n", ( total > 2000)?  total *0.1 : 0);
7  }
```

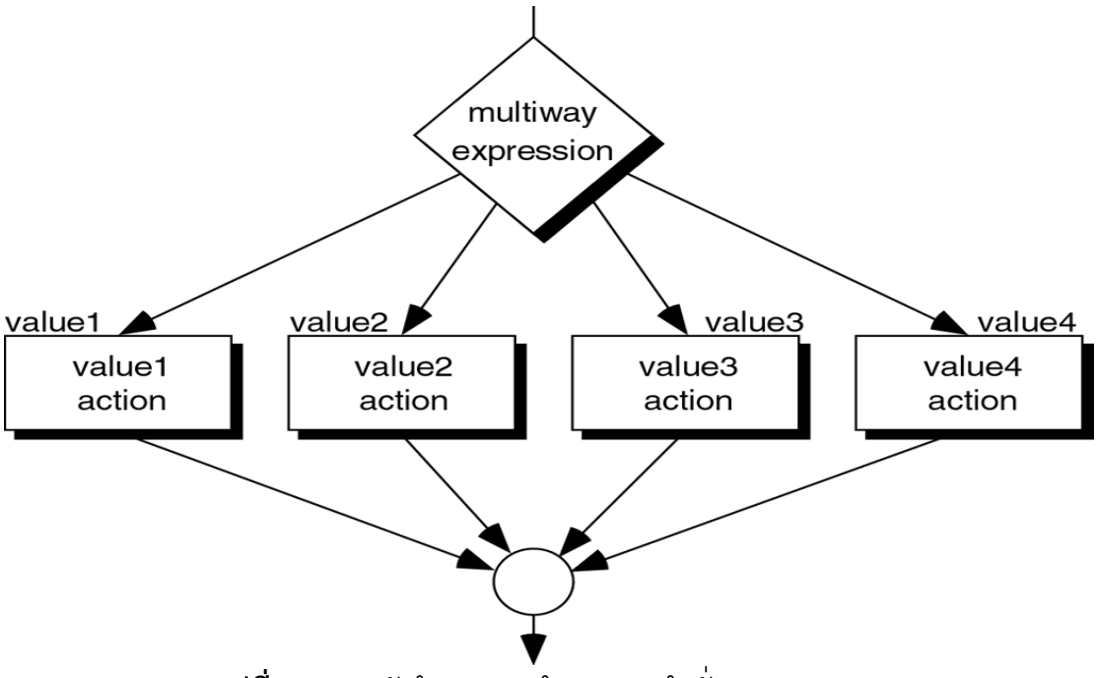
7.9 คำสั่ง switch-case

คำสั่ง switch-case จะใช้ในกรณีที่มีทางเลือกให้ทำงานหลายทางเลือกโดยใช้เงื่อนไขร่วมกัน ซึ่งตัวแปลภาษาซี จะตรวจสอบเงื่อนไขเพียงครั้งเดียว ผลจากการตรวจสอบเงื่อนไขจะถูกนำไปพิจารณาอีกครั้งว่าจะทำงานตามทางเลือกใด รูปแบบของคำสั่ง switch แสดงดังต่อไปนี้

```
switch (variable)
{
    case constant-1 :
        statement;
        break;
    case constant-2 :
        statement;
        break;
          …
    case constant-n :
        statement;
        break;
    default :
         statement;
}
```

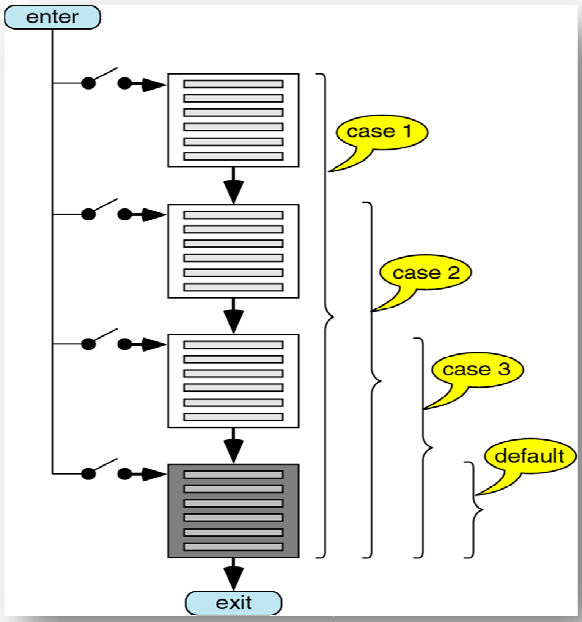
- variable : ตัวแปรชนิด int หรือ char หรืออาจจะเป็นนิพจน์การคำนวณที่ให้ผลออกมาเป็น int หรือ char ก็ได้ โดยตัวแปรนี้จะถูกใช้เป็นเงื่อนไขในการเลือกทำงานต่อไป
- constant-1,constant-2,… , constant-n : ค่าคงที่ชนิด int หรือ char โดยต้องเป็นชนิดเดียวกับค่าของตัวแปร variable ถ้าค่าของ variable ตรงกับค่า constant ของ case ใด โปรแกรมจะทำงานตามคำสั่งของ case นั้น
- break : คำสั่งสำหรับออกการทำงานของ switch…case ซึ่งควรจะใส่ไว้ตอนท้ายของทุก case เพื่อให้โปรแกรมออกจากคำสั่ง switch ทันที หลังจากจบการทำงานตาม case ที่เงื่อนไขตรงกัน
- default : ถ้าค่าของตัวแปร variable ไม่ตรงกับค่าของ constant ใน case ใดๆ เลย โปรแกรมต้องเข้ามาทำงานตามคำสั่งของ default ก่อนออกจากคำสั่ง switch…case

แผนผังจำลองการทำงานของคำสั่ง switch-case แสดงดังรูปที่ 7.7



รูปที่ 7.7 แผนผังจำลองการทำงานของคำสั่ง switch-case

```
switch (expression)
  {
  case constant-1: statement
                   …
                   statement
  case constant-2: statement
                   …
                   statement
  case constant-n: statement
                   …
                   statement
  default          : statement
                   …
                   statement
  } /* end switch */
```

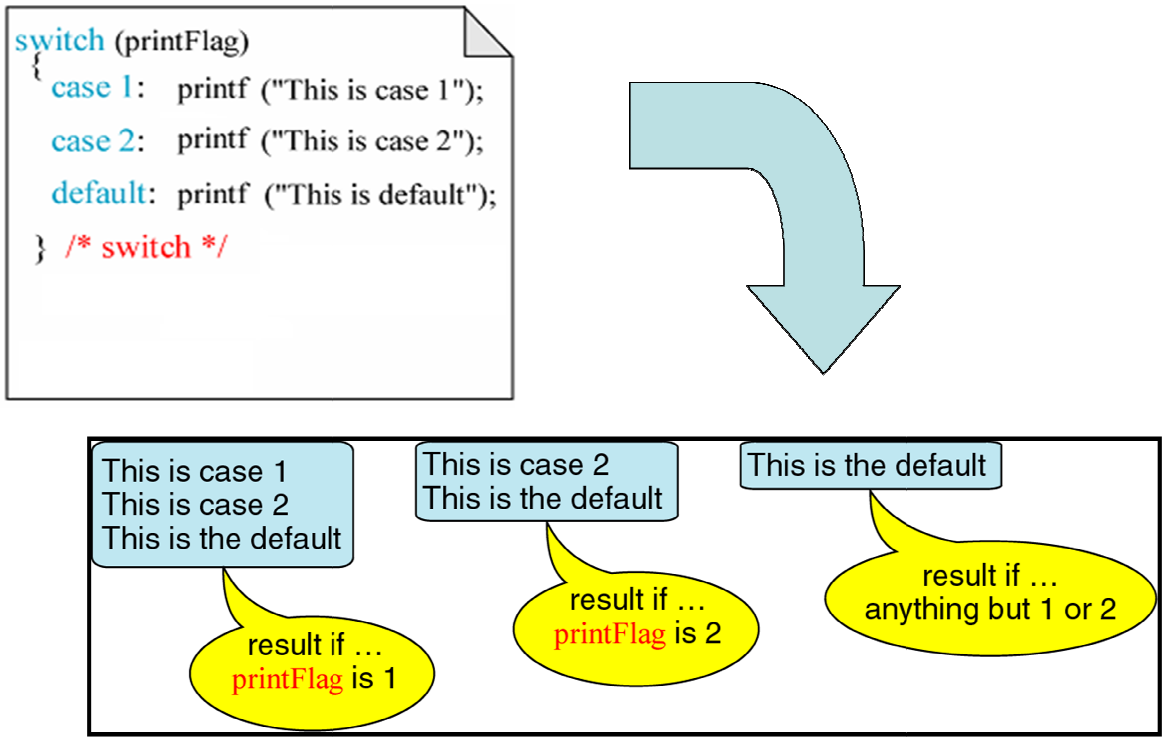


(a) (b)

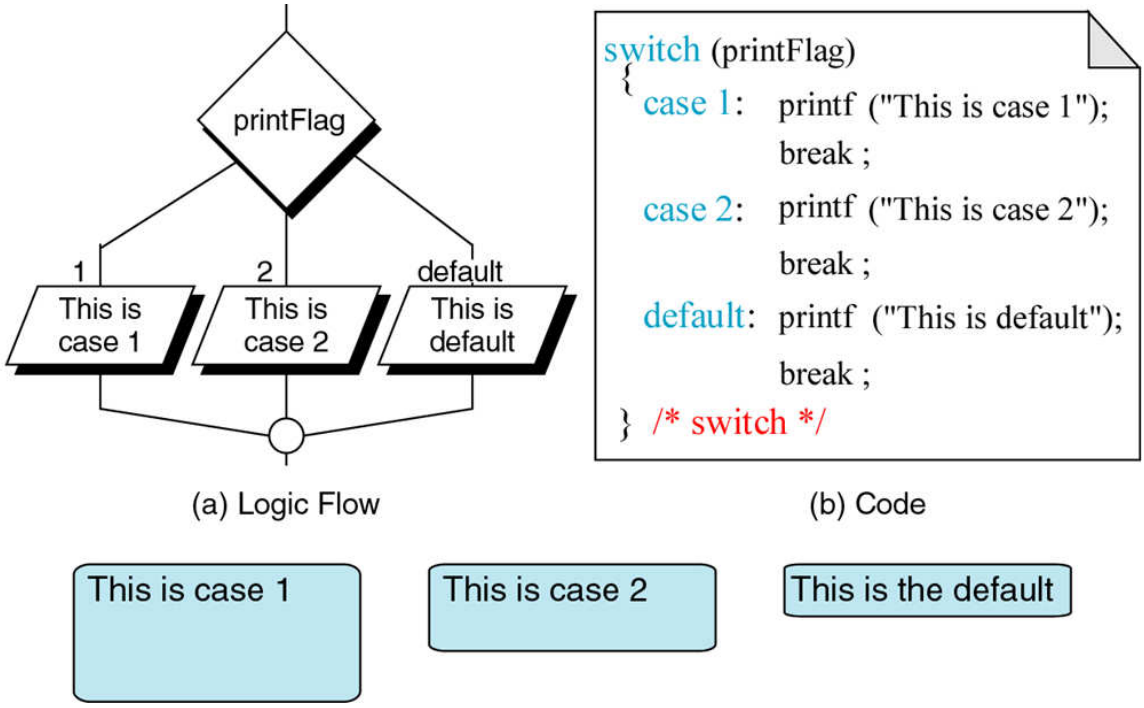
รูปที่ 7.8 ลักษณะการทำงานของคำสั่ง switch-case

จากรูปที่ 7.8 แสดงให้เห็นถึงลักษณะการทำงานของคำสั่ง switch-case จากรูปที่ 7.8(a) แสดงคำสั่งที่ใช้ในโปรแกรม ส่วนของ expression หลัง switch จะเป็นชื่อตัวแปร หรือนิพจน์ที่เราต้องการเปรียบเทียบกับค่า constant ต่างๆ หลังคำว่า case จากรูปที่ 7.8(b) จะเห็นว่าการทำงานของคำสั่ง switch-case จะทำงานในลักษณะลำดับขั้นซึ่งหมายความว่า ถ้าค่าในส่วนของ expression เท่ากับ constant-1 โปรแกรมจะทำให้ส่วนคำสั่งภายใน case นั้นและทำคำสั่งในส่วนของ case อื่นๆ ที่อยู่ข้างใต้ด้วยโดยไม่เปรียบเทียบเงื่อนไขใดๆ เช่นจากรูปจะเห็นว่า switch-case มี case ทั้งหมด 3 case และมี default อยู่ด้วย ถ้าการเปรียบเทียบตรงกับ case-1 โปรแกรมจะทำชุดคำสั่งใน case-1 ถึง default , ถ้าการเปรียบเทียบตรงกับ case-3 โปรแกรมจะทำชุดคำสั่งใน case-3 และ default ดังแสดงตัวอย่างในรูปที่ 7.9

- ถ้า printFlag มีค่า 1 โปรแกรมจะแสดงข้อความ “This is case1”, “This is case 2” และ “This is default” ออกทางหน้าจอ
- ถ้า printFlag มีค่า 2 โปรแกรมจะแสดงข้อความ “This is case 2” และ “This is default” ออกทางหน้าจอ
- ถ้า printFlag มีค่าอื่นที่ไม่ใช่ 1 และ 2 โปรแกรมจะแสดงข้อความ “This is default” ออกทางหน้าจอ



รูปที่ 7.9 ผลการรันจากการใช้คำสั่ง switch-case



รูปที่ 7.10 แผนผังการทำงานของ switch-case และคำสั่ง break

จากตัวอย่างการใช้งานคำสั่ง switch-case ในรูปที่ 7.9 อาจจะไม่ตรงกับจุดมุ่งหมาย ในการใช้งานคำสั่ง switch-case มากนัก เนื่องจากปกติเมื่อมีเงื่อนไขเราต้องการจะให้โปรแกรม ทำชุดคำสั่งเฉพาะชุดคำสั่งที่ตรงกับการเปรียบเทียบของเงื่อนไขเท่านั้น ภาษาซีจึงมีคำสั่งอยู่ 1 คำสั่งคือ break เพื่อเป็นการหยุดการทำงานที่ต่อเนื่องของ switch-case คำสั่ง break นิยมใส่หลังชุดคำสั่งที่ต้องการให้ทำภายใน case และก่อนการเริ่มต้นของ case ถัดไป ดังตัวอย่างในรูปที่ 7.10 ซึ่งจะคล้ายกับโปรแกรมในรูปที่ 7.9 แต่จะมีการใช้คำสั่ง break ก่อนขึ้น case ถัดไป ผลการทำงานในส่วนของโปรแกรมดังกล่าวจะเป็นดังนี้

- ถ้า printFlag มีค่าเป็น 1 โปรแกรมจะแสดงคำว่า “This is case 1” เท่านั้น
- ถ้า printFlag มีค่าเป็น 2 โปรแกรมจะแสดงคำว่า “This is case 2” เท่านั้น
- ถ้า printFlag ไม่มีค่าเป็น 1 หรือ 2 โปรแกรมจะแสดงคำว่า “This is the default”

จะเห็นว่าการทำงานจะคล้ายกันกับการทำงานแบบ if-else if-else ที่ได้อธิบายมาแล้ว และในภาษาซีการทำงานแบบ switch-case จะมีความเร็วในการทำงานมากกว่าการใช้งาน if-else if-else ดังนั้นถ้าสามารถประยุกต์โปรแกรมให้ใช้ switch-case ได้จะดีมาก

ตัวอย่างโปรแกรมที่ 7.11

```
1   #include <stido.h>
2   int  main(int  argc,  char  **argv)  {
3       int choice;
4       printf(“1.LCD Monitor\n”);
5       printf(“2.Optical Mouse\n”);
6       printf(“3.Wireless Keyboard\n”);
7       printf(“4.Notebook\n”);
8       printf(“5.Digital Camera\n”);
9       printf(“Enter your choice :    ”);      scanf(“%d”, &choice);
10      switch (choice)   {
11          case 1 :
12                  printf(“LCD Monitor : 15,000 baht.\n”);
13                  break;
14          case 2 :
15                  printf(“Optical Mouse : 800 baht.\n”);
16                  break;
17          case 3 :
18                  printf(“Wireless Keyboard : 700 baht.\n”);
19                  break;
20          case 4 :
21                  printf(“Notebook : 32,000 baht.\n”);
22                  break;
23          case 5 :
24                  printf(“Digital Camera : 9,500 baht.\n”);
25                  break;
26          default :
27                  printf(“Thank  you.\n”);
28      }
29  }
```

ผลการรัน(1)

1.LCD Monitor

2.Optical Mouse

3.Wireless Keyboard

4.Notebook

5. Digital Camera

Enter your choice : ***1***

LCD Monitor : 15,000 baht.

ผลการรัน(2)

1.LCD Monitor

2.Optical Mouse

3.Wireless Keyboard

4.Notebook

5. Digital Camera

Enter your choice : ***4***

Notebook : 32,000 baht

ผลการรัน(3)

1.LCD Monitor

2.Optical Mouse

3.Wireless Keyboard

4.Notebook

5. Digital Camera

Enter your choice : ***20***

Thank you.

จากตัวอย่างโปรแกรมที่ 7.11 จะมีการรับค่าจำนวนเต็มจากผู้ใช้ผ่านทางแป้นพิมพ์มาเก็บยังตัวแปรชื่อ choice ซึ่งในบรรทัดที่ 10 จะผ่านค่าในตัวแปร choice ให้กับคำสั่ง switch เพื่อเปรียบเทียบหา case ที่ตรงกับค่าของ choice จากผลการรันที่ 2 จะเห็นได้ว่าถ้าผู้ใช้ป้อนค่า 4 โปรแกรมจะเข้าไปใน case 4 แล้วแสดงคำว่า “Notebook : 32,000 baht” จากนั้นเจอคำสั่ง break จึงสิ้นสุดการทำงานของ switch-case แต่ถ้าผู้ใช้ป้อนค่าที่ไม่ตรงกับ case ใดๆ ดังผลการรันที่ 3 โปรแกรมจะข้ามไปทำงานในส่วนของ default จากตัวอย่างจะแสดงคำว่า “Thank you” ออกทางหน้าจอ

ตัวอย่างโปรแกรมที่ 7.12

```
1   #include <stdio.h>
2   int  main(int  argc,  char  **argv)
3   {
4         char grade;
5         printf(“Enter your grade : ”);
6         scanf(“%c”,  &grade);
7         switch(grade)
8         {
9               case ‘A’ :
10                    printf(“100-80 points.\n”);
11                    break;
12              case ‘B’ :
13                    printf(“79-70 points.\n”);
14                    break;
15              case ‘C’ :
16                    printf(“69-60 points.\n”);
17                    break;
18              case ‘D’ :
19                    printf(“59-50 points.\n”);
20                    break;
21              default :
22                    printf(“49-0 points.\n”);
23        }
24  }
```

ผลการรัน

Enter your grade : ***B***

79-70 points.

ตัวอย่างโปรแกรมที่ 7.13

```
1   #include <stdio.h>
2   int  main(int  argc,  char  **argv)
3   {
4         char grade;
5         printf(“Enter your grade : ”);
6         scanf(“%c”,  &grade);
7         switch(grade)
8         {
9               case ‘A’ :
10                    printf(“100-80 points.\n”);
11              case ‘B’ :
12                    printf(“79-70 points.\n”);
13              case ‘C’ :
14                    printf(“69-60 points.\n”);
15              case ‘D’ :
16                    printf(“59-50 points.\n”);
17              default :
18                    printf(“49-0 points.\n”);
19        }
20  }
```

ผลการรัน

```
Enter your grade : B
79-70 points.
69-60 points.
59-50 points.
49-0 points.
```

ตัวอย่างโปรแกรมที่ 7.12 และ 7.13 แสดงตัวอย่างของโปรแกรมที่ทำงานในลักษณะเดียวกัน คือ มีการรับค่าจากผู้ใช้เป็นตัวอักษรมาเก็บไว้ในตัวแปร grade โดยตัว switch จะมีการเปรียบเทียบค่าตัวอักษรที่อยู่ในตัวแปร grade กับ case ต่างๆ คือ 'A', 'B', 'C', 'D' และค่าอื่นๆที่ไม่ใช่ 4 ตัวอักษรเหล่านี้ ถ้าค่าที่อยู่ในตัวแปร grade ตรงกับ case ใด โปรแกรมจะพิมพ์ช่วงคะแนนที่จะทำให้ได้เกรดนั้นๆ ถ้าดูจากผลของการรันของตัวอย่างโปรแกรมที่ 7.12 จะเห็นว่า เมื่อผู้ใช้ป้อนตัวอักษร 'B' โปรแกรมจะพิมพ์ค่าช่วงคะแนนได้อย่างถูกต้องคือ "79-70 points" แล้วก็จบการทำงานของโปรแกรม เนื่องจากในทุก case มีการใช้คำสั่ง break ปิดท้ายทำให้โปรแกรมไม่ทำงานชุดคำสั่งภายในของ case อื่น ในทางกลับกันในตัวอย่างโปรแกรมที่ 7.13 ถ้าผู้พัฒนาโปรแกรมลืมใส่คำสั่ง break ผลการรันที่ได้รับจะไม่ตรงกับจุดประสงค์และความต้องการของโปรแกรม

บทที่ 8
ควบคุมทิศทางแบบวนรอบ

การควบคุมทิศทางแบบวนรอบ (Iteration) หรือที่เรียกกันว่า การทำงานแบบวนลูป (Loop) คือ การที่เขียนโปรแกรมให้วนรอบทำงานซ้ำคำสั่งเดิม โดยมีเงื่อนไขเพื่อให้โปรแกรมวนรอบการทำงาน คำสั่งในภาษาซีที่ใช้สำหรับควบคุมทิศทางแบบวนรอบ ได้แก่ while, do-while และ for

8.1 คำสั่ง while

คำสั่งวนลูปแบบ while จะเริ่มต้นทำงานจากการตรวจสอบเงื่อนไข ถ้าเงื่อนไขเป็นจริงจึงจะทำงานตามคำสั่งของ while เมื่อทำงานเสร็จแล้วก็จะวนกลับไปตรวจสอบเงื่อนไขใหม่ เป็นเช่นนี้ไปเรื่อยๆ จนกว่าเงื่อนไขจะเป็นเท็จจึงจะหลุดออกจากการทำงานของลูป while รูปแบบการเขียนคำสั่ง while มีด้วยกัน 2 วิธี คือแบบมีคำสั่งภายในคำสั่งเดียว และแบบมีคำสั่งภายในหลายคำสั่ง ดังแสดงต่อไปนี้

```
while (condition)
{
    statement-1;
    statement-2;
        …
    statement-n;
}
```

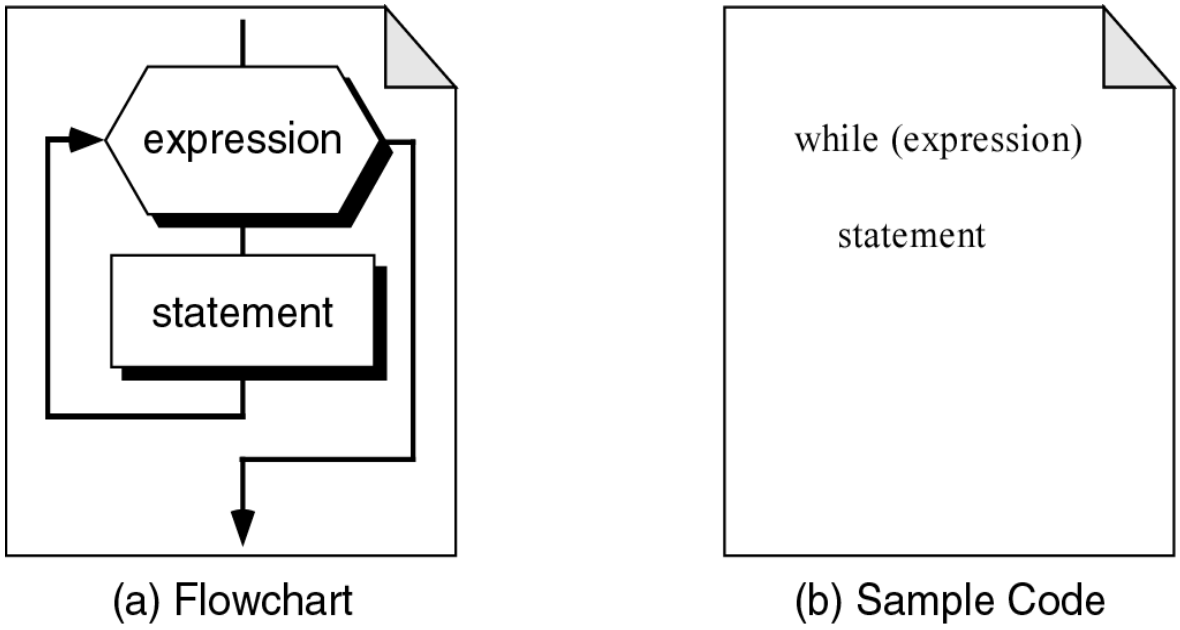
- Condition : เงื่อนไขที่จะให้วนทำงานในลูป
- statement-1, statement-2, statement-n : คำสั่งที่จะให้ทำงาน ถ้าผลการตรวจสอบเงื่อนไขออกมาเป็นจริง (true)

และ

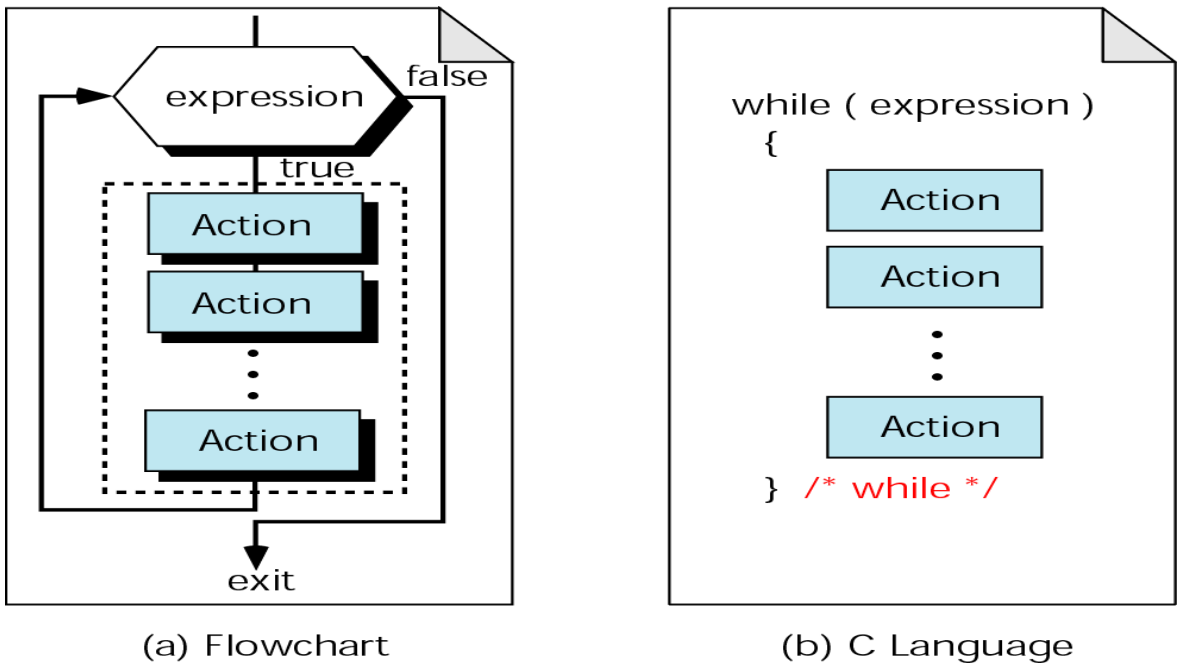
```
while (condition)
    statement;
```

- Condition : เงื่อนไขที่จะให้วนทำงานในลูป
- statement : คำสั่งที่จะให้ทำงาน ถ้าผลการตรวจสอบเงื่อนไขออกมาเป็นจริง (true)

แผนผังแสดงการทำงานของลูป while ทั้ง 2 แบบแสดงดังรูปที่ 8.1 และ รูปที่ 8.2



รูปที่ 8.1 การทำงานของ while แบบคำสั่งภายในคำสั่งเดียว (a) แผนผังการทำงาน (b) ตัวอย่างโปรแกรม



รูปที่ 8.2 การทำงานของ while แบบคำสั่งภายในหลายคำสั่ง (a) แผนผังการทำงาน (b) ตัวอย่างโปรแกรม

ตัวอย่างโปรแกรมที่ 8.1

```
1  #include <stdio.h>
2  int  main(int  argc,  char  **argv)
3  {
4      int count = 0;
5      printf("Show number from zero to ten.\n");
6      while (count <= 10)
7      {
8          printf("%d ", count);
9          count++;
10     }
11 }
```

ผลการรัน

```
Show number from zero to ten.
0  1  2  3  4  5  6  7  8  9  10
```

ตัวอย่างโปรแกรมที่ 8.1 แสดงการใช้งานคำสั่ง while( ) โดยมีการประกาศตัวแปรประเภทจำนวนเต็มชื่อ count ตัวแปรนี้จะถูกใช้ในการตรวจสอบเงื่อนไขการทำงานเสร็จสิ้นของการวนลูป ตัวแปร count มีการกำหนดค่าเริ่มต้นให้คือ 0 จากนั้นการวนลูปจะอยู่ในบรรทัดที่ 6 - 10 โดยบรรทัดที่ 6 เป็นการตรวจสอบเงื่อนไข count ที่มีค่าเริ่มต้นเป็น 0 จะน้อยกว่าหรือเท่ากับ 10 ดังนั้นเงื่อนไขจึงเป็นจริง เมื่อเงื่อนไขเป็นจริง โปรแกรมจะทำงานคำสั่งที่อยู่ใน while( ) ซึ่งก็คือในบรรทัดที่ 8 - 9 จะแสดงผลค่าของตัวแปร count ออกทางหน้าจอ และเพิ่มค่าให้กับตัวแปร count ไปอีก 1 เมื่อถึงบรรทัดที่ 10 โปรแกรมจะกระโดดกลับไปที่บรรทัดที่ 6 อีกครั้งเพื่อตรวจสอบเงื่อนไข โปรแกรมจะทำงานซ้ำในลักษณะนี้ต่อไปเรื่อยๆ จนกระทั่งค่าในตัวแปร count มีค่าเท่ากับ 11 จึงจะทำให้เงื่อนไขของ while( ) เป็นเท็จ และโปรแกรมจะกระโดดมายังบรรทัดที่ 11 ซึ่งเป็นการสิ้นสุดของโปรแกรม

ข้อควรระวังในการใช้คำสั่งลูปแบบ while คือ ภายในลูปจะต้องมีหนึ่งคำสั่งในการเพิ่ม ลด หรือเปลี่ยนแปลงค่าของตัวแปรที่ใช้ในการตรวจสอบเงื่อนไข เพื่อให้เงื่อนไขเข้าใกล้จุดที่เป็นเท็จ ลูปจะได้มีจุดจบ เพราะไม่เช่นนั้นจะกลายเป็นลูปที่ไม่มีวันจบ โปรแกรมจะทำงานไม่หยุด และไม่สามารถทำงานในคำสั่งต่อจากลูป while ได้

ตัวอย่างโปรแกรมที่ 8.2

```
1   #include <stdio.h>
2   int  main(int  argc,  char **argv)
3   {
4       int   begin = 0, sum = 0, end;
5       printf("Enter end number :  ");
6       scanf("%d", &end);
7       while(begin <= end)
8       {
9               sum = sum + begin;
10              begin++;
11      }
12      printf("Sum = %d", sum);
13  }
```

ผลการรัน

Enter end number : ***10***

Sum = 55

ตัวอย่างโปรแกรมที่ 8.2 แสดงการใช้งานคำสั่ง while( ) โดยมีการประกาศตัวแปรประเภทจำนวนเต็มชื่อ begin มีค่าเริ่มต้นคือ 0,sum มีค่าเริ่มต้นเป็น 0 และ end ตัวโปรแกรมจะแสดงคำว่า "Enter end number : " ออกทางหน้าจอผ่านฟังก์ชัน printf( ) ในบรรทัดที่ 5 จากนั้นโปรแกรมจะรอรับค่าจากผู้ใช้แล้วเก็บค่าที่รับได้ไว้ในตัวแปร end ด้วยฟังก์ชัน scanf( ) ในบรรทัดที่ 6 สำหรับบรรทัดที่ 7 - 11 จะเป็นการเขียนลูปเพื่อทำการทำซ้ำโดยเงื่อนไขก็คือ begin <= end (ค่าในตัวแปร begin น้อยกว่าหรือเท่ากับ end) จากตัวอย่างผลการรันถ้ากำหนดให้ผู้ใช้ป้อนค่า 10 เข้าโปรแกรม เงื่อนไขใน while( ) ครั้งแรกจะเป็นจริงเนื่องจาก 0 <= 10 ตัวโปรแกรมจะทำงานต่อในบรรทัดที่ 9 -10 คือการนำค่าในตัวแปร begin ไปบวกกับค่าในตัวแปร sum แล้วเก็บไว้ในตัวแปร sum จากนั้นเพิ่มค่าในตัวแปร begin ไปอีก 1 ค่า ซึ่งการทำซ้ำจะทำซ้ำจนกระทั่งเงื่อนไขใน while( ) เป็นเท็จก็คือ 11 <= 10 ดังนั้นถ้าที่เก็บอยู่ในตัวแปร sum เมื่อสิ้นสุดโปรแกรมคือ 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7+ 8 + 9 + 10 = 55 ดังนั้นในบรรทัดที่ 12 โปรแกรมจึงแสดง Sum = 55 ออกทางหน้าจอ

8.2 คำสั่ง do-while

คำสั่งวนลูปแบบ do-while จะคล้ายกับ while แตกต่างกันตรงที่ ลูป do-while จะทำงานตามคำสั่งของ do ก่อนหนึ่งรอบ จากนั้นจึงตรวจสอบเงื่อนไขที่ while ถ้าเงื่อนไขเป็นจริงจะวนกลับไปทำงานตามคำสั่งของ do อีกครั้ง แล้วกลับมาตรวจสอบเงื่อนไขใหม่ และจะเป็นอย่างนี้ไปจนกว่าผลการตรวจสอบเงื่อนไขจะเป็นเท็จ จึงเลิกทำงาน ออกจากลูป do-while รูปแบบการเขียนคำสั่ง do-while สามารถแสดงได้ดังต่อไปนี้

```
do
{
    statement-1;
    statement-2;
    statement-3;
        …
    statement-n;
}
while (condition);
```

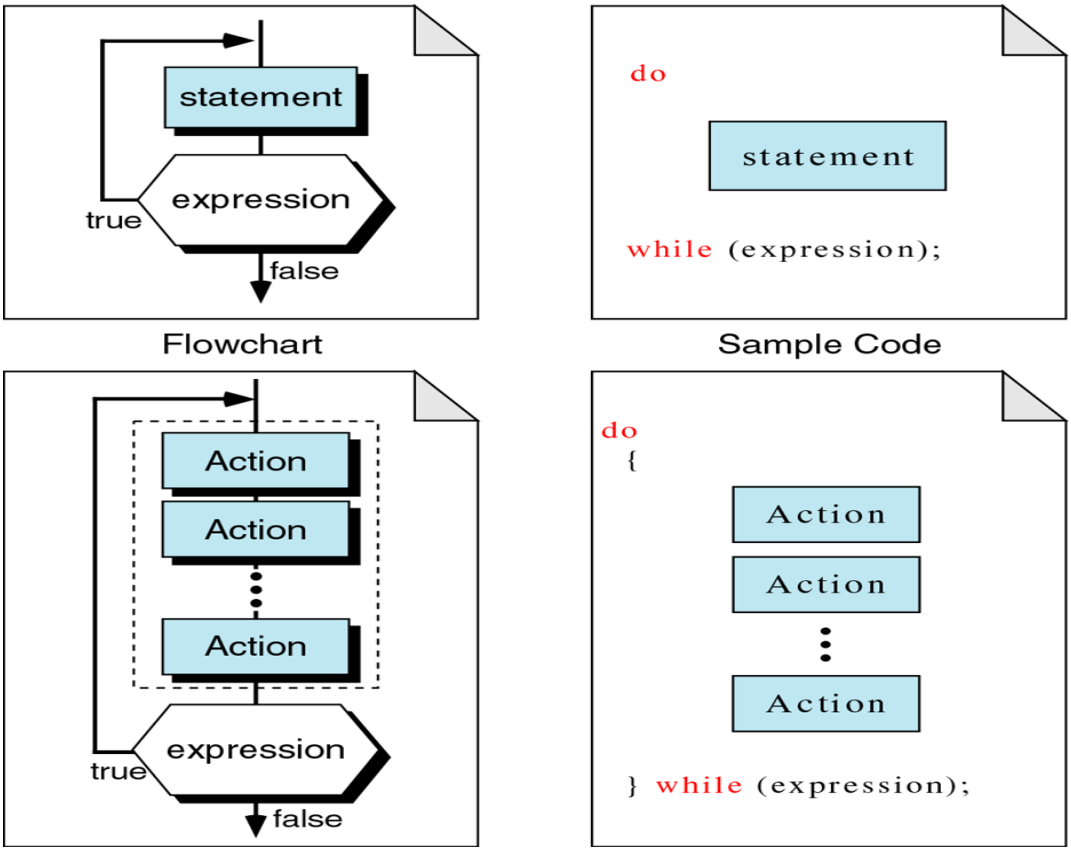
- statement-1, statement-2, statement-3 : คำสั่งที่จะให้ทำงาน ถ้าผลการตรวจสอบเงื่อนไขออกมาเป็นจริง (true)
- condition : เงื่อนไขที่จะให้วนกลับไปทำงานในลูป

หรือ

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

- statement : คำสั่งที่จะให้ทำงาน ถ้าผลการตรวจสอบเงื่อนไขออกมาเป็นจริง (true)
- condition : เงื่อนไขที่จะให้วนกลับไปทำงานในลูป

แผนผังแสดงการทำงานของลูป do-while และโปรแกรมตัวอย่างแสดงดังรูปที่ 8.3



รูปที่ 8.3 แผนผังการทำงานของลูป do-while

ตัวอย่างโปรแกรมที่ 8.3

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int  num = -13;
5      do {
6          printf("%d  ",  num);
7          num--;
8      } while(num >= 0);
9  }
```

ผลการรัน

-13

ตัวอย่างโปรแกรมที่ 8.3 แสดงให้เห็นว่าเมื่อมีการใช้คำสั่ง do-while( ) จะมีการทำชุดคำสั่งครั้งแรกก่อนการตรวจสอบเงื่อนไข

ตัวอย่างโปรแกรมที่ 8.4

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int  num = 10;
5      do {
6          printf("%d  ",  num);
7          num--;
8      } while(num >= 0);
9  }
```

ผลการรัน

```
10  9  8  7  6  5  4  3  2  1  0
```

ตัวอย่างโปรแกรมที่ 8.4 เป็นการทำซ้ำโดยการกำหนดค่าเริ่มต้นให้กับตัวแปร num เป็น 10 แล้วใช้คำสั่ง do-while( ) เพื่อแสดงค่าที่เก็บอยู่ในตัวแปร num ออกทางหน้าจอและลดค่าในตัวแปร num ลงไป 1 ค่า ทำซ้ำจนกระทั่งค่าในตัวแปรเป็น -1 จึงหลุดจากการทำซ้ำ

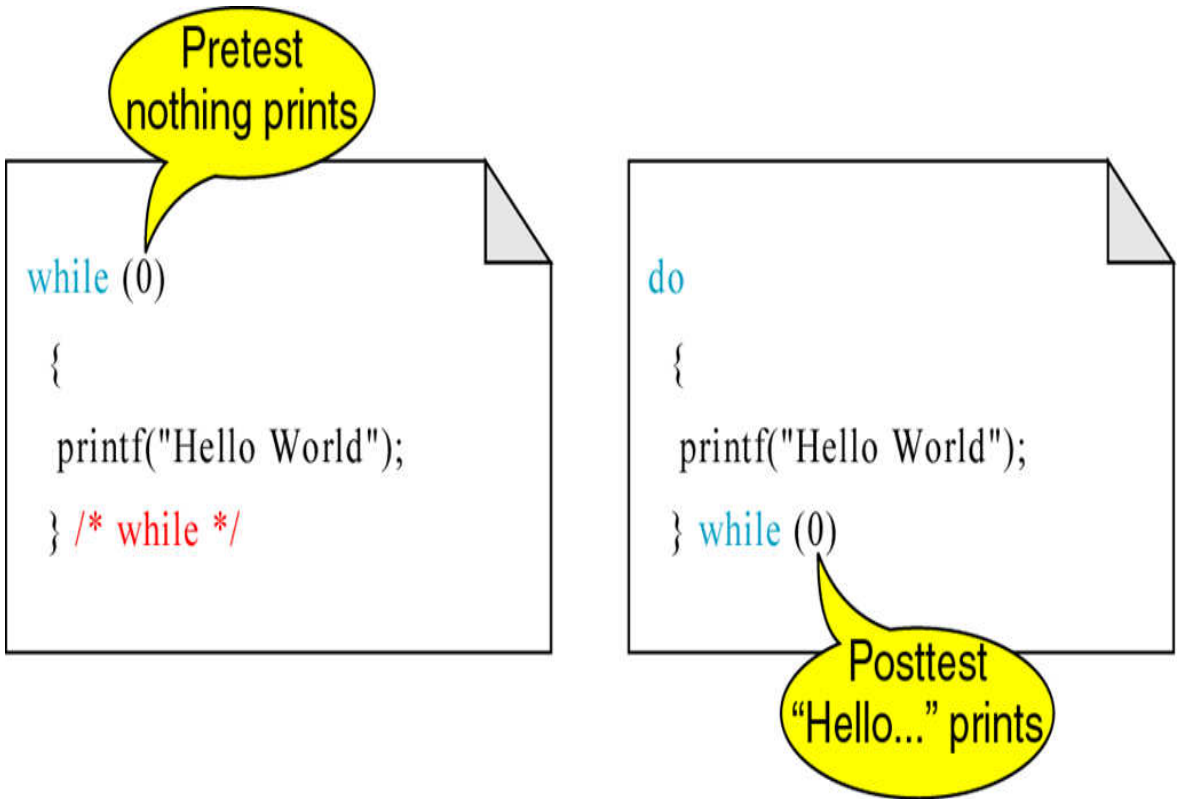
ตัวอย่างโปรแกรมที่ 8.5

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)  {
3      float   sum = 0.0;  int k = 1;
4      do  {
5          sum = sum + k;
6          k++;
7      }  while(k <= 10);
8      printf("Average = %.4f\n", sum/10);
9  }
```

ผลการรัน

```
Average = 55.0000
```

ตัวอย่างโปรแกรมที่ 8.5 เป็นการหาค่าเฉลี่ยของตัวเลข 1 ถึง 10 ในแต่ละรอบค่าของตัวแปร k จะเพิ่มขึ้นอีก 1 แต่จะมีการตรวจสอบว่ามีค่าน้อยกว่าหรือเทียบเท่ากับ 10 หรือไม่ที่ส่วนท้ายของลูป เมื่อตัวแปร k มีค่าเป็น 11 จะทำให้เงื่อนไขใน while ไม่เป็นจริง การวนลูปจะสิ้นสุดลง จากนั้นขั้นตอนสุดท้ายคือการนำผลรวมของตัวเลข 1 ถึง 10 ไปหาร 10 เพื่อหาค่าเฉลี่ยแล้วแสดงออกทางหน้าจอ



รูปที่ 8.4 ความแตกต่างของ while และ do-while

รูปที่ 8.4 แสดงความแตกต่างของ while และ do-while จะเห็นได้ว่าทางฝั่งซ้ายจะเป็นการใช้คำสั่ง while โดยเงื่อนไขคือ 0 ซึ่งหมายถึงเท็จ ถ้าเป็นการเรียกใช้คำสั่ง while เมื่อเงื่อนไขเป็นเท็จ คำสั่งภายในของ while จะไม่ถูกกระทำและโปรแกรมจะกระโดดออกจากการทำซ้ำในทันที ดังนั้นผลการรันที่ได้ จะไม่แสดงข้อความใดๆ ทั้งสิ้นออกทางหน้าจอ กลับกันในรูปทางด้านขวาเป็นการใช้งานคำสั่ง do-while โปรแกรมจะทำงานคำสั่งภายใน do ก่อนหนึ่งครั้งซึ่งในที่นี้คือพิมพ์คำว่า Hello World ออกทางหน้าจอ จากนั้นจึงค่อยทำการเปรียบเทียบเงื่อนไขในการทำซ้ำ แต่เนื่องด้วยเงื่อนไขใน while เป็น 0 เงื่อนไขเป็นเท็จ ทำให้หลุดจากการทำซ้ำ เพราะฉะนั้นในหน้าจอจึงมีคำว่า Hello World เพียงคำเดียว

8.3 คำสั่ง for

คำสั่ง for ใช้สำหรับการควบคุมทิศทางของโปรแกรมให้ทำงานแบบวนรอบ เช่นเดียวกันกับคำสั่ง while และ do-while แต่คำสั่ง for มีลักษณะพิเศษกว่าคำสั่งลูปแบบอื่นๆ ตรงที่คำสั่ง for เหมาะสมกับกรณีที่รู้จำนวนแน่นอนแล้วว่า ต้องการให้วนลูปทำงานกี่รอบ รูปแบบการเรียกใช้งานคำสั่ง for ก็ต่างจากคำสั่งลูปอื่นๆ ดังนี้

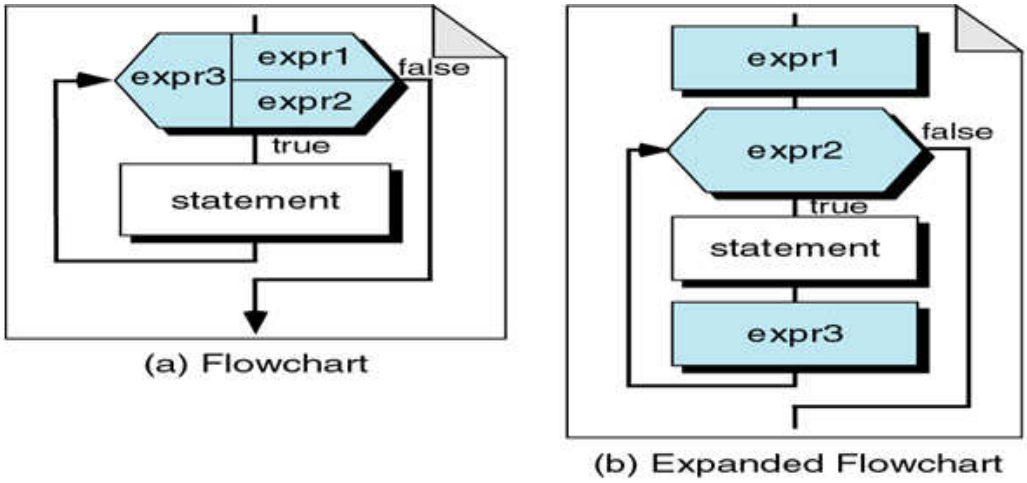
```
for(expr1;  expr2;  expr3)
{
    statement-1;
    statement-2;
    statement-3;
        …
    statement-n;
}
```

- expr1 : การกำหนดค่าเริ่มต้นของตัวแปรที่จะใช้เป็นเงื่อนไข ยกตัวอย่างเช่น x = 0 หมายถึงกำหนดค่าเริ่มต้นให้กับตัวแปร x เป็น 0
- expr2 : เงื่อนไขที่กำหนดขึ้น เพื่อให้โปรแกรมวนทำงานตามคำสั่งภายในของ for
- expr3 : ส่วนของการเปลี่ยนแปลงค่าของตัวแปรที่ใช้เป็นเงื่อนไข ซึ่งอาจเป็นการเพิ่มหรือลดค่าของตัวแปร เช่น x++, x-- , x+=10
- statement-1, statement-2, statement-3 : คำสั่งที่จะให้ทำงาน ถ้าผลการตรวจสอบเงื่อนไขออกมาเป็นจริง (true)

หรือ

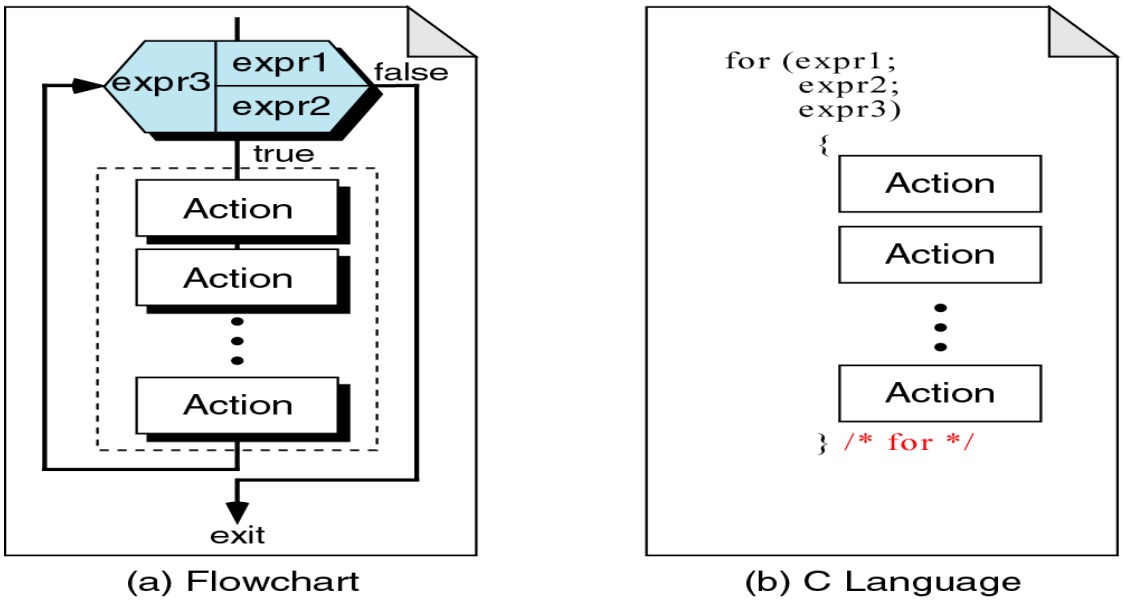
```
for(expr1;  expr2;  expr3)
   statement-1;
```

- expr1 : การกำหนดค่าเริ่มต้นของตัวแปรที่จะใช้เป็น
- expr2 : เงื่อนไขที่กำหนดขึ้น เพื่อให้โปรแกรมวนทำงานตามคำสั่งภายในของ for
- expr3 : ส่วนของการเปลี่ยนแปลงค่าของตัวแปรที่ใช้เป็นเงื่อนไข ซึ่งอาจเป็นการเพิ่มหรือลดค่าของตัวแปร เช่น x++, x-- , x+=10
- statement : คำสั่งที่จะให้ทำงาน ถ้าผลการตรวจสอบเงื่อนไขออกมาเป็นจริง (true)

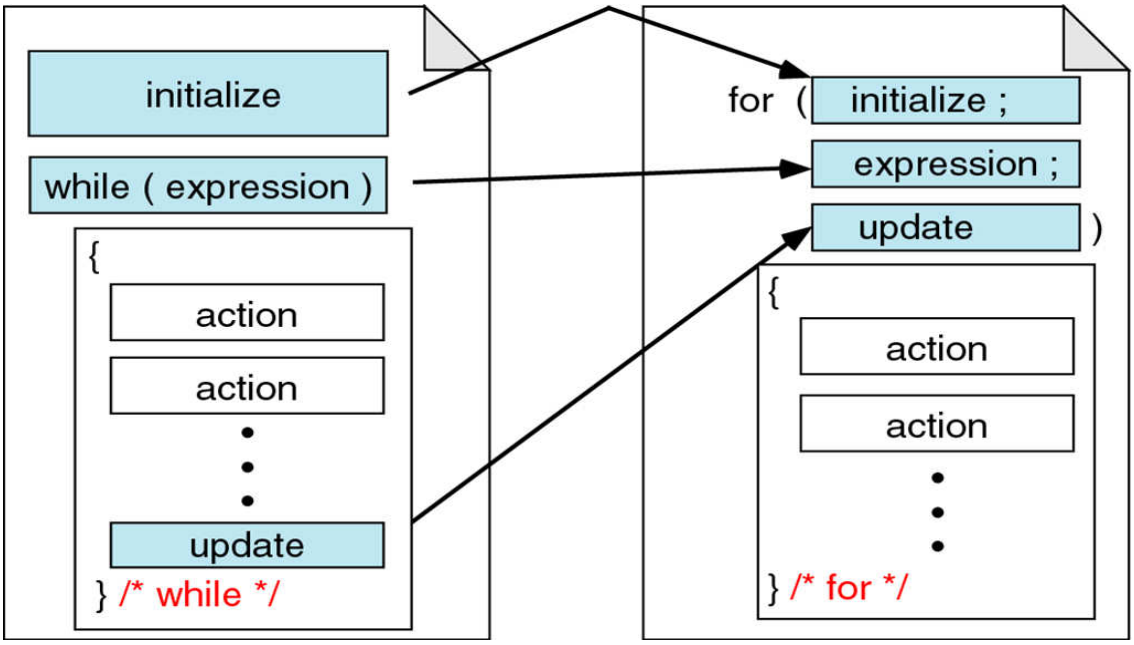


รูปที่ 8.5 แผนผังการทำงานของคำสั่ง for

จากรูปที่ 8.5 แสดงให้เห็นถึงแผนผังการทำงานของคำสั่ง for โดย 8.5(a) คือแผนผังการทำงานของคำสั่ง for แบบย่อ ในขณะที่ 8.5(b) เป็นแผนผังการทำงานของคำสั่ง for แบบขยาย ซึ่งจะเห็นได้ว่าการทำงานของคำสั่ง for จะทำงานคำสั่งใน expr1 ก่อนทุกครั้งหลังจากนั้นจะเป็นการเปรียบเทียบเงื่อนไขของชุดคำสั่งที่อยู่ใน expr2 ถ้าเงื่อนไขเป็นเท็จโปรแกรมจะกระโดดออกจากลูปเพื่อจบการทำงานของคำสั่ง for แต่ถ้าเงื่อนไขเป็นจริง โปรแกรมจะทำงานชุดคำสั่งภายใน for เมื่อทำงานเสร็จเรียบร้อย ตัวโปรแกรมจะทำงานชุดคำสั่งใน expr3 จากนั้นค่อยไปเปรียบเทียบเงื่อนไขเพื่อทำงานซ้ำในชุดคำสั่ง expr2 ต่อไป ซึ่งการใช้งานคำสั่ง for แสดงดังรูปที่ 8.6

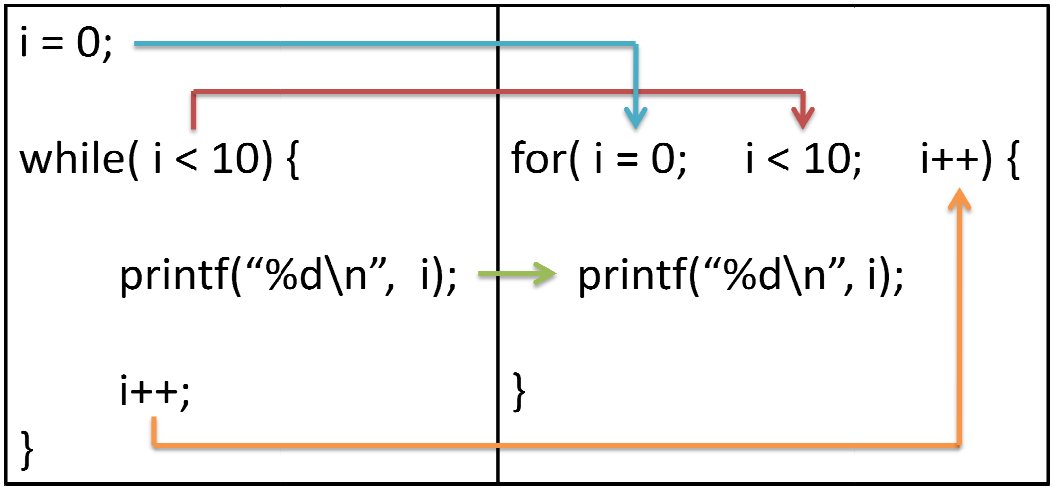


รูปที่ 8.6 การทำงานของคำสั่ง for



รูปที่ 8.7 เปรียบเทียบการทำงานของ while และ for

การทำงานของคำสั่ง for นั้นสามารถเปรียบเทียบกับการทำงานของคำสั่ง while ได้ดังรูปที่ 8.7 โดยปกติเมื่อมีการใช้งานคำสั่ง while จะมีการกำหนดค่าให้กับตัวแปรที่จะใช้เป็นตัวตรวจสอบเงื่อนไขในการทำซ้ำอยู่ก่อนการเรียกใช้งาน while ซึ่งจะเห็นได้ว่า คำสั่งในส่วนนี้จะนำมาใส่ไว้ในส่วน initialize ของคำสั่ง for สำหรับเงื่อนไขที่ตรวจสอบใน while ก็สามารถนำมาใส่ในส่วนของ expression ของคำสั่ง for สุดท้ายการเปลี่ยนแปลงค่าของตัวแปรที่ให้ในการตรวจสอบการทำซ้ำที่ปกติจะอยู่ที่บรรทัดสุดท้ายของการทำงานในคำสั่ง while ก็นำมาอยู่ในส่วน update ของคำสั่ง for ตัวอย่างโปรแกรมแสดงดังรูปที่ 8.8



รูปที่ 8.8 การเปลี่ยนโปรแกรมจากคำสั่ง while มาใช้คำสั่ง for

ตัวอย่างโปรแกรมที่ 8.6

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int  i;
5      for( i = 0;  i <= 10;  i++) {
6          printf("%d\n",  i);
7      }
8      printf("End of loop\n");
9  }
```

ผลการรัน

```
1
2
3
4
5
6
7
8
9
10
End of loop
```

การทำงานของตัวอย่างโปรแกรมที่ 8.6 จะเริ่มโดยกำหนดให้ตัวแปร i มีค่าเริ่มต้นเป็น 1 แล้วทดสอบว่า i <=10 หรือไม่ ซึ่งขณะนี้ i มีค่าเป็น 1 ดังนั้นจึงพิมพ์ค่าของตัวแปร i แล้วจึงเพิ่มค่าของตัวแปร i อีก 1 เป็น 2 ซึ่งมีค่าน้อยกว่า 10 ค่าของตัวแปร i จึงถูกพิมพ์อีกครั้ง การทำงานจะซ้ำกับที่กล่าวมาแล้วจนกระทั่งค่าของตัวแปร i มีค่าเป็น 11 จะทำให้เงื่อนไขไม่เป็นจริง การวนลูปจึงสิ้นสุดลง

ตัวอย่างโปรแกรมที่ 8.7

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int  i;
5      for( i = 0;  i <= 50;  i+=5) {
6          printf("%d ",  i);
7      }
8  }
```

ผลการรัน

```
 0  5  10  15  20  25  30  35  40  45  50
```

ตัวอย่างโปรแกรมที่ 8.7 จะพิมพ์ตัวเลข 0, 5, 10, 15…50 ออกทางหน้าจอโดยภายในโปรแกรมนี้ตัวแปร i จะมีค่าเพิ่มขึ้นครั้งละ 5

ตัวอย่างโปรแกรมที่ 8.8

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int  i;
5      for( i = 10;  i >= 1;  i--) {
6          printf("%d ",  i);
7      }
8  }
```

ผลการรัน

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

ตัวอย่างโปรแกรมที่ 8.8 จะพิมพ์ค่า 10 , 9 , 8 , 7……1 โดยการทำซ้ำจะกำหนดค่าเริ่มต้นให้กับตัวแปร i เป็น 10 จากนั้นจึงลดค่าตัวแปร i ทีละ 1 จนกระทั่งเงื่อนไขในลูปเป็นเท็จและสิ้นสุดการทำงาน

ตัวอย่างโปรแกรมที่ 8.9

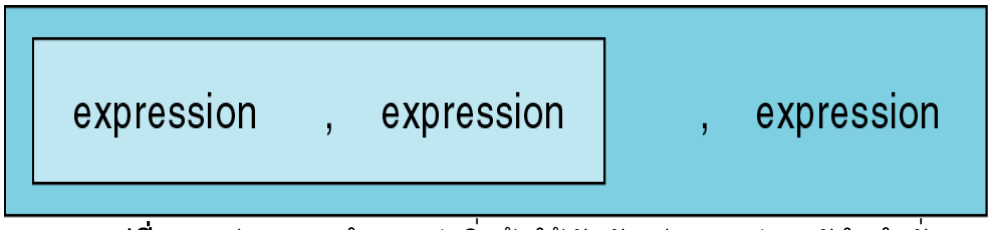
```
 1  #include <stdio.h>
 2  int  main(int  argc,  char **argv)
 3  {
 4      int  i,  sum;
 5      sum  = 0;
 6      for( i = 1;  i <= 10;  i++) {
 7           sum += i;
 8      }
 9      printf("summation of 1-10 = %d\n",  sum);
10  }
```

ผลการรัน

```
summation of 1-10 = 55
```

ตัวอย่างโปรแกรมที่ 8.9 แสดงการหาผลบวกของชุดตัวเลข 1-10 ในการหาผลบวกจะนำค่าของตัวแปร i มาบวกกับตัวแปร sum ขณะนี้มีค่าเป็นศูนย์ แล้วเก็บผลในตัวแปร sum จากนั้นเพิ่มค่าของตัวแปร i เป็น 2 แล้วมาบวกกับค่าของตัวแปร sum ซึ่งมีค่าเป็น 1 แล้วเก็บในตัวแปร sum อีกขณะนี้ sum จะมีค่า 1+2=3 การทำงานของโปรแกรมจะวนซ้ำจนกระทั่งตัวแปร i มีค่าเป็น 11 จึงเป็นเท็จแล้วจึงออกจากลูป

การกำหนดค่าเริ่มต้นให้กับคำสั่ง for นอกจากจะทำเพียงแค่ตัวแปรเดียวเช่น i = 0 หรือ i = 1 ภาษาซียังอนุญาตให้สามารถกำหนดค่าเริ่มต้นให้กับตัวแปรมากกว่า 1 ตัวได้ โดยการใส่เครื่องหมาย "," คั่นระหว่างตัวแปรต่างๆ ดังรูปที่ 8.9



รูปที่ 8.9 รูปแบบการกำหนดค่าเริ่มต้นให้กับตัวแปรมากกว่า 1 ตัวในคำสั่ง for

ยกตัวอย่างเช่น ถ้าในคำสั่ง for ต้องการจะกำหนดค่าเริ่มต้นให้กับตัวแปร 2 ตัว x และ y โดยมีค่าเริ่มต้นเป็น 10 และ 0 ตามลำดับสามารถเขียนได้ดังนี้

for(x = 10, y = 0; เงื่อนไข; การเปลี่ยนแปลงค่า)

ตัวอย่างโปรแกรมที่ 8.10

```
 1  #include <stdio.h>
 2  int  main(int  argc,  char **argv)
 3  {
 4      int   sum,  p,  x,  y;
 5      sum  = 0;
 6      for( x = 1,  y  = 1;  x * y <= 50;  x++,  y+=2) {
 7           p  = x * y;
 8           sum = sum + p;
 9           printf("%d   *  %d = %d\n", x,  y,  p);
10      }
11      printf("summation of x * y = %d\n",  sum);
12  }
```

ผลการรัน

```
1 * 1 = 1
2 * 3 = 6
3 * 5 = 15
4 * 7 = 28
5 * 9 = 45
summation of x * y = 95
```

ตัวอย่างโปรแกรมที่ 8.10 แสดงการใช้งานตัวแปรควบคุมลูปที่มีจำนวนมากกว่า 1 ตัว ซึ่งมี x และ y เป็นตัวแปรควบคุมการวนลูป โดยเมื่อเริ่มต้นการทำงานของลูปจะมีการกำหนดค่าให้กับ x และ y เป็น 1 จากนั้นจะมีการตรวจสอบเงื่อนไขการทำซ้ำ x * y <= 50 ในการทำงานครั้งแรกนั้น 1 * 1 <= 50 เป็นจริง จึงทำให้โปรแกรมทำคำสั่งภายในของลูป for คือการคำนวณค่า p ที่เกิดจาก x * y ก็คือ 1 * 1 = 1 และมีการเพิ่มค่าในตัวแปร sum จาก ค่าในตัวแปร p ที่คำนวณได้ จากนั้นจึงแสดงผลลัพธ์ของมาทางหน้าจอเป็น 1 * 1 = 1 เมื่อทำงานซ้ำเสร็จสิ้น 1 รอบ ตัวแปร x จะถูกเพิ่มค่าไป 1 ค่าจาก (x++) แต่ตัวแปร y จะถูกเพิ่มค่าไป 2 ค่า (y += 2) โปรแกรมนี้จะพิมพ์ผลคูณของตัวแปร x และ y ไปเรื่อยๆ จนกระทั่งผลคูณมีค่ามากกว่า 50 ซึ่งค่า

ตัวอย่างโปรแกรมที่ 8.11

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int   i,  j;
5      for( i = 1;  i <= 4;  i++ ) {
6          for( j = 1;  j <= 3; j++) {
7              printf("C is very easy\n");
8          }
9      }
10 }
```

ผลการรัน

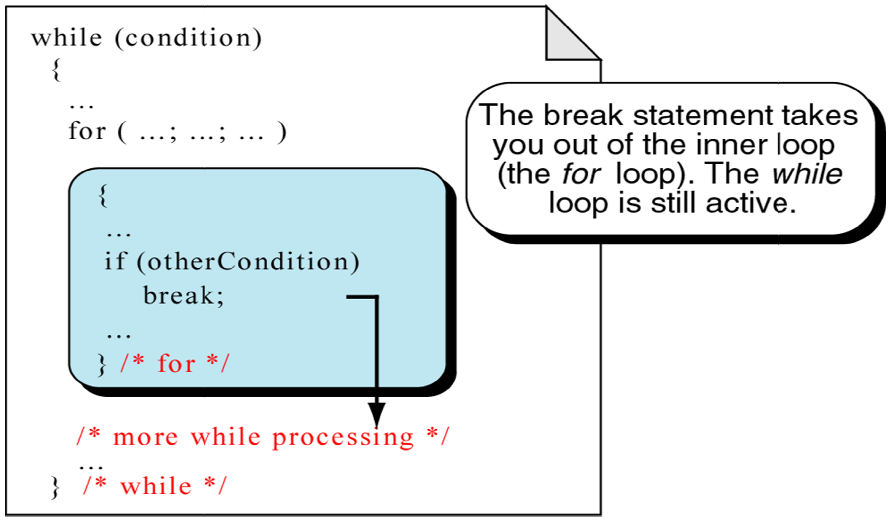
```
C is very easy
C is very easy
C is very easy
C is very easy
C is very easy
C is very easy
C is very easy
C is very easy
C is very easy
C is very easy
C is very easy
C is very easy
```

ตัวอย่างโปรแกรมที่ 8.11 จะมีลูปซ้อนกัน (Nested loops) ในตอนเริ่มแรกตัวแปร i และ j จะมีค่าเป็น 1 การวนลูปจะเริ่มที่ลูปในสุดก่อนโดยจะพิมพ์ข้อความ "C is very easy" 3 บรรทัด (ทำ 3 รอบ) จากนั้นเพิ่มค่าของตัวแปร i เป็น 2 แล้วทำลูป j อีกโดยพิมพ์ข้อความ " C is very easy" 3 บรรทัดอีก การทำงานจะซ้ำเดิมอีกจนกระทั่งตัวแปร i มีค่ามากกว่า 4 ก็จะจบโปรแกรม นั่นคือที่ค่า i หนึ่ง ๆ จะพิมพ์ข้อความ " C is very easy" 3 บรรทัด ดังนั้นในโปรแกรมนี้จะพิมพ์ทั้งหมด 4x3=12 บรรทัด

8.4 คำสั่ง break

นอกจากคำสั่งควบคุมทิศทางการทำงานของโปรแกรมในแบบต่างๆ แล้ว ภาษาซียังมีคำสั่งอีกกลุ่มหนึ่ง ซึ่งจะไม่ได้ใช้สำหรับควบคุมทิศทางของโปรแกรมโดยตรง แต่คำสั่งเหล่านี้มักจะถูกนำไปใช้ร่วมกับคำสั่งอื่นๆ เช่น นำไปใช้เพื่อหยุดการเลือกทำ หรือออกจากการทำงานของระบบ

คำสั่ง break ถูกนำไปร่วมกับคำสั่งแบบเลือกทำหรือวนรอบ เพื่อสั่งให้โปรแกรมหยุดการทำงานของคำสั่งแบบเลือกทำหรือวนรอบที่กำลังทำงานอยู่ ในกรณีที่มีการทำลูปซ้อนกัน เมื่อเจอคำสั่ง break จะเป็นเพียงการหยุดการทำงานของการทำงานซ้ำเพียงลูปเดียว จะไม่เกี่ยวข้องกันลูปภายนอกที่เหลือ ดังแสดงในรูปที่ 8.10



รูปที่ 8.10 การทำงานของ break ใน nested loop

ตัวอย่างโปรแกรมที่ 8.11

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int   i;
5      for (i = 0;  i < 5; i++) {
6            printf("<");
7            if (i == 2)
8                  break;
9            printf("%d  >", i);
10     }
11 }
```

ผลการรัน

```
<0><1><
```

จากตัวอย่างโปรแกรมที่ 8.11 แสดงให้เห็นถึงการใช้งานคำสั่ง break โดยโปรแกรมจะวนลูปด้วยคำสั่ง for ซึ่งมีตัวแปร i เป็นตัวควบคุมลูปโดยปกติแล้ว i จะมีค่า 0, 1, 2, 3, 4 ทำงานในลูปตามลำดับ

- เมื่อ i มีค่าเป็น 0 บรรทัดที่ 6 จะแสดง “<” ออกทางหน้าจอ จากนั้นเป็นการตรวจสอบเงื่อนไขใน if ในที่นี้ i = 0 ไม่ใช่ 2 โปรแกรมจึงไม่ทำคำสั่ง break ข้ามไปทำคำสั่ง printf( ) เพื่อแสดง “0>” ออกทางหน้าจอ ดังนั้นเมื่อจบการทำงานลูปครั้งแรก จะมีข้อความ “<0>” แสดงออกทางหน้าจอ
- เมื่อ i มีค่าเป็น 1 บรรทัดที่ 6 จะแสดง “<” ออกทางหน้าจอ จากนั้นเป็นการตรวจสอบเงื่อนไขใน if ในที่นี้ i = 1 ไม่ใช่ 2 โปรแกรมจึงไม่ทำคำสั่ง break ข้ามไปทำคำสั่ง printf( ) เพื่อแสดง “1>” ออกทางหน้าจอ ดังนั้นเมื่อจบการทำงานลูปครั้งแรก จะมีข้อความ “<1>” แสดงออกทางหน้าจอ เมื่อรวมกับค่าที่ถูกแสดงเมื่อ i มีค่าเป็น 0 ข้อความที่เห็นบนหน้าจอจึงเป็น “<0><1>”
- เมื่อ i มีค่าเป็น 2 บรรทัดที่ 6 จะแสดง “<” ออกทางหน้าจอ จากนั้นเป็นการตรวจสอบเงื่อนไขใน if ในที่นี้ i = 2 ซึ่งทำให้เงื่อนไขของ if เป็นจริง โปรแกรมจึงทำคำสั่ง break เพื่อหยุดการทำงานของการวนลูปทำให้โปรแกรมสิ้นสุดลงที่จุดนี้

ดังนั้นผลลัพธ์สุดท้ายที่แสดงบนหน้าจอจึงเป็น <0><1><

ตัวอย่างโปรแกรมที่ 8.12

```
1  #include <stdio.h>
2  int main(int argc, char **argv) {
3     int i, j;
4     for (i = 0; i < 5; i++) {
5        for(j = 0; j < 5; j++) {
6           break;
7           printf(“J = %d\n”, j);
8        }
9        printf("I = %d\n", i);
10    }
11 }
```

ผลการรัน

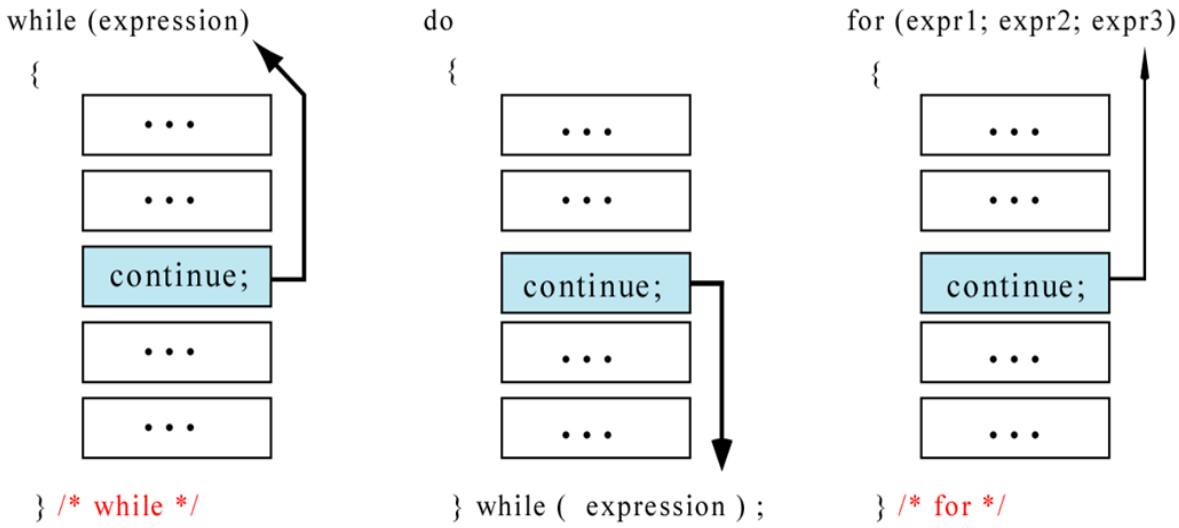
```
I = 0
I = 1
I = 2
I = 3
I = 4
```

ลูปข้างในคือ j จะมีการใช้คำสั่ง break ก่อนฟังก์ชัน printf( ) ทำให้ไม่มีข้อความ J = ออกทางหน้าจอ จากตัวอย่างเห็นได้ว่า การใช้คำสั่ง break จะหยุดการทำงานของลูปที่อยู่เพียงลูปเดียวไม่ได้กระทบกับลูปอื่นที่อยู่ภายนอก

8.5 คำสั่ง continue

ในการเขียนโปรแกรมบางครั้งอาจจะต้องการให้ข้ามบางคำสั่งซึ่งอยู่ในลูปเพื่อเริ่มต้นการวนซ้ำในรอบต่อไป การบังคับให้โปรแกรมทำงานในลักษณะดังกล่าวจะต้องใช้คำสั่ง continue ซึ่งทำงานตรงกันข้ามกับคำสั่ง break

เมื่อมีคำสั่ง continue อยู่ในลูปของคำสั่ง while และ do-while จะทำให้โปรแกรมกระโดดไปทดสอบเงื่อนไข สำหรับในคำสั่ง for โปรแกรมจะไปเพิ่มหรือลดค่าของตัวแปรควบคุมลูปตามที่กำหนดไว้จากนั้นจะไปทดสอบเงื่อนไข ดังรูปที่ 8.11



รูปที่ 8.11 การทำงานของคำสั่ง continue

ตัวอย่างโปรแกรมที่ 8.13

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)
3  {
4      int   i;
5      for (i = 0;  i < 5; i++) {
6            printf("<");
7            if (i == 2)
8                  continue;
9            printf("%d  >", i);
10     }
11 }
```

ผลการรัน

```
<0><1><<3><4>
```

จากตัวอย่างโปรแกรมที่ 8.13 แสดงให้เห็นถึงการใช้งานคำสั่ง continue โดยโปรแกรมจะวนลูปด้วยคำสั่ง for ซึ่งมีตัวแปร i เป็นตัวควบคุมลูปโดยปกติแล้ว i จะมีค่า 0, 1, 2, 3, 4 ทำงานในลูปตามลำดับ

- เมื่อ i มีค่าเป็น 0 บรรทัดที่ 6 จะแสดง “<” ออกทางหน้าจอ จากนั้นเป็นการตรวจสอบเงื่อนไขใน if ในที่นี้ i = 0 ไม่ใช่ 2 โปรแกรมจึงไม่ทำคำสั่ง continue ข้ามไปทำคำสั่ง printf( ) เพื่อแสดง “0>” ออกทางหน้าจอ ดังนั้นเมื่อจบการทำงานลูปครั้งแรก จะมีข้อความ “<0>” แสดงออกทางหน้าจอ
- เมื่อ i มีค่าเป็น 1 บรรทัดที่ 6 จะแสดง “<” ออกทางหน้าจอ จากนั้นเป็นการตรวจสอบเงื่อนไขใน if ในที่นี้ i = 1 ไม่ใช่ 2 โปรแกรมจึงไม่ทำคำสั่ง continue ข้ามไปทำคำสั่ง printf( ) เพื่อแสดง “1>” ออกทางหน้าจอ ดังนั้นเมื่อจบการทำงานลูปครั้งแรก จะมีข้อความ “<1>” แสดงออกทางหน้าจอ เมื่อรวมกับค่าที่ถูกแสดงเมื่อ i มีค่าเป็น 0 ข้อความที่เห็นบนหน้าจอจึงเป็น “<0><1>”
- เมื่อ i มีค่าเป็น 2 บรรทัดที่ 6 จะแสดง “<” ออกทางหน้าจอ จากนั้นเป็นการตรวจสอบเงื่อนไขใน if ในที่นี้ i = 2 ซึ่งทำให้เงื่อนไขของ if เป็นจริง โปรแกรมจึงทำคำสั่ง continue เพื่อข้ามการทำงานของลูปไปยังคำสั่งเพิ่มค่า i เพื่อทำงานลูปถัดไป ข้อความที่เห็นบนหน้าจอจึงเป็น “<0><1><”
- เมื่อ i มีค่าเป็น 3 บรรทัดที่ 6 จะแสดง “<” ออกทางหน้าจอ จากนั้นเป็นการตรวจสอบเงื่อนไขใน if ในที่นี้ i = 3 ไม่ใช่ 2 โปรแกรมจึงไม่ทำคำสั่ง continue ข้ามไปทำคำสั่ง printf( ) เพื่อแสดง “3>” ออกทางหน้าจอ ดังนั้นเมื่อจบการทำงานลูปครั้งแรก จะมีข้อความ “<3>” แสดงออกทางหน้าจอ เมื่อรวมกับค่าที่ถูกแสดงที่ผ่านมาแล้ว ข้อความที่เห็นบนหน้าจอจึงเป็น “<0><1><<3>”
- เมื่อ i มีค่าเป็น 4 บรรทัดที่ 6 จะแสดง “<” ออกทางหน้าจอ จากนั้นเป็นการตรวจสอบเงื่อนไขใน if ในที่นี้ i = 4 ไม่ใช่ 2 โปรแกรมจึงไม่ทำคำสั่ง continue ข้ามไปทำคำสั่ง printf( ) เพื่อแสดง “4>” ออกทางหน้าจอ ดังนั้นเมื่อจบการทำงานลูปครั้งแรก จะมีข้อความ “<4>” แสดงออกทางหน้าจอ เมื่อรวมกับค่าที่ถูกแสดงที่ผ่านมาแล้ว ข้อความที่เห็นบนหน้าจอจึงเป็น “<0><1><<3><4>”
- เมื่อ i มีค่าเป็น 5 เงื่อนไขของคำสั่ง for จะเป็นเท็จและหยุดการทำงานของลูป ดังนั้นผลลัพธ์สุดท้ายที่แสดงบนหน้าจอจึงเป็น “<0><1><<3><4>”

ตัวอย่างโปรแกรมที่ 8.14

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv)  {
3      float   sum = 0,  avg,  x;
4      int   i,  n  = 0;
5      for( i = 1;  i <=  5;  i++ )   {
6            printf("x%d = ",  i);
7            scanf("%f",   &x);
8            if( x > 60)
9                  continue;
10           sum += x;
11           n++;
12     }
13     avg  =  sum / n;
14     printf("Average value of numbers between 1-60 is  %.4f\n", avg);
15 }
```

ผลการรัน

x1 = ***12***

x2 = ***89***

x3 = ***65***

x4 = ***78***

x5 = ***23***

Average value of numbers between 1-60 is 17.5000

โปรแกรมที่ 8.14 เป็นการหาค่าเฉลี่ยของกลุ่มตัวเลขมีค่าระหว่าง 1-60 ถ้าตัวเลขที่ป้อนมีค่ามากกว่า 60 จะไม่นำมารวมกับตัวแปร sum จะเห็นว่าคำสั่ง continue จะทำให้โปรแกรมวนลูปใหม่โดยไม่ทำคำสั่ง sum += x และ n++ จากผลการรันจะเห็นได้ว่ามีค่าที่รับเข้ามาคือ x1 และ x5 เท่านั้นที่น้อยกว่าหรือเท่ากับ 60 ดังนั้นผลลัพธ์ที่ได้จากการรันก็คือ (12 + 23) / 2 ซึ่งก็คือ 17.5000

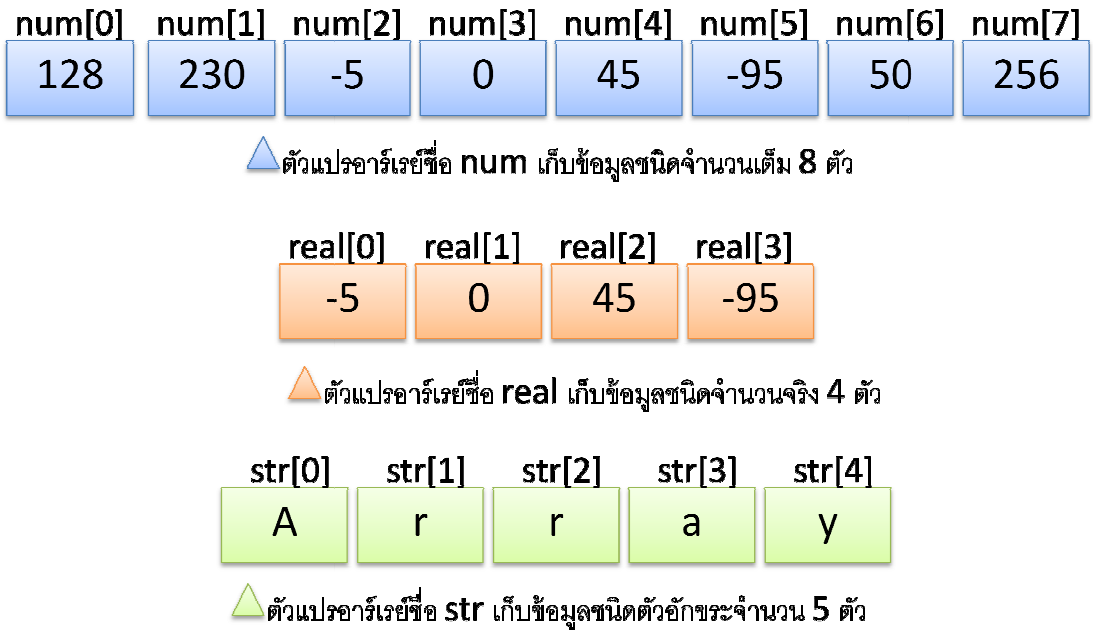
บทที่ 9
อาร์เรย์และสตริงก์

จากที่ผ่านมาเรารู้ว่าการจะนำข้อมูลเข้ามาใช้ในโปรแกรม ต้องสร้างตัวแปรสำหรับเก็บข้อมูลนั้นขึ้นมาเสียก่อนซึ่งถ้าข้อมูลที่จะนำเข้ามาหลายตัว อย่างเช่น จำนวนเต็ม 10 ตัว เราก็จำเป็นต้องสร้างตัวแปรประเภทจำนวนเต็ม (int) ขึ้นมา 10 ตัว ซึ่งบางครั้งอาจจะเกิดความสับสนชื่อของตัวแปรได้ ในภาษาซี มีวิธีที่ง่ายและสะดวกในการสร้างตัวแปรสำหรับเก็บข้อมูลชนิดเดียวกันหลายๆ ตัว นั่นก็คือ การสร้างตัวแปรชุด หรือที่เรียกว่า ตัวแปรอาร์เรย์ (array)

9.1 ตัวแปรอาร์เรย์

ตัวแปรอาร์เรย์ คือ ตัวแปรที่สามารถเก็บข้อมูลได้เป็นชุด โดยประกาศสร้างตัวแปรขึ้นมาเพียงตัวเดียว แต่ข้อมูลทั้งหมดนั้นจะต้องเป็นข้อมูลชนิดเดียวกัน อย่างเช่น จำนวนเต็ม 5 ตัว จำนวนจริง 7 ตัว หรืออักขระ 24 ตัว เป็นต้น ตัวแปรอาร์เรย์สามารถเรียกได้อีกอย่างหนึ่งว่า ตัวแปรชุด

เพื่อเพิ่มความเข้าใจเกี่ยวกับการเก็บข้อมูลในตัวแปรอาร์เรย์ ให้ดูจากรูปที่ 9.1 จะเห็นได้ว่าตัวแปรอาร์เรย์ก็เหมือนกับการนำตัวแปรชนิดเดียวกันหลายๆตัว มาเรียงต่อกันโดยใช้ชื่อเดียวกัน วิธีแยกความแตกต่างของข้อมูลแต่ละตัว จะใช้ตัวเลขเขียนต่อท้ายชื่อของตัวแปรอาร์เรย์ภายในเครื่องหมาย [ ] โดยเริ่มจากตัวแปรตัวแรกเป็น 0 นับเพิ่มไปเรื่อยๆ จนถึงตัวสุดท้าย ซึ่งตัวเลขนี้จะเรียกว่า อินเด็กซ์ (index)



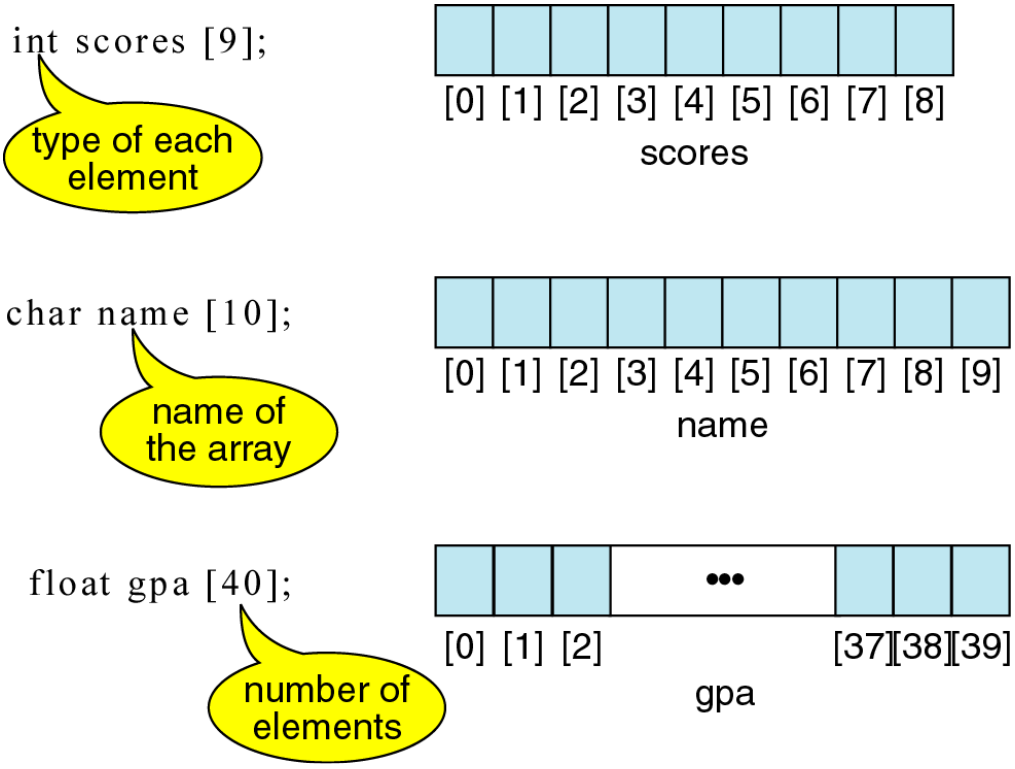
รูปที่ 9.1 โครงสร้างและการเก็บข้อมูลของตัวแปรอาร์เรย์

9.2 อาร์เรย์ 1 มิติ

การสร้างตัวแปรอาร์เรย์ขึ้นมาใช้งานในโปรแกรม จะใช้คำสั่งประกาศตัวแปรเหมือนกับตัวแปรชนิดอื่นๆที่ผ่านมา แต่มีส่วนที่แตกต่างก็คือ การระบุขนาดของตัวแปรอาร์เรย์ที่จะสร้างขึ้นด้วย โดยเริ่มจากรูปแบบการประกาศตัวแปรอาร์เรย์ 1 มิติ ดังนี้

type variable-name[n] ;
• type : ชนิดของตัวแปรอาร์เรย์ที่จะสร้างขึ้น โดยพิจารณาจากชนิดของข้อมูลที่ต้องการจะเก็บในตัวแปรอาร์เรย์ • variable-name : ชื่อของตัวแปรอาร์เรย์ • n : ขนาดของตัวแปรอาร์เรย์ที่จะสร้างขึ้น

ตัวอย่างการประกาศสร้างตัวแปรอาร์เรย์ 1 มิติ แสดงดังรูปที่ 9.2



รูปที่ 9.2 การประกาศตัวแปรอาร์เรย์ 1 มิติ

จากรูปที่ 9.2 มีการประกาศตัวแปรอาร์เรย์ 1 มิติ 3 ตัวซึ่งก็คือ

- int score [9]; (type คือ int , variable-name คือ score , n คือ 9)
- char name [10] ; (type คือ char , variable-name คือ name , n คือ 10)
- float gpa [40] ; (type คือ float , variable-name คือ gpa , n คือ 40)

โดยมีการกำหนดตัวแปรอาร์เรย์ชื่อ score มีขนาดเป็น 9 นั่นคือจะมีตัวแปรอาร์เรย์ (array variable) 9 ตัวคือ score[0], score[1], …, score[8] ตัวแปรอาร์เรย์เหล่านี้จะเก็บข้อมูลที่เป็นเลขจำนวนเต็ม สำหรับตัวแปรอาร์เรย์ name จะเก็บข้อมูลสตริงก์ซึ่งมีความยาวหรือจำนวนตัวอักษรไม่เกิน 10 ตัว และตัวแปรอาร์เรย์ gpa จะเก็บข้อมูลชนิดจำนวนจริงโดยสามารถเก็บค่าจำนวนจริงได้ทั้งหมด 40 ตัว

ในภาษาซีตัวแปรอาร์เรย์ตัวแรกจะมีอินเด็กซ์(Index) เป็นศูนย์เสมอ ในการอ้างถึงตัวแปรอาร์เรย์ใด ๆ จะต้องระบุด้วยชื่ออาร์เรย์และอินเด็กซ์ซึ่งอยู่ภายในเครื่องหมาย [ ] เช่น x[2] หมายถึงตัวแปรอาร์เรย์ตัวที่ 3

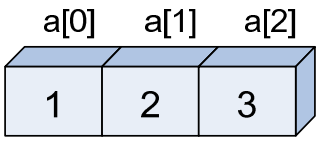
ตัวอย่างโปรแกรมที่ 9.1

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3       int   a[3];
4      a[0]  =  1;    a[1]  =  2;    a[2]  =  3;
5      printf("a[0] = %d", a[0]);
6      printf("a[1] = %d", a[1]);
7      printf("a[2] = %d", a[2]);
8  }
```

ผลการรัน

```
a[0]  = 1
a[1]  = 2
a[2]  = 3
```

ในโปรแกรมที่ 9.1 กำหนดตัวแปรอาร์เรย์ a มีขนาดเป็น 3 โดยตัวแปรอาร์เรย์ a[0] , a[1] และ a[2] ถูกกำหนดให้มีค่าเป็น 1, 2 และ 3 ตามลำดับ อาร์เรย์ a จะจัดเก็บข้อมูลดังรูปที่ 9.3



รูปที่ 9.3 ตัวแปรอาร์เรย์ a จากตัวอย่างโปรแกรมที่ 9.1

ตัวอย่างโปรแกรมที่ 9.2

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3      int   a[3],   i;
4      for(i = 0;  i  <  3;  i++)
5          a[i]  =  i+1;
6      for(i = 0;  i  <  3;  i++)
7          printf("a[%d] = %d", i, a[0]);
8  }
```

ผลการรัน

```
a[0]  = 1
a[1]  = 2
a[2]  = 3
```

จากตัวอย่างโปรแกรมที่ 9.2 จะให้ผลลัพธ์เหมือนกับตัวอย่างโปรแกรมที่ 9.1 แต่มีการใช้คำสั่ง for เพื่อสร้างลูปในการกำหนดค่าและพิมพ์ข้อมูลให้กับตัวแปรอาร์เรย์ซึ่งทำให้โปรแกรมสั้นกะทัดรัดกว่า

ตัวอย่างโปรแกรมที่ 9.3 จะแสดงให้เห็นถึงการรับค่าทางแป้นพิมพ์มาใส่ในตัวแปรอาร์เรย์โดยค่าที่จะป้อนเข้ามานั้นจะเป็นค่าของอุณหภูมิของ 12 เดือน ดังในตารางที่ 9.1 แล้วนำมาคำนวณหาค่าเฉลี่ยของอุณหภูมิทั้ง 12 เดือนออกแสดงผลทางหน้าจอ

ตารางที่ 9.1 อุณหภูมิของแต่ละเดือนใน 1 ปี

เดือน	อุณหภูมิ C
ม.ค.	26.0
ก.พ	28.7
มี.ค.	29.5
เม.ย	31.0
พ.ค.	30.3
มิ.ย.	29.1
ก.ค.	28.8
ส.ค.	29.3
ก.ย.	28.6
ต.ค.	27.9
พ.ย.	27.9
ธ.ค.	26.8

ตัวอย่างโปรแกรมที่ 9.3

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3      float  temp[12],  sum,  avg_temp;
4      int  i;
5      sum  =  0;
6      for(i  = 1;  i <= 12;  i++)   {
7          printf("temp[%d] = ", i);
8          scanf("%f",  &temp[i - 1]);
9          sum +=  temp[i - 1];
10     }
11     avg_temp = sum / 12;
12     printf("Average temperature = %.2f", avg_temp);
13 }
```

ผลการรัน

temp[1] = ***26***

temp[2] = ***28.7***

temp[3] = ***29.5***

temp[4] = ***31***

temp[5] = ***30.3***

temp[6] = ***29.1***

temp[7] = ***28.8***

temp[8] = ***29.3***

temp[9] = ***28.6***

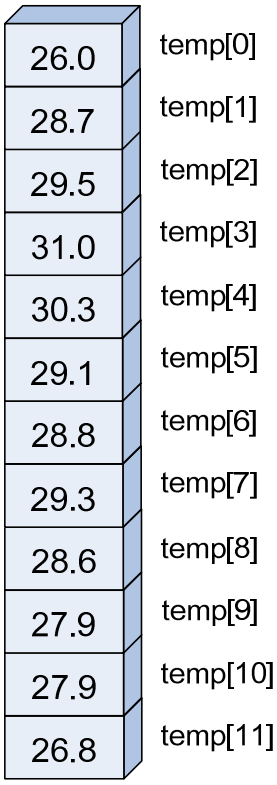
temp[10] = ***27.9***

temp[11] = ***27.9***

temp[12] = ***26.8***

Average temperature = 28.66

จากตัวอย่างโปรแกรมที่ 9.3 มีการประกาศตัวแปรอาร์เรย์ประเภทข้อมูลชนิดจำนวนจริงชื่อ temp ซึ่งมีขนาดเก็บค่าจำนวนจริงได้ 12 ค่า ในบรรทัดที่ 3 สำหรับบรรทัดที่ 6 -10 มีการสร้างลูปเพื่อรับค่าทางแป้นพิมพ์ และทำการรวมค่าที่รับเข้ามาในตัวแปรชื่อ sum รูปที่ 9.4 แสดงค่าในตัวแปรอาร์เรย์ temp ที่รับเข้ามาทางแป้นพิมพ์



รูปที่ 9.4 ค่าที่เก็บในตัวแปรอาร์เรย์ temp

เมื่อลูปทำงานเสร็จสิ้นทั้ง 12 ครั้งค่าในตัวแปร sum จะมีค่าเท่ากับ 26.0 + 28.7 + 29.5 + 31.0 + 30.3 + 29.1 + 28.8 + 29.3 + 28.6 + 27.9 + 27.9 + 26.8 = 343.9 จากนั้นในบรรทัดที่ 11 จะเป็นการหาค่าเฉลี่ยให้กับค่าจำนวนจริงทั้ง 12 ค่าที่รับเข้ามาจากแป้นพิมพ์เมื่อรวมค่ากันแล้วจะเก็บค่าไว้ในตัวแปร sum ดังนั้นจึงนำค่าที่อยู่ในตัวแปร sum มาหาร 12 แล้วเก็บค่าไว้ในตัวแปรชื่อ avg_temp ค่าที่ได้คือ 343.9 / 12 = 28.658333 จากนั้นในบรรทัดที่ 12 จะเป็นการแสดงผลออกทางหน้าจอด้วยฟังก์ชัน printf( ) แต่เนื่องจากตัวควบคุมในฟังก์ชัน printf( ) คือ %.2f ซึ่งหมายถึงการแสดงค่าที่อยู่ในตัวแปร avg_temp ออกทางหน้าจอด้วยจุดทศนิยมเพียง 2 ตำแหน่ง ดังนั้นจึงสามารถเห็นจากผลการรันได้ว่าค่าเฉลี่ยของอุณหภูมิทั้ง 12 เดือน คือ 28.66

ตัวอย่างโปรแกรมที่ 9.4

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3      int  A1[4],  B1[4],  C1[4], i;
4      for(i  = 0;  i < 4;  i++)   {
5          printf("A1[%d] = ", i);
6          scanf("%d",  &A1[i]);
7      }
8      for(i  = 0;  i < 4;  i++)   {
9          printf("B1[%d] = ", i);
10         scanf("%d",  &B1[i]);
11     }
12     for(i  = 0;  i < 4;  i++)
13         C1[i] = A1[i] + B1[i];
14     for(i  = 0;  i < 4;  i++)
15         printf("C1[%d] = %d\n",  C1[i]);
16 }
```

ผลการรัน

A1[0] = *2*

A1[1] = *1*

A1[2] = *4*

A1[3] = *3*

B1[0] = *1*

B1[1] = *5*

B1[2] = *3*

B1[3] = *2*

C1[0] = *3*

C2[1] = *6*

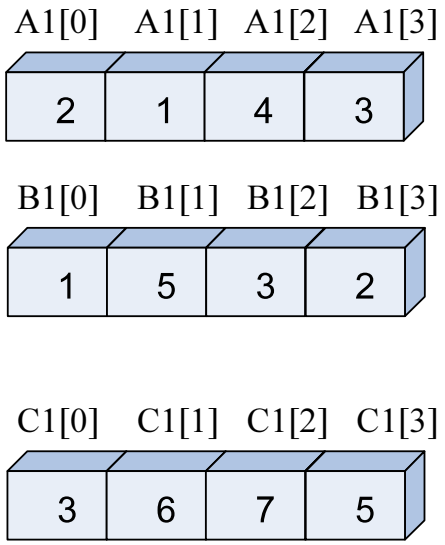
C3[2] = *7*

C4[3] = *5*

ตัวอย่างโปรแกรมที่ 9.4 จะมีการประกาศตัวแปรอาร์เรย์ของข้อมูลประเภทจำนวนเต็มขึ้นมาทั้งหมด 3 ตัวในบรรทัดที่ 3 คือ A1, B1, และ C1 โดยทั้ง 3 ตัวแปรอาร์เรย์นี้สามารถเก็บข้อมูลได้ทั้งหมด 4 ค่า บรรทัดที่ 4 - 7 เป็นการวนลูปเพื่อรับค่าจากแป้นพิมพ์มาเก็บไว้ยังตัวแปรอาร์เรย์ A1 จากผลการรันกำหนดให้ใส่ค่า 2, 1, 4, และ 3 ในตัวแปร A1[0], A1[1], A1[2], และ A1[3] ตามลำดับ ในขณะที่บรรทัดที่ 8 - 11 เป็นการวนลูปเพื่อรับค่าจากแป้นพิมพ์มาเก็บไว้ยังตัวแปรอาร์เรย์ B1 จากผลการรันกำหนดให้ใส่ค่า 1, 5, 3, และ 2 ในตัวแปร B1[0], B1[1], B1[2], และ B1[3] ตามลำดับ ในขณะที่บรรทัดที่ 12 - 13 เป็นการวนลูปเพื่อคำนวณหาค่าให้กับตัวแปรอาร์เรย์ C1 ซึ่งจะคำนวณให้ลักษณะนี้

- C1[0] = A1[0] + B1[0]
- C1[1] = A1[1] + B1[1]
- C1[2] = A1[2] + B1[2]
- C1[3] = A1[3] + B1[3]

ค่าในตัวแปรอาร์เรย์ C1 จะถูกแสดงผลออกทางหน้าจอด้วยคำสั่งในบรรทัดที่ 14 - 15 ดังรูปที่ 9.5 แสดงค่าที่เก็บไว้ในตัวแปรอาร์เรย์ A1, B1 และ C1



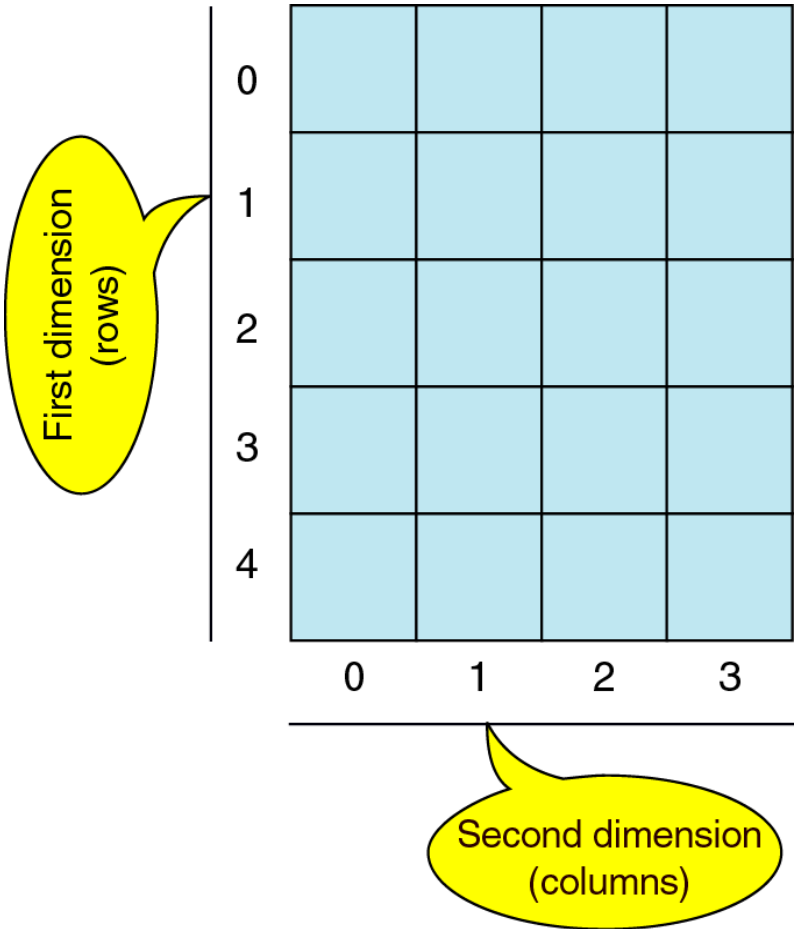
รูปที่ 9.5 ค่าในตัวแปรอาร์เรย์ A1, B1, และ C1

9.3 อาร์เรย์ 2 มิติ

การประกาศตัวแปรอาร์เรย์ 2 มิติจะคล้ายกับการสร้างตัวแปรอาร์เรย์ 1 มิติ ต่างกันที่ตรงการกำหนดขนาดของตัวแปรอาร์เรย์ ซึ่งต้องกำหนดทั้งจำนวนแถวและคอลัมน์ ดังแสดงต่อไปนี้

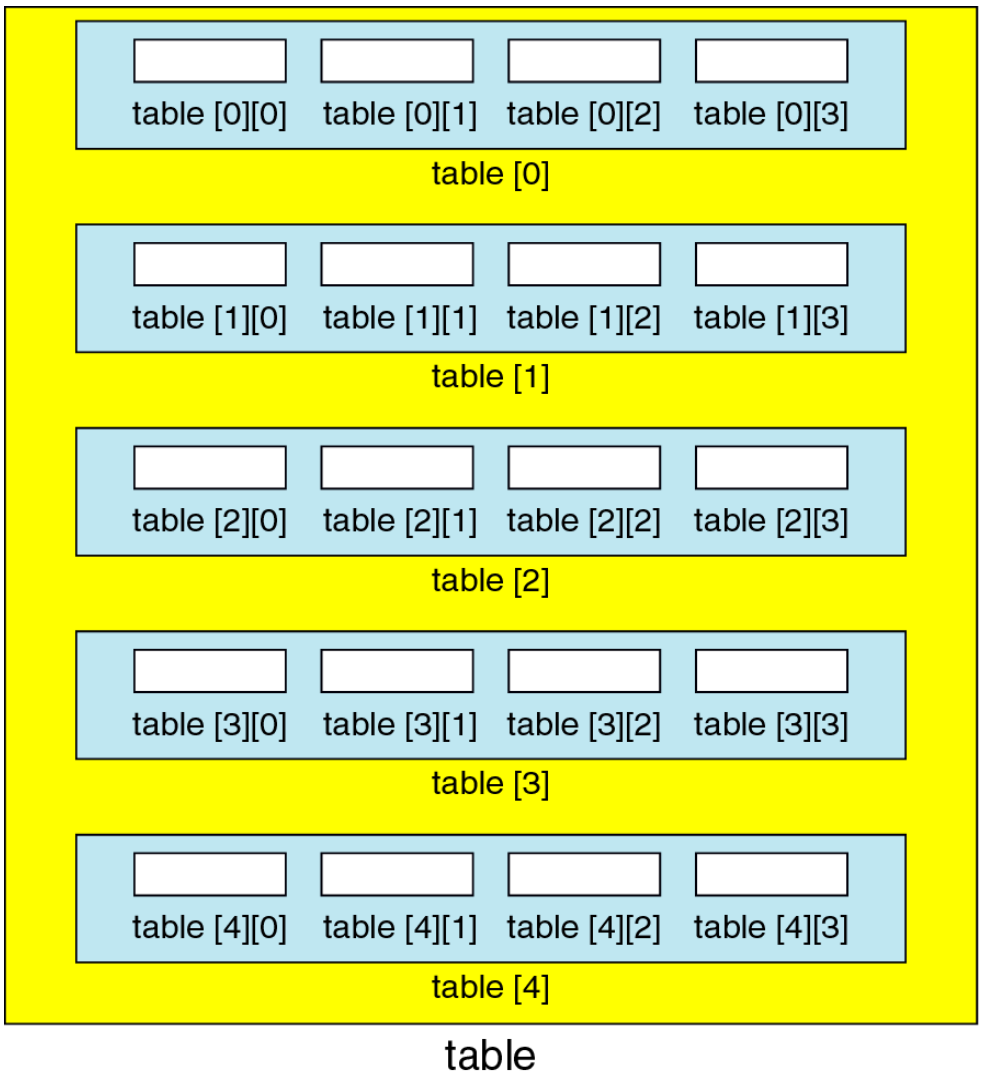
type variable-name[rows][columns] ;
• type : ชนิดของตัวแปรอาร์เรย์ที่จะสร้างขึ้น โดยพิจารณาจากชนิดของข้อมูลที่ต้องการจะเก็บในตัวแปรอาร์เรย์ • variable-name : ชื่อของตัวแปรอาร์เรย์ • rows : จำนวนแถวของตัวแปรอาร์เรย์ที่จะสร้างขึ้น • columns : จำนวนคอลัมน์ของตัวแปรอาร์เรย์ที่จะสร้างขึ้น

ตัวอย่างแผนผังการเรียงตัวกันของข้อมูลในตัวแปรอาร์เรย์ 2 มิติแสดงดังรูปที่ 9.6



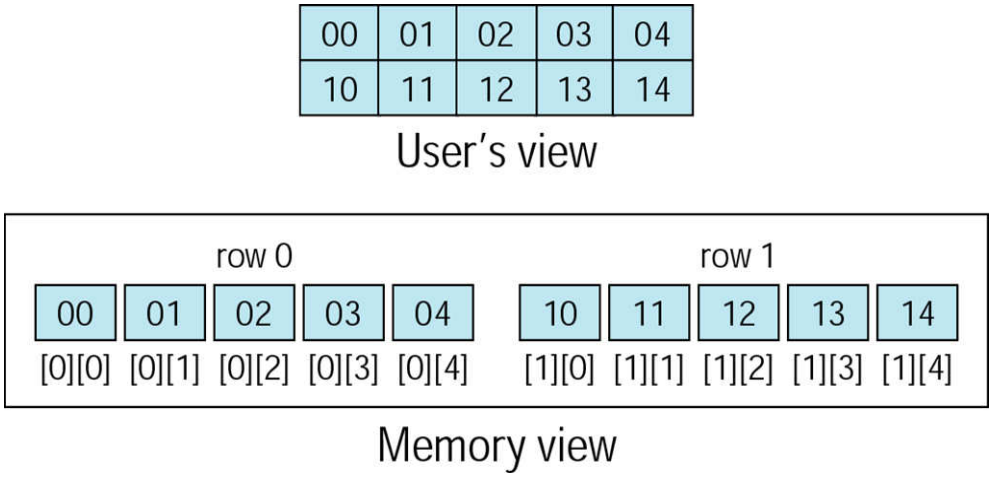
รูปที่ 9.6 การเรียงข้อมูลของตัวแปรอาร์เรย์ 2 มิติ

การอ้างอิงข้อมูลของตัวแปรอาร์เรย์ 2 มิติจะต้องทำการอ้างอิงทั้งอินเด็กซ์ของแถว และทั้งอินเด็กซ์ของคอลัมน์ ดังแสดงในรูปที่ 9.7 เป็นการประกาศตัวแปรอาร์เรย์ 2 มิติชื่อ table มีจำนวนแถวทั้งหมด 5 แถว และจำนวนคอลัมน์ทั้งหมด 4 แถว ถ้ากำหนดให้ตัวแปรอาร์เรย์ 2 มิติตัวนี้เก็บค่าจำนวนเต็ม จะสามารถประกาศด้วยคำสั่ง int table[5][4]



รูปที่ 9.7 โครงสร้างข้อมูลของตัวแปรอาร์เรย์ 2 มิติ table[5][4]

อย่างไรก็ตามก็เก็บข้อมูลของตัวแปรอาร์เรย์ 2 มิติในหน่วยความจำจะเป็นการเรียงตัวเหมือนกับอาร์เรย์ 1 มิติหลายตัวติดกัน ดังแสดงในรูปที่ 9.8



รูปที่ 9.8 การเก็บข้อมูลของตัวแปรอาร์เรย์ 2 มิติ

ตัวอย่างการใช้งานอาร์เรย์ 2 มิติ ให้พิจารณาตารางที่ 9.2 ซึ่งแสดงจำนวนนักศึกษาในชั้นปีต่างๆ ของคณะวิทยาศาสตร์ วิศวกรรมศาสตร์ อักษรศาสตร์ รัฐศาสตร์ และบัญชี

ตารางที่ 9.2 จำนวนนักศึกษาของคณะต่าง ๆ

คณะ	ชั้นปี			
	1	2	3	4
วิทยาศาสตร์	200	175	175	175
วิศวกรรมศาสตร์	400	390	385	385
อักษรศาสตร์	150	150	150	150
รัฐศาสตร์	100	100	98	98
บัญชี	200	200	195	195

จะเห็นได้ว่าข้อมูลของนักศึกษามีลักษณะเป็นตาราง 2 มิติดังนั้นจึงสามารถประยุกต์ใช้กับอาร์เรย์ 2 มิติได้เลย กำหนดให้ตัวแปรอาร์เรย์ 2 มิติที่จะให้เก็บข้อมูลใน ตารางที่ 9.2 ชื่อ student โดยจะมีจำนวนแถว 5 แถว และจำนวนคอลัมน์ 4 แถว ซึ่งสามารถสร้างตัวแปรได้ด้วยคำสั่งต่อไปนี้

```
int   student [5] [4];
```

การอ้างถึงตัวแปรอาร์เรย์ในอาร์เรย์ 2 มิติ จะต้องระบุด้วยชื่ออาร์เรย์และอินเด็กซ์ของแถวและอินเด็กซ์ของคอลัมน์ เช่น จำนวนนักศึกษาปี 3 ของคณะวิศวกรรมศาสตร์จะอยู่แถวที่ 2 และคอลัมน์ที่ 3 หรือก็คือ student [1] [2] ในอาร์เรย์ 2 มิติอินเด็กซ์ตัวแรกของแถวและคอลัมน์จะเป็นศูนย์ เช่น student[0][0] = 200 คือจำนวนนักศึกษาชั้นปีที่ 1 ของคณะวิทยาศาสตร์

ตัวอย่างโปรแกรมที่ 9.5

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3      int  student[5][4],  i,  j;
4      for(i  = 0;  i  <  5;  i++) {
5          for(j = 0;  j < 4; j++) {
6              printf("student[%d][%d] = ", i,  j);
7              scanf("%d\n", &student[i][j]);
8          }
9          printf("\n");
10     }
11 }
```

ผลการรัน

student [0][0] = ***200***
student [0][1] = ***175***
student [0][2] = ***175***
student [0][3] = ***175***

student [1][0] = ***400***
student [1][1] = ***390***
student [1][2] = ***385***
student [1][3] = ***385***

student [2][0] = ***150***
student [2][1] = ***150***
student [2][2] = ***150***
student [2][3] = ***150***

ผลการรัน(ต่อ)

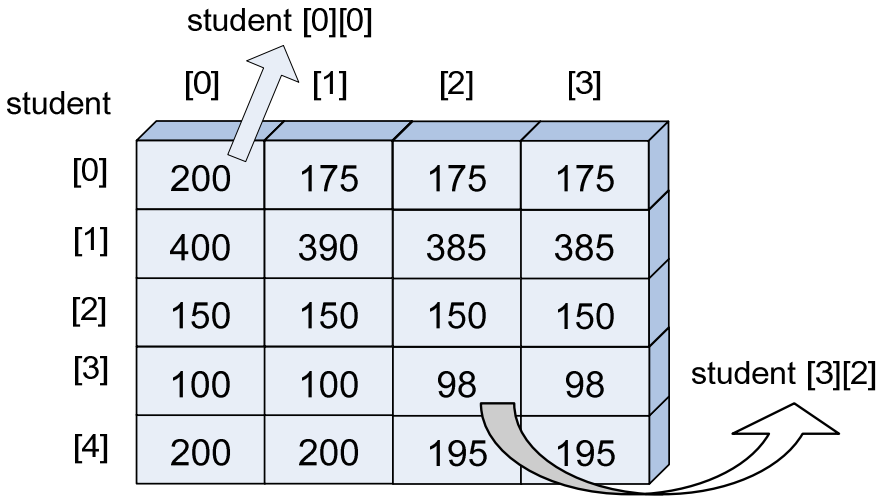
student [3][0] = ***100***
student [3][1] = ***100***
student [3][2] = ***98***
student [3][3] = ***98***

student [4][0] = ***200***
student [4][1] = ***200***
student [4][2] = ***195***
student [4][3] = ***195***

ตัวอย่างโปรแกรมที่ 9.5 เป็นการป้อนจำนวนนักศึกษาในตารางที่ 9.2 ไปเก็บในอาร์เรย์ student ซึ่งมีขนาด 5 x 4 = 20 การป้อนข้อมูลเริ่มจากจำนวนนักศึกษาของคณะวิทยาศาสตร์ ชั้นปีที่ 1, 2, 3, และ 4 แล้วจึงตามด้วยจำนวนนักศึกษาของคณะวิศวกรรมศาสตร์จนกระทั่งถึงจำนวนนักศึกษาชั้นปีที่ 4 ของคณะบัญชี อาร์เรย์ student จะมีข้อมูลดังรูปที่ 9.9 และรูปที่ 9.10 แสดงการอ้างอิงข้อมูลในตัวแปรอาร์เรย์ 2 มิติ

student	[0]	[1]	[2]	[3]
[0]	200	175	175	175
[1]	400	390	385	385
[2]	150	150	150	150
[3]	100	100	98	98
[4]	200	200	195	195

รูปที่ 9.9 ข้อมูลในตัวแปรอาร์เรย์ 2 มิติ student



รูปที่ 9.10 การอ้างอิงข้อมูลของตัวแปรอาร์เรย์ 2 มิติ

ตัวอย่างโปรแกรมที่ 9.6 และตัวอย่างโปรแกรมที่ 9.7 เป็นส่วนของโปรแกรมที่สามารถทำงานได้หลังจากมีการรับค่าตัวแปรอาร์เรย์ student จากแป้นพิมพ์แล้ว โดยทั้ง 2 ตัวอย่างโปรแกรมนี้จะเป็นตัวอย่างแสดงการหาจำนวนนักศึกษาของคณะอักษรศาสตร์ และการหานักศึกษาในชั้นปีที่ 3 ของตารางที่ 9.2 ตามลำดับ ส่วนโปรแกรมที่ 9.8 เป็นส่วนของโปรแกรมแสดงจำนวนนักเรียนทั้งหมดของทุกคณะและทุกชั้นปี

ตัวอย่างโปรแกรมที่ 9.6

```
1  int   sum = 0,  i;
2  for(i  = 0;  i < 4;  i++)
3           sum +=  student[2][i];
4  printf("The number of students = %d\n",  sum);
```

ผลการรัน

```
The number of students = 600
```

ตัวอย่างโปรแกรมที่ 9.7

```
1  int   sum = 0,  i;
2  for(i  = 0;  i < 5;  i++)
3           sum +=  student[i][2];
4  printf("The number of students = %d\n",  sum);
```

ผลการรัน

```
The number of students = 1003
```

ตัวอย่างโปรแกรมที่ 9.8

```
1  int   sum = 0,  i,  j;
2  for(i  = 0;  i < 5;  i++)
3       for(j = 0;  j < 4;  j++)
4           sum +=  student[i][j];
5  printf("The number of students = %d\n",  sum);
```

ผลการรัน

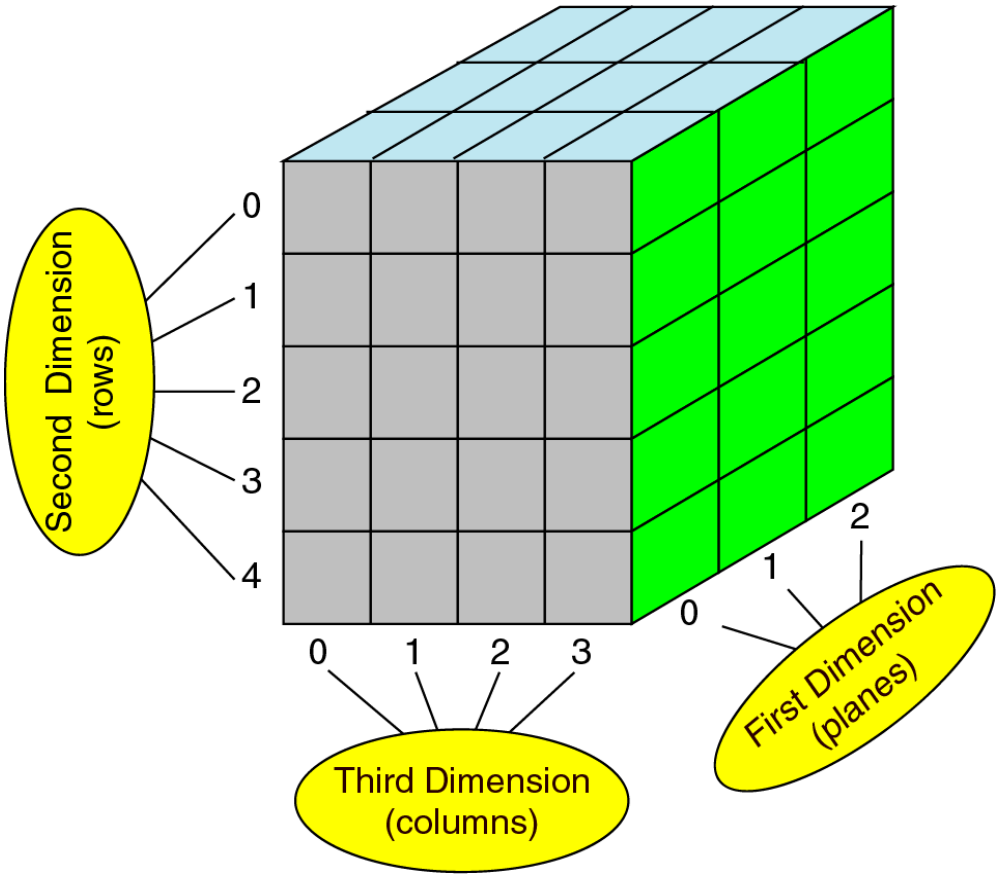
```
The number of students = 4071
```

9.4 อาร์เรย์ 3 มิติ

อาร์เรย์ 3 มิติ มีรูปแบบการกำหนดดังนี้

type variable-name[planes][rows][columns] ;
• type : ชนิดของตัวแปรอาร์เรย์ที่จะสร้างขึ้น โดยพิจารณาจากชนิดของข้อมูลที่ต้องการจะเก็บในตัวแปรอาร์เรย์ • variable-name : ชื่อของตัวแปรอาร์เรย์ • planes : จำนวนแถวในแนวลึกของตัวแปรอาร์เรย์ที่จะสร้างขึ้น • rows : จำนวนแถวของตัวแปรอาร์เรย์ที่จะสร้างขึ้น • columns : จำนวนคอลัมน์ของตัวแปรอาร์เรย์ที่จะสร้างขึ้น

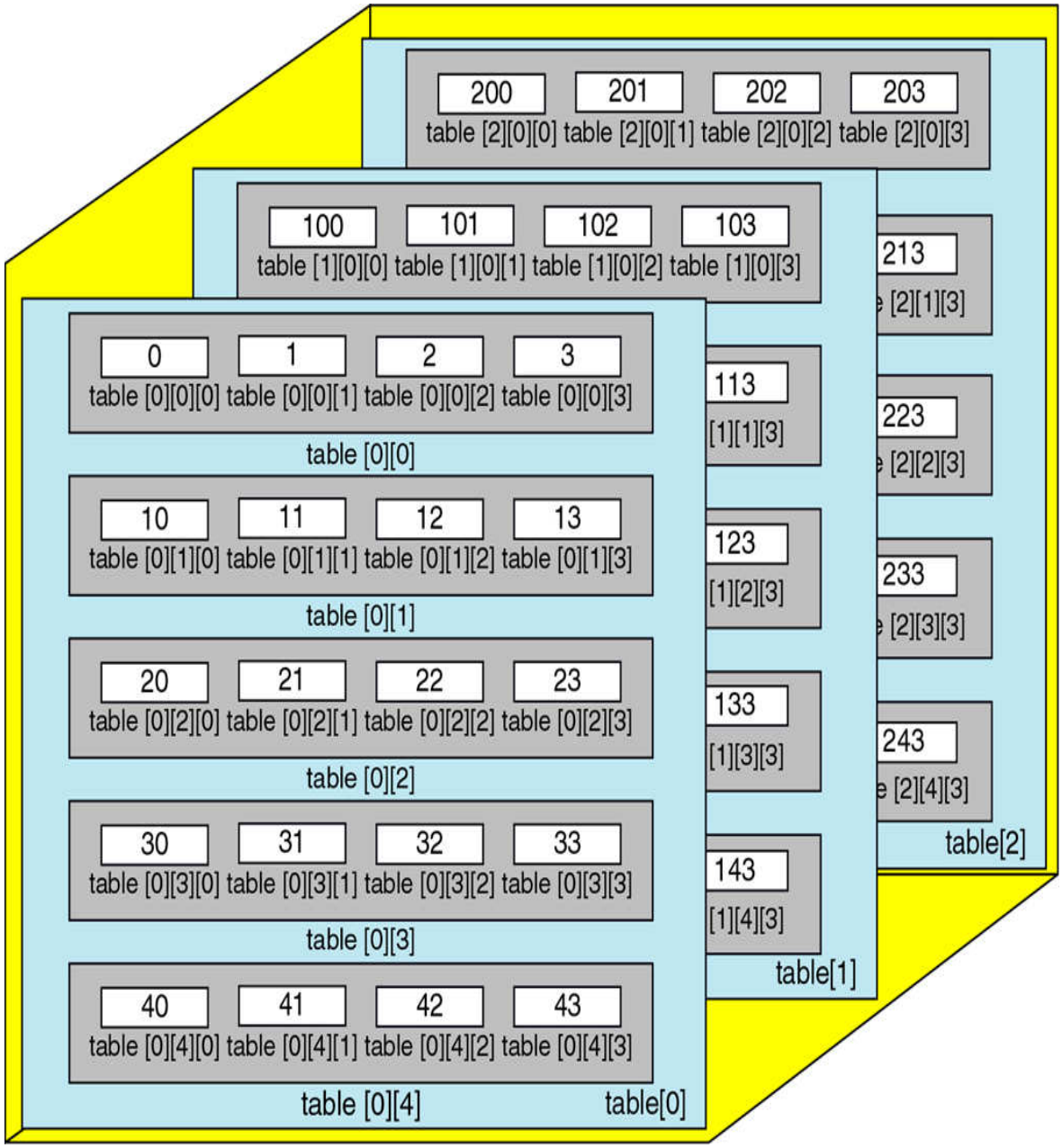
รูปที่ 9.11 แสดงโครงสร้างการเรียงตัวกันของตัวแปรอาร์เรย์ 3 มิติ โดยตัวแปรอาร์เรย์ 3 มิติมีจะมีขนาดทั้งหมดคือ planes x rows x columns



รูปที่ 9.11 โครงสร้างของตัวแปรอาร์เรย์ 3 มิติ

การอ้างอิงข้อมูลของตัวแปรอาร์เรย์ 3 มิติจะต้องทำการอ้างอิงทั้งอินเด็กซ์ของแถว อินเด็กซ์ของคอลัมน์ และอินเด็กของแถวในแนวลึกดังแสดงในรูปที่ 9.12 เป็นการประกาศตัวแปรอาร์เรย์ 3 มิติชื่อ table มีจำนวนแถวทั้งหมด 5 แถว จำนวนคอลัมน์ทั้งหมด 4 แถว และจำนวนแถวในแนวลึกทั้งหมด 3 แถว ถ้ากำหนดให้ตัวแปรอาร์เรย์ 3 มิติตัวนี้เก็บค่าจำนวนเต็ม จะสามารถประกาศด้วยคำสั่ง

```
int   table[3][5][4]
```



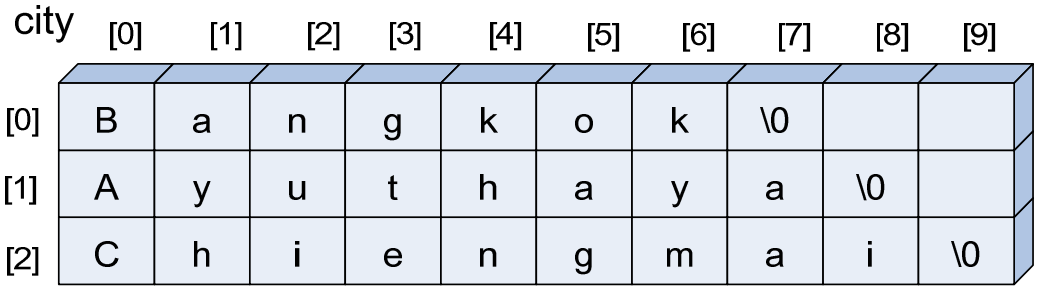
รูปที่ 9.12 การอ้างอิงข้อมูลในตัวแปรอาร์เรย์ 3 มิติ

9.5 ตัวแปรอาร์เรย์ประเภทข้อความหรือสตริงก์

ในภาษาซีข้อความต่างๆ จะเก็บในอาร์เรย์ชนิดตัวอักษรโดยขนาดของอาร์เรย์จะต้องมากกว่าจำนวนตัวอักษรในข้อความอย่างน้อย 1 ตัวอักษรสำหรับเก็บค่าตัวอักขระศูนย์ อาร์เรย์ซึ่งเก็บข้อมูลสตริงก์บางทีเรียกว่า ตารางสตริงก์ (string tables) ตัวอย่างเช่น

```
char  city [3] [10]  ;
```

ตัวแปรอาร์เรย์ city จะมีลักษณะเป็นอาร์เรย์ 2 มิติ โดยตารางสตริงก์จะโดยมี 3 แถวนอนและ 10 แถวตั้งโดยที่แถวนอนจะรับตัวอักษรได้ 9 ตัว เมื่อต้องการอ้างถึงตัวแปรอาร์เรย์ให้ระบุเฉพาะอินเด็กซ์แถวนอนเท่านั้น เช่น city [1] หมายถึงตัวแปรอาร์เรย์ตัวที่ 2 ซึ่งมีความยาวทั้งหมด 10 ตัวอักษร (นับตัวอักษรศูนย์ด้วย) ดังแสดงในรูปที่ 9.13



รูปที่ 9.13 การเก็บข้อมูลแบบสตริงก์

ตัวอย่างโปรแกรมที่ 9.9 จะกำหนดชื่อเมืองให้กับตัวแปรอาร์เรย์ city[0] city[1] และ city[2] จากนั้นจะพิมพ์ชื่อเมืองทั้งหมดบนจอภาพ

ตัวอย่างโปรแกรมที่ 9.9

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3       char  city[3][10];   int  i;
4       strcpy(city[0], “Bangkok”);
5       strcpy(city[1], “Ayuthaya”);
6       strcpy(city[2], “Chiengmai”);
7       for(i = 0;  i  <  3;  i++)
8            printf(“city[%d] = %s\n”, i, city[i]);
9  }
```

ผลการรัน

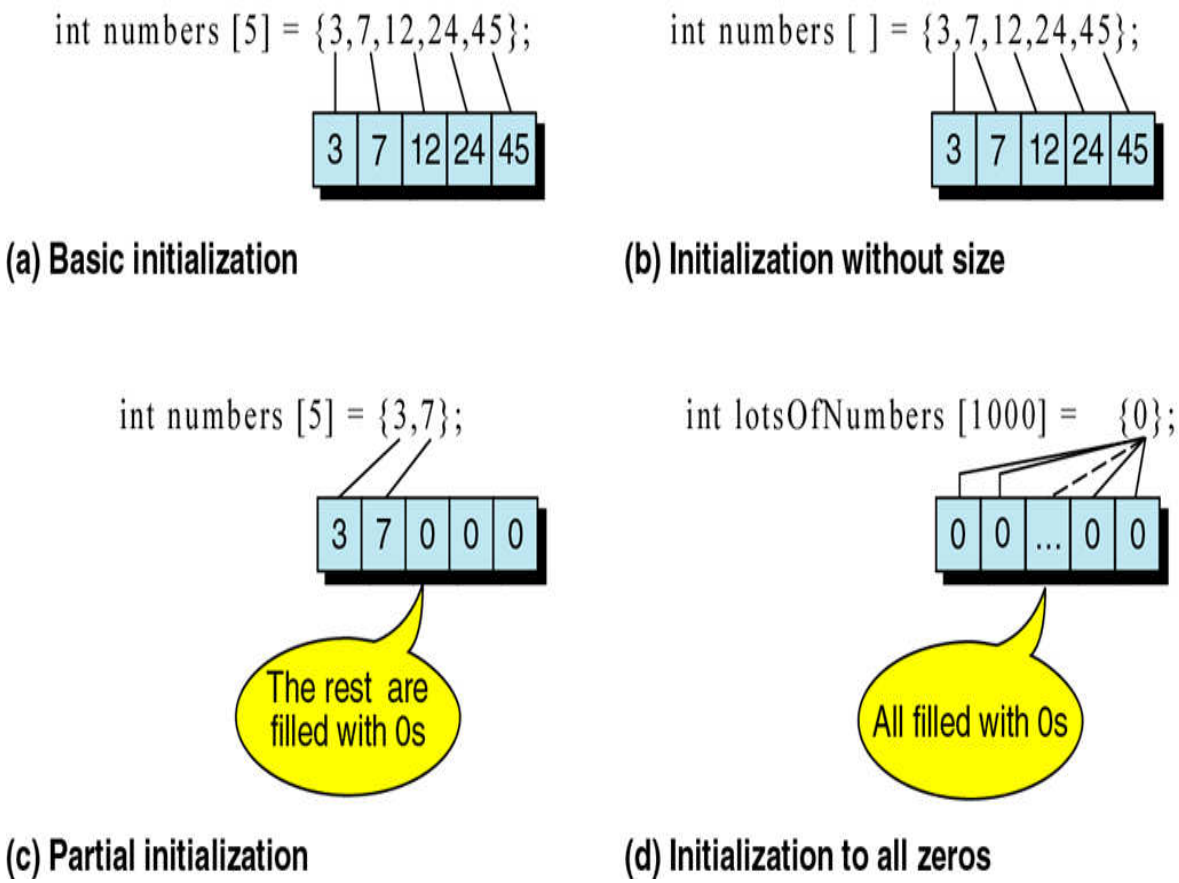
```
city[0]  = Bangkok
city[1]  = Ayuthaya
city[2]  = Chiengmai
```

ในโปรแกรมที่ 9.9 มีฟังก์ชั่น strcpy ( ) ซึ่งฟังก์ชั่นนี้จะนำค่าคงที่สตริงก์ไปเก็บในตัวแปรอาร์เรย์ city แต่ละตัว ฟังก์ชั่น strcpy ( ) จะกำหนดในไฟล์ #include<string.h> ดังนั้นจึงต้องกำหนดไฟล์ดังกล่าวในโปรแกรม

9.6 การกำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์

การกำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์สามารถกำหนดค่าเริ่มต้นได้ทั้งหมด 4 รูปแบบดังรูปที่ 9.14 ได้แก่

- แบบแจกแจงสมาชิก (Basic initialization)
- แบบไม่กำหนดขนาด (Initialization without size)
- แบบกำหนดค่าบางสมาชิก (Partial initialization)
- แบบกำหนดค่าทั้งหมดให้เป็น 0 (Initialization to all zeros)



รูปที่ 9.14 รูปแบบการกำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์

ตัวอย่างการกำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์ 1 มิติ เช่น

```
int  number [5]  =  {10 , 20 , 30 , 40 , 50};
```

อาร์เรย์ number จะมีค่าต่าง ๆ ดังนี้

number [0] = 10

number [1] = 20

number [2] = 30

number [3] = 40

number [4] = 50

สำหรับอาร์เรย์ที่เก็บข้อมูลเป็นตัวอักษร เช่น

```
char   character [3]  =  { 'a' , 'b' , 'c'} ;
```

อาร์เรย์ character จะมีค่าต่าง ๆ ดังนี้

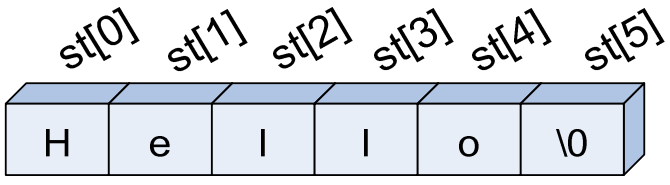
character [0] = a

character [1] = b

character [2] = c

สำหรับการกำหนดค่าเริ่มแรกให้กับอาร์เรย์สตริงก์ดังรูปที่ 9.15

```
char  st [6] = "Hello";
```



รูปที่ 9.15 การเก็บข้อมูลแบบสตริงก์

อาร์เรย์ที่มีมิติมากกว่า 1 มิติ สามารถกำหนดค่าเริ่มต้นได้เช่นกัน เช่น

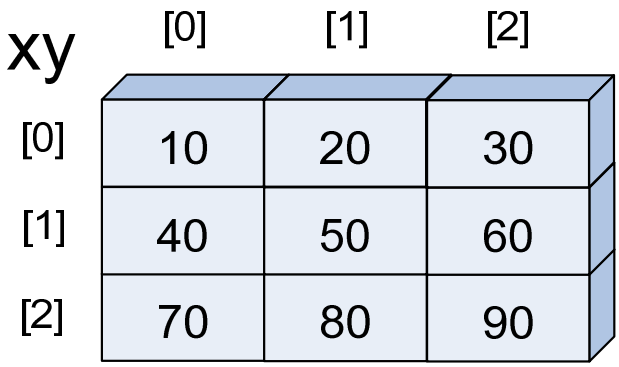
```
int  xy [3] [3]  =  {  {10 , 20 , 30},
                       {40 , 50 , 60},
                       {70 , 80 , 90}  };
```

อาร์เรย์ xy จะมีค่าต่าง ๆ ดังนี้ และแสดงดังรูปที่ 9.16

xy [0] [0] = 10 xy [0] [1] = 20 xy [0] [2] = 30

xy [1] [0] = 40 xy [1] [1] = 50 xy [1] [2] = 60

xy [2] [0] = 70 xy [2] [1] = 80 xy [2] [2] = 90



รูปที่ 9.16 การเก็บข้อมูลของตัวแปรอาร์เรย์ 2 มิติ

ตัวอย่างโปรแกรมที่ 9.10

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3       int  number[5] = { 10, 20, 30, 40, 50 };
4       int  i;
5
6       for(i = 0;  i  <  5;  i++)
7            printf("number[%d] = %d\n", i,  number[i]);
8  }
```

ผลการรัน

```
number[0]  =  10
number[1]  =  20
number[2]  =  30
number[3]  =  40
number[4]  =  50
```

ตัวอย่างโปรแกรมที่ 9.10 แสดงการกำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์ 1 มิติจากนั้นแสดงค่านั้นออกทางหน้าจอ

ตัวอย่างโปรแกรมที่ 9.11

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3      int  xy[3][3] = { {10, 20, 30},
4                        {40, 50, 60},
5                        {70, 80, 90} };
6      int   i,  j;
7
8      for(i = 0;  i  <  3;  i++)
9          for(j = 0; j < 3; j++)
10             printf("xy[%d][%d] = %d\n",  i,  j,  xy[i][j]);
11 }
```

ผลการรัน

```
xy[0][0]  =  10
xy[0][1]  =  20
xy[0][2]  =  30
xy[1][0]  =  40
xy[1][1]  =  50
xy[1][2]  =  60
xy[2][0]  =  70
xy[2][1]  =  80
xy[2][2]  =  90
```

ตัวอย่างโปรแกรมที่ 9.11 แสดงการกำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์ 2 มิติจากนั้นแสดงค่านั้นออกทางหน้าจอ

ตัวอย่างโปรแกรมที่ 9.12

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3      char  city[3][10] = { "Bangkok",
4                            "Ayuthaya",
5                            "Chiengmai"};
6      int   i;
7
8      for(i = 0;  i  <  3;  i++)
9              printf("city[%d] = %s\n",  city[i]);
10 }
```

ผลการรัน

```
city[0]  = Bangkok
city[1]  = Ayuthaya
city[2]  = Chiengmai
```

ตัวอย่างโปรแกรมที่ 9.12 ได้กำหนดค่าเริ่มแรกเป็นชื่อเมืองต่าง ๆ ให้กับอาร์เรย์สตริงก์ city ซึ่งมีตัวแปรอาร์เรย์ 3 ตัวโดยแต่ละตัวจะรับอักษรได้ไม่เกิน 10 ตัว

เท่าที่ผ่านมาการกำหนดค่าเริ่มแรกให้กับอาร์เรย์จะต้องกำหนดขนาดของอาร์เรย์เสมอ ในภาษาซีเราสามารถกำหนดค่าเริ่มแรกให้กับอาร์เรย์ได้โดยไม่ต้องกำหนดขนาดของอาร์เรย์ก็ได้ ซึ่งเราเรียกอาร์เรย์ในลักษณะนี้ว่าอาร์เรย์ไม่มีขนาด (Unsized array) อาร์เรย์ไม่มีขนาดนี้จะทำให้การเขียนโปรแกรมมีความยืดหยุ่นมาก โดยผู้เขียนโปรแกรมสามารถเปลี่ยนจำนวนข้อมูลได้ตามต้องการโดยไม่ต้องคำนึงถึงขนาดของอาร์เรย์ ตัวอย่างเช่น

```
int power [ ]  =  { 10 , 20 , 40 } ;
```

ในที่นี้ตัวแปรอาร์เรย์ 1 มิติประเภทจำนวนเต็ม power จะมีขนาดเป็น 3 โดยมี

- power [0] = 10
- power [1] = 20
- power [2] = 40

หรือในกรณีของอาร์เรย์แบบสตริงก์เช่น

```
char message [ ]  = " COMPUTER";
```

ในที่นี้ตัวแปรอาร์เรย์ 1 มิติประเภทตัวอักขระ message จะมีขนาดเป็น 9 โดยมี

- message[0] = C
- message[1] = O
- message[2] = M
- message[3] = P
- message[4] = U
- message[5] = T
- message[6] = E
- message[7] = R
- message[8] = \0

ตัวอักขระหลังจากจุดสิ้นสุดของข้อความภาษาซีจะใส่ '\0' เพื่อเป็นการปิดข้อมูลแบบสตริงก์ให้อัตโนมัติ

สำหรับอาร์เรย์ที่มีมิติมากกว่า 1 จะต้องระบุขนาดทั้งหมดยกเว้นขนาดของแถวที่อยู่ด้านซ้ายสุด (จำนวนของแถวนอน) เช่น

```
int  two_dim  [ ] [3]  =     {
                                    {10 , 20 ,  30},
                                    {40 , 50 , 60},
                                    {70 , 80 , 90}    };
```

จะเห็นว่าเราระบุเฉพาะจำนวนของแถวตั้งเท่านั้น ส่วนจำนวนของแถวนอนไม่ต้องกำหนด

ตัวอย่างโปรแกรมที่ 9.13

```
1  #include <stdio.h>
2  int main(int argc,  char **argv) {
3      char  city[ ][10] = { “Bangkok”,
4                            “Ayuthaya”,
5                            “Chiengmai”};
6      int   i;
7
8      for(i = 0;  i  <  3;  i++)
9              printf(“city[%d] = %s\n”,  city[i]);
10 }
```

ผลการรัน

```
city[0]  = Bangkok
city[1]  = Ayuthaya
city[2]  = Chiengmai
```

โปรแกรมที่ 9.13 จะให้ผลลัพธ์เหมือนโปรแกรมที่ 9.12 ต่างกันตรงที่ไม่ได้กำหนดขนาดทางแถวให้กับอาร์เรย์สตริงก์ city

บทที่ 10
พอยน์เตอร์

ข้อมูลส่วนใหญ่ที่นำเข้ามาใช้งานในโปรแกรมมักจะเป็นค่าคงที่ชนิดต่างๆ อย่างเช่น จำนวนเต็ม จำนวนจริง อักขระ หรือข้อความ แต่ในทางปฏิบัติแล้วยังมีข้อมูลอีกชนิดหนึ่งที่มีโอกาสใช้งานได้เช่นกัน นั่นก็คือตำแหน่งในหน่วยความจำ อย่างเช่น ในกรณีที่ต้องเขียนโปรแกรมควบคุมการทำงานของอุปกรณ์ประเภทไอซี หรือคอนโทรลเลอร์ โดยในบทนี้เราจะมารู้จักตัวแปรอีกชนิดหนึ่งในภาษาซีที่เรียกว่าตัวแปรพอยน์เตอร์ (Pointer) ซึ่งเป็นตัวแปรที่เก็บค่าตำแหน่งในหน่วยความจำ

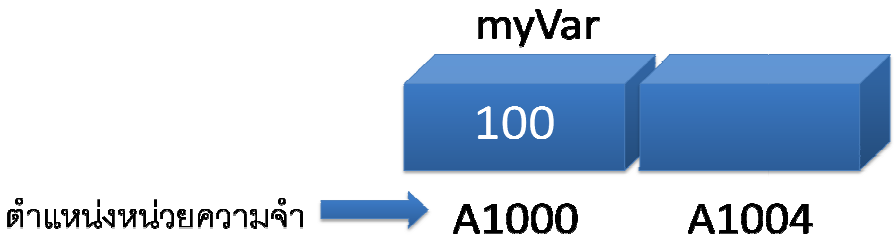
10.1 พอยน์เตอร์หรือตัวแปรพอยน์เตอร์

เมื่อประกาศสร้างตัวแปรชนิดใดก็ตามขึ้นมา ตัวแปรภาษาซีจะจัดการจองพื้นที่ในหน่วยความจำ (Memory address) ตำแหน่งที่ว่างเพื่อเป็นที่เก็บข้อมูลของตัวแปรนั้น ซึ่งเราจะไม่รู้ว่าตัวแปรที่สร้างขึ้นอยู่ตำแหน่งใดในหน่วยความจำ เนื่องจากเวลาเรียกใช้ข้อมูลเราจะเรียกผ่านชื่อของตัวแปร แต่ถ้าต้องการรู้ตำแหน่งในหน่วยความจำของตัวแปรที่สร้างขึ้น ก็สามารถทำได้โดยใช้ตัวแปรพอยน์เตอร์

ตัวแปรพอยน์เตอร์เป็นตัวแปรอีกชนิดในภาษาซี ที่มีความแตกต่างจากตัวแปรชนิดอื่นๆตรงที่ว่าตัวแปรชนิดอื่นที่เคยสร้างจะใช้สำหรับเก็บข้อมูลซึ่งเป็นค่าคงที่ แต่ตัวแปรชนิดพอยน์เตอร์จะใช้สำหรับเก็บตำแหน่งในหน่วยความจำของตัวแปรอื่น อย่างเช่น ถ้าต้องการสร้างตัวแปรชนิด int ชื่อ myVar สำหรับเก็บค่า 100 ในภาษาซีจะมีการเขียนในลักษณะดังต่อไปนี้

```
int   myVar = 100;
```

ตัวแปลภาษาซีจะจองพื้นที่ในหน่วยความจำให้ 4 bytes (สำหรับระบบ 32-bits) ตามขนาดของตัวแปรชนิด int ดังรูปที่ 10.1



รูปที่ 10.1 การเก็บข้อมูลของตัวแปรในภาษาซี

ตัวแปร myVar จะเก็บค่า 100 ไว้ ซึ่งถ้าต้องการสร้างตัวแปรพอยน์เตอร์สำหรับเก็บตำแหน่งของหน่วยความจำของตัวแปร myVar ขึ้นมาสมมุติว่าชื่อ pt_myVar ก็จะเก็บค่า A1000 ซึ่งเป็นตำแหน่งของหน่วยความจำของตัวแปร myVar ทั้งนี้ตำแหน่งในหน่วยความจำของคอมพิวเตอร์ (Memory address) จะอ้างถึงตัวเลขในระบบฐานสิบหกและข้อมูลจะถูกเก็บในหน่วยความจำในระบบเลขฐานสอง

10.2 การประกาศตัวแปรพอยน์เตอร์

การสร้างตัวแปรพอยน์เตอร์จะเหมือนกับการสร้างตัวแปรชนิดอื่น นั่นคือใช้คำสั่งประกาศสร้าง แต่เนื่องจากตัวแปรพอนย์เตอร์ ใช้สำหรับเก็บตำแหน่งในหน่วยความจำของตัวแปรอื่นที่สร้างขึ้นมาก่อนแล้ว ดังนั้นเราต้องพิจารณาก่อนว่าจะสร้างตัวแปรพอยน์เตอร์สำหรับชนิดข้อมูลประเภทใดด้วยรูปแบบต่อไปนี้

type * variable-name ;
• type : ชนิดของตัวแปรพอยน์เตอร์ โดยต้องพิจารณาว่าตัวแปรพอนย์เตอร์ที่จะสร้างขึ้นใช้เพื่อเก็บตำแหน่งในหน่วยความจำของตัวแปรชนิดใด • * : เครื่องหมายแสดงว่าตัวแปรที่สร้างขึ้นมีประเภทเป็นพอยน์เตอร์ • variable-name : ชื่อของตัวแปรพอยน์เตอร์

ตัวอย่างการเขียนคำสั่งเพื่อประกาศสร้างตัวแปรพอยน์เตอร์แสดงได้ดังต่อไปนี้

```
int  num;            //<---------- สร้างตัวแปรชนิด int ชื่อ num
int  *pt_int;        //<---------- สร้างตัวแปรพอยน์เตอร์ชนิด int เพื่อเก็บตำแหน่งใน
                     //            หน่วยความจำของตัวแปรชนิด int
float  salary;       //<---------- สร้างตัวแปรชนิด float ชื่อ salary
float  *pt_float;    //<---------- สร้างตัวแปรพอยน์เตอร์ชนิด float เพื่อเก็บตำแหน่งใน
                     //            หน่วยความจำของตัวแปรชนิด float
char  letter;        //<---------- สร้างตัวแปรชนิด char ชื่อ letter
char  *pt_char;      //<---------- สร้างตัวแปรพอยน์เตอร์ชนิด char เพื่อเก็บตำแหน่งใน
                     //            หน่วยความจำของตัวแปรชนิด char
```

10.3 เครื่องหมายของพอยน์เตอร์

สำหรับการทำงานใดๆ กับตัวแปรพอยน์เตอร์ ไม่ว่าจะเป็นการนำค่าตำแหน่งในหน่วยความจำเข้ามาเก็บหรือการไปดึงข้อมูลจากหน่วยความจำที่เก็บไว้ในตัวแปรพอยน์เตอร์มาแสดงผล การทำงานเหล่านี้จะต้องใช้เครื่องหมายที่ภาษาซีกำหนดไว้ซึ่งก็คือ เครื่องหมาย & และ เครื่องหมาย *

10.3.1 การแสดงตำแหน่งข้อมูลด้วย &

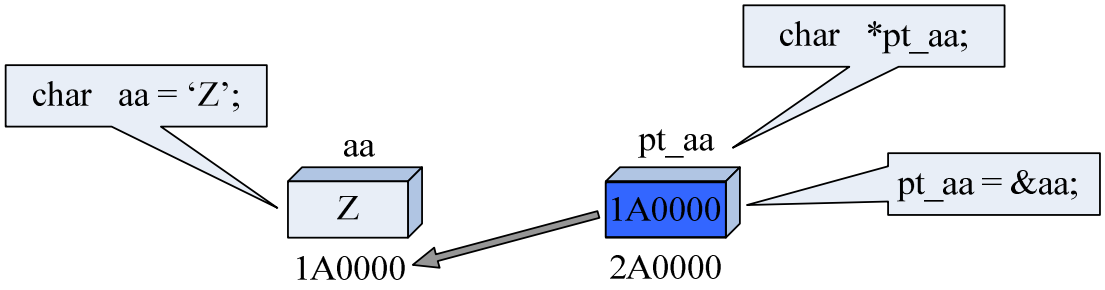
เมื่อสร้างตัวแปรชนิดพอยน์เตอร์ขึ้นมาแล้ว วิธีการที่จะนำค่าตำแหน่งในหน่วยความจำของตัวแปรใดๆ มาเก็บไว้ในตัวแปรพอยน์เตอร์นั้นต้องใช้เครื่องหมาย & (Unary) โดยรูปแบบการเขียน คำสั่งให้ใส่เครื่องหมาย & นำหน้าชื่อของตัวแปรที่เราต้องการตำแหน่ง แล้วกำหนดค่านั้นให้กับตัวแปรพอยน์เตอร์ ดังแสดงต่อไปนี้

pointer = &variable;
• pointer : ตัวแปรพอยน์เตอร์ที่จะใช้เก็บตำแหน่งในหน่วยความจำ • & : เครื่องหมายที่ใช้หาตำแหน่งในหน่วยความจำของตัวแปรที่อยู่หลังเครื่องหมาย • variable : ตัวแปรที่ต้องการหาตำแหน่งในหน่วยความจำ

ตัวอย่างการใช้เครื่องหมาย & เพื่อกำหนดค่าตำแหน่งในหน่วยความจำให้กับตัวแปรพอยน์เตอร์แสดงได้ดังนี้

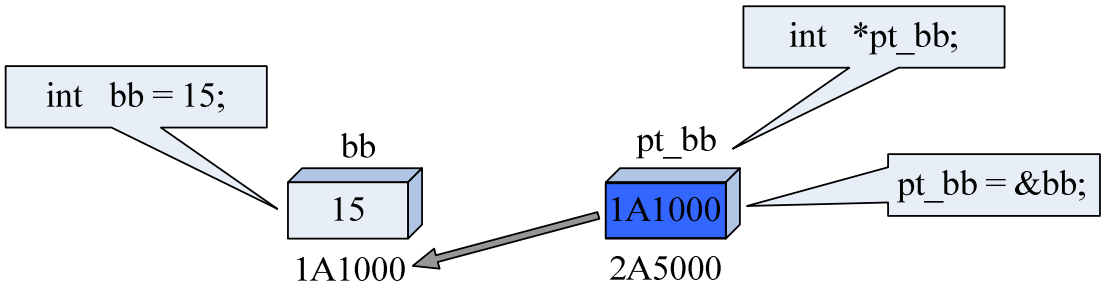
- ตัวแปรประเภทตัวอักขระ (char)

```
char   aa  = 'Z';
char   *pt_aa;
pt_aa  = &aa;
```



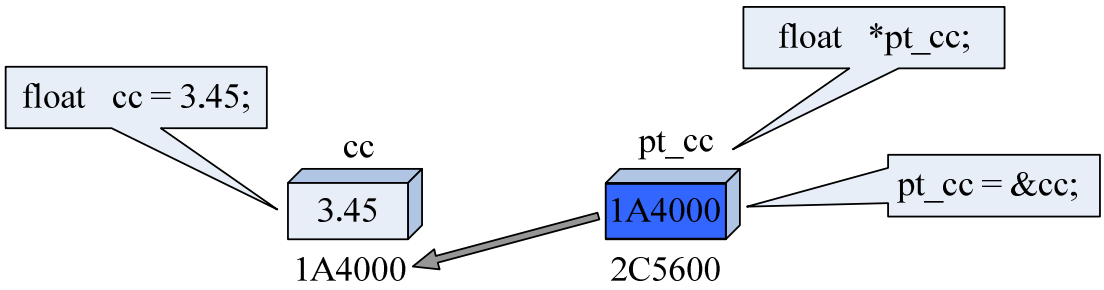
- ตัวแปรประเภทจำนวนเต็ม (int)

```
int  bb  = 15;
int   *pt_bb;
pt_bb  = &bb;
```



- ตัวแปรประเภทจำนวนจริง (float)

```
float  cc  = 3.45;
 float   *pt_cc;
pt_cc  = &cc;
```



สำหรับการเขียนโปรแกรม เพื่อให้แสดงค่าตำแหน่งหน่วยความจำของตัวแปรพอยน์เตอร์ออกทางหน้าจอจะมีรูปแบบในการแสดงผลเป็น %p ซึ่งใช้สำหรับการแสดงตำแหน่งหน่วยความจำในตัวแปรพอยน์เตอร์โดยเฉพาะ ดังแสดงในตัวอย่างโปรแกรมที่ 10.1

ตัวอย่างโปรแกรมที่ 10.1

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv) {
3     char   aa = 'Z';
4     int      bb = 15;
5     float   cc = 3.45;
6     char  *pt_aa;
7     int      *pt_bb;
8     float   *pt_cc;
9
10    pt_aa  =  &aa;
11    pt_bb  = &bb;
12    pt_cc  =  &cc;
13    printf("Address of AA =  %p\n", pt_aa);
14    printf("Address of BB =  %p\n", pt_bb);
15    printf("Address of CC =  %p\n", pt_cc);
16 }
```

ผลการรัน

```
Address of AA =  0028FF47
Address of BB =  0028FF40
Address of CC =  0028FF3C
```

ตัวอย่างโปรแกรมที่ 10.1 แสดงการใช้งานตัวแปรพอยน์เตอร์และแสดงค่าตำแหน่งหน่วยความจำ โดยบรรทัดที่ 3 - 5 จะเป็นการประกาศตัวแปรปกติประเภทตัวอักขระ aa, ประเภทจำนวนเต็ม bb, และประเภทจำนวนจริง cc โดยกำหนดค่าเริ่มต้นให้เป็น 'Z', 15 และ 3.45 ตามลำดับ ในขณะที่บรรทัดที่ 6 – 8 เป็นการประกาศตัวแปรประเภทพอยน์เตอร์ประเภทตัวขระ *pt_aa, ประเภทจำนวนเต็ม pt_bb และ จำนวนจริง pt_cc ตามลำดับ บรรทัดที่ 10 – 12 เป็นการใช้เครื่องหมาย & เพื่อนำค่าตำแหน่งหน่วยความจำที่ใช้เก็บค่าของตัวแปรปกติไปเก็บไว้ยังตัวแปรพอยน์เตอร์ จากนั้นแสดงค่าตำแหน่งหน่วยความจำที่เก็บไว้ในตัวแปรพอยน์เตอร์ออกทางหน้าจอ ผลการรันจะเห็นไว้ว่าตำแหน่งหน่วยความจำมีค่า 32 bits ซึ่งเป็นเพราะระบบที่ใช้มีขนาด 32 bits

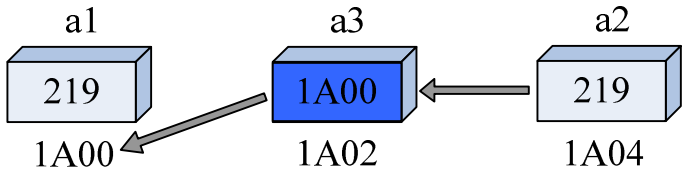
10.3.2 การแสดงค่าข้อมูลด้วย *

สำหรับการหาค่าของข้อมูลจากตำแหน่งในหน่วยความจำ ที่เก็บไว้ในตัวแปรพอยน์เตอร์ จะใช้เครื่องหมาย * (Indirect Operator) นำหน้าชื่อตัวแปรพอยน์เตอร์ซึ่งมี รูปแบบดังต่อไปนี้

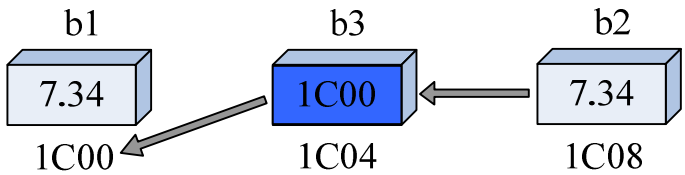
variable = * pointer;
• variable : ตัวแปรที่สร้างขึ้นเพื่อเก็บข้อมูลจากหน่วยความจำในตำแหน่งที่ตัวแปรพอยน์เตอร์เก็บค่าไว้ • * : เครื่องหมายที่ใช้หาค่าของข้อมูลจากตำแหน่งในหน่วยความจำ • pointer : ตัวแปรพอยน์เตอร์ที่จะใช้เก็บตำแหน่งในหน่วยความจำ

ตัวอย่างการใช้เครื่องหมาย * เพื่อหาค่าของข้อมูลในหน่วยความจำในตำแหน่งที่ตัวแปรพอยน์เตอร์ชี้อยู่ โดยแสดงด้วยส่วนของโปรแกรมและแผนภาพประกอบได้ดังนี้

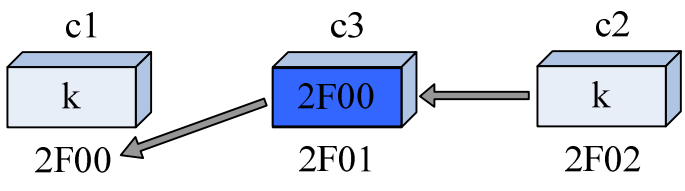
```
short  a1 = 219,  a2,  *a3;
a3  =  &a1;
a2  =  *a3;
```




```
float  b1 = 7.34, b2,  *b3;
b3 = &b1;
b2 = *b3;
```

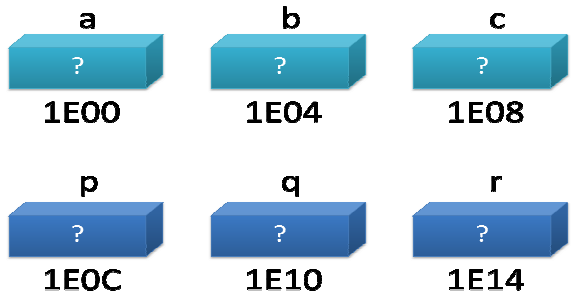


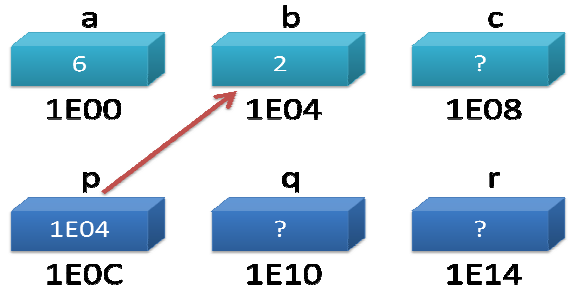

```
char c1 = 'k',  c2,  *c3;
c3 = &c1;
c2 = *c3;
```

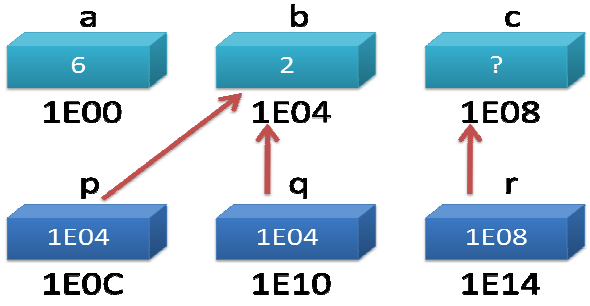


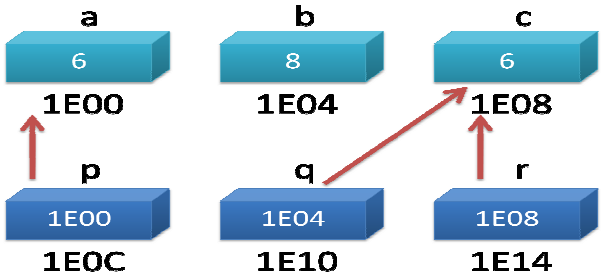
ตัวอย่างโปรแกรมที่ 10.2 และ 10.3 เป็นการเขียนโปรแกรมโดยใช้เครื่องหมาย * กับตัวแปรพอยน์เตอร์ เพื่อหาข้อมูลจากตำแหน่งในหน่วยความจำ

ตัวอย่างโปรแกรมที่ 10.2

1 2 3 4	#include <stdio.h> int main(int argc, char **argv) int a, b, c; int *p, *q, *r;	

5
6
7
	a = 6;
	b = 2;
	p = &b;
		

8 9	q = p; r = &c;	

10
11
12
13
	b = 8;
	p = &a;
	q = r;
	c = 6;
		

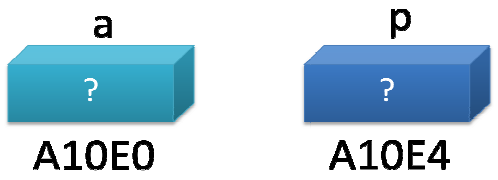
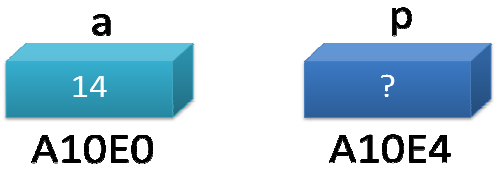
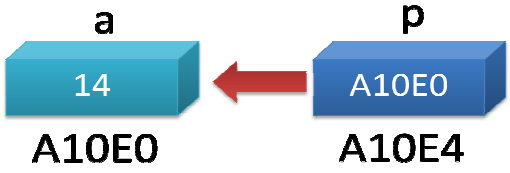
13 14	printf(“ %d %d %d \n”, a, b, c); printf (“ %d %d %d ”, *p, *q, *r);	

ผลการรัน

6 8 6

6 6 6

ตัวอย่างโปรแกรมที่ 10.3

1 2 3	#include <stdio.h> int main(int argc, char **argv) { int a, *p;	
4	a = 14;	
5	p = &a;	

6
7
8
	printf("%d %p\n", a, &a);
	printf("%p %d %d\n", &p, p, *p);
	}	

ผลการรัน

14 000A10E0

000A10E4 A10E0 14

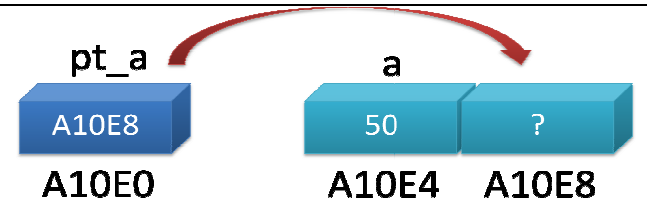
10.4 คณิตศาสตร์กับพอยน์เตอร์

ตัวแปรชนิดพอยน์เตอร์ก็สามารถดำเนินการทางคณิตศาสตร์ได้ แต่จำกัดอยู่แค่การบวกและการลบเท่านั้น โดยเครื่องหมายทางคณิตศาสตร์ที่ใช้กับตัวแปรพอยน์เตอร์ได้ ประกอบด้วย +, -, ++, -- และข้อมูลที่จะนำมาบวกหรือลบกับตัวแปรพอยน์เตอร์ต้องเป็นจำนวนเต็มเท่านั้น

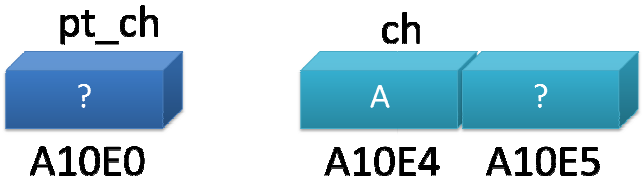
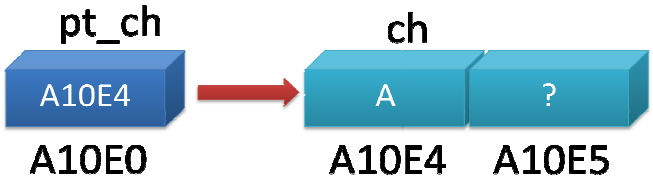
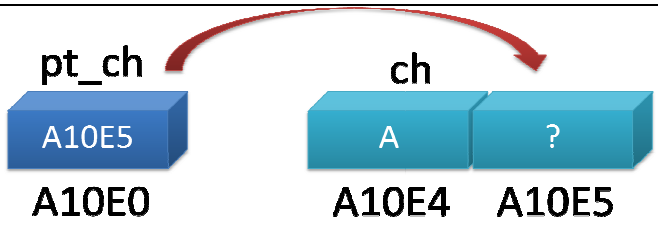
การบวกของตัวแปรพอยน์เตอร์จะหมายถึง การเพิ่มค่าแอสเดรดที่เก็บอยู่ในตัวแปรพอยน์เตอร์ หรือการเลื่อนพอยน์เตอร์ไปชี้ที่แอสเดรดสูงขึ้น การลดค่าแอสเดรดหรือการเลื่อนพอยน์เตอร์ไปชี้ที่แอสเดรดที่อยู่ต่ำลง ซึ่งก็จะขึ้นอยู่กับชนิดของตัวแปรที่พอยน์เตอร์ชี้อยู่ด้วย ดังแสดงในตัวอย่างโปรแกรมที่ 10.4 และตัวอย่างโปรแกรมที่ 10.5

ตัวอย่างโปรแกรมที่ 10.4

1
2
3
	#include <stdio.h>
	int main(int argc, char **argv) {
	int *pt_a, a = 50;
		

4	pt_a = &a;	
5 6	pt_a++; }	

ตัวอย่างโปรแกรมที่ 10.5

1 2 3	#include <stdio.h> int main(int argc, char **argv) { char *pt_ch, ch = 'A';	
4	pt_ch = &ch;	
5 6	pt_ch++; }	

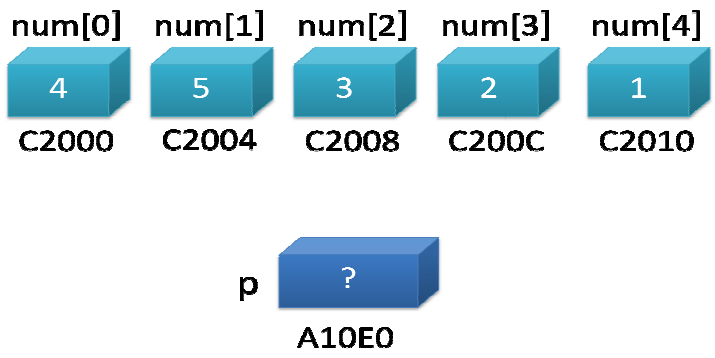
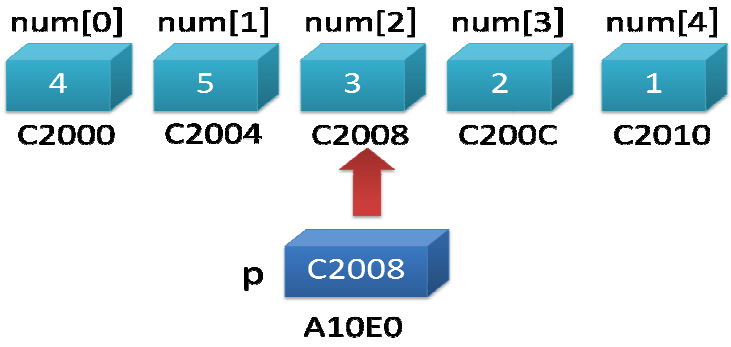
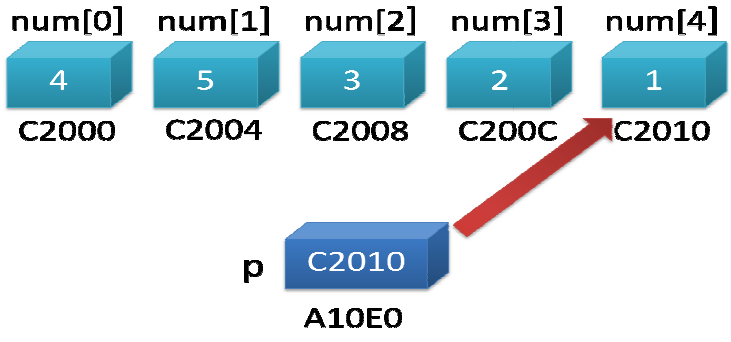
จากตัวอย่างโปรแกรมที่ 10.4 การเขียนคำสั่ง pt_a++ เป็นการกำหนดให้พอยน์เตอร์ pt_a เลื่อนไปยังแอดเดรสถัดไป แต่เนื่องจาก pt_a เป็นตัวแปรพอยน์เตอร์ที่ใช้เก็บตำแหน่งของตัวแปรชนิด int ซึ่งตัวแปรชนิด int มีขนาด 4 ไบต์ การเลื่อนพอยน์เตอร์ pt_a ไปยังตำแหน่งถัดไป จึงเป็นการบวกเพิ่ม 4 ตำแหน่ง

สำหรับตัวอย่างโปรแกรมที่ 10.5 การเขียนคำสั่ง pt_ch++ เป็นการกำหนดให้พอยน์เตอร์ pt_ch เลื่อนไปยังแอดเดรสถัดไป แต่เนื่องจาก pt_ch เป็นตัวแปรพอยน์เตอร์ที่ใช้เก็บตำแหน่งของตัวแปรชนิด char ซึ่งตัวแปรชนิด char มีขนาด 1 ไบต์ การเลื่อนพอยน์เตอร์ pt_ch ไปยังตำแหน่งถัดไป จึงเป็นการบวกเพิ่ม 1 ตำแหน่ง

10.5 ตัวแปรพอยน์เตอร์กับอาร์เรย์

จากที่ผ่านมาเป็นการใช้ตัวแปรพอยน์เตอร์กับตัวแปรชนิดพื้นฐานได้แก่ int, float และ char สำหรับกรณีของตัวแปรอาร์เรย์ก็สามารถใช้ตัวแปรพอนย์เตอร์ได้เช่นกัน เพื่อหาตำแหน่งในหน่วยความจำของตัวแปรอาร์เรย์ได้เช่นเดียวกันดังแสดงเป็นส่วนของโปรแกรมในตัวอย่างโปรแกรมที่ 10.6

ตัวอย่างโปรแกรมที่ 10.6

int num[5] = {4, 5, 3, 2, 1}; int *p;	
p = &num[2];	
p = &num[4];	

การใช้งานตัวแปรพอยน์เตอร์กับตัวแปรอาร์เรย์แสดงไว้ดังตัวอย่างโปรแกรมที่ 10.7

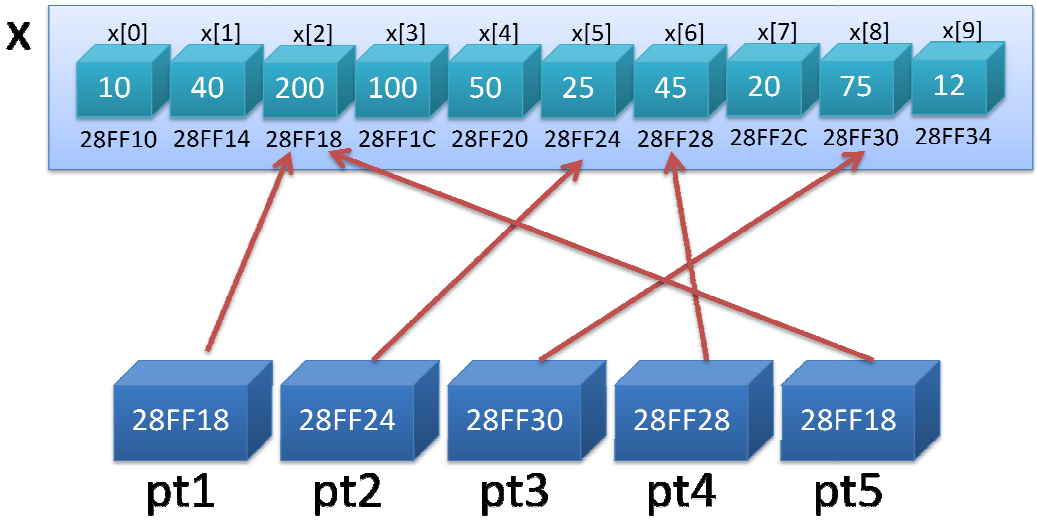
ตัวอย่างโปรแกรมที่ 10.7

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv) {
3     int  x[10] = {10,  40,  200,  100,  50,  25,  45,  20,  75, 12};
4     int   *pt1, *pt2,  *pt3,  *pt4,  *pt5;
5
6     pt1 = &x[2];
7     pt2 = &x[5];
8     pt3 = &x[8];
9     pt4 = &x[6];
10    pt5 = &x[2];
11    printf("Address of  x[2] =  %p\n", pt1);
12    printf("Address of  x[5] =  %p\n", pt2);
13    printf("Address of  x[8] =  %p\n", pt3);
14    printf("Value of x[6] = %d\n", *pt4);
15    printf("Value of x[2] = %d\n", *pt5);
16    x[3] = *pt5;
17    printf("Value of x[3] = %d\n", x[3]);
18 }
```

ผลการรัน

```
Address of  x[2] =  0028FF18
Address of  x[5] =  0028FF24
Address of  x[8] =  0028FF30
Value of x[6] = 45
Value of x[2] = 200
Value of x[3] = 200
```

ตัวอย่างโปรแกรมที่ 10.7 แสดงการใช้งานตัวแปรพอยน์เตอร์และตัวแปรอาร์เรย์ ในบรรทัดที่ 3 จะเห็นได้ว่ามีการประกาศตัวแปรประเภทจำนวนเต็มชื่อ x ที่มีการเก็บข้อมูลอยู่ในรูปแบบของอาร์เรย์ขนาด 10 จำนวน (x[10]) ซึ่งมีการกำหนดค่าเริ่มต้นให้กับตัวแปรอาร์เรย์คือ 10, 40, 200, 100, 50, 25, 45, 20, 75, และ 12 ตามลำดับ ในขณะที่บรรทัดที่ 4 มีการประกาศตัวแปรพอยน์เตอร์ประเภทจำนวนเต็ม 5 ตัวคือ pt1, pt2, pt3, pt4 และ pt5 ในบรรทัดที่ 6 – 10 มีการใส่ค่าของแอสเดรสในตำแหน่งของอาร์เรย์ให้กับตัวแปรพอยน์เตอร์ ดังรูปที่ 10.2



รูปที่ 10.2 ความสัมพันธ์ของตัวแปรพอยน์เตอร์และตัวแปรอาร์เรย์ในตัวอย่างโปรแกรมที่ 10.7

จากนั้นจากตัวอย่างโปรแกรมบรรทัดที่ 11 – 15 เป็นการแสดงค่าต่างๆของตัวแปร

- บรรทัดที่ 11 จะเป็นการแสดงผลค่าที่เก็บในตัวแปรพอยน์เตอร์ pt1 ผ่านฟังก์ชัน printf( ) จะเห็นจากรูป 10.2 ว่าค่าที่ออกจากผลการรันคือค่า 0028FF18
- บรรทัดที่ 12 แสดงค่าที่เก็บในตัวแปรพอยน์เตอร์ pt2 ซึ่งค่าที่ได้คือ 0028FF24
- บรรทัดที่ 13 แสดงค่าที่เก็บในตัวแปรพอยน์เตอร์ pt3 ซึ่งค่าที่ได้คือ 0028FF30
- บรรทัดที่ 14 เป็นการแสดงค่าในหน่วยความจำในตำแหน่งที่พอยน์เตอร์ชี้อยู่ ซึ่ง pt4 ชี้ไปยังตำแหน่งในหน่วยความจำ 0028FF28 ซึ่งก็คือ x[6] ดังนั้นค่าที่แสดงคือ 45
- บรรทัดที่ 15 เป็นการแสดงค่าในหน่วยความจำในตำแหน่งที่พอยน์เตอร์ชี้อยู่ ซึ่ง pt5 ชี้ไปยังตำแหน่งในหน่วยความจำ 0028FF18 ซึ่งก็คือ x[2] ดังนั้นค่าที่แสดงคือ 200

บรรทัดที่ 16 เป็นการให้ค่ากับตัวแปร x[3] ใหม่ โดยการนำค่าที่ตัวแปร pt5 ชี้อยู่มาทับค่าเก่า ซึ่งจะเห็นได้ว่า pt5 ชี้อยู่ที่ x[2] ซึ่งมีค่า 200 ดังนั้นจึงเป็นการนำค่า 200 มาใส่ x[3] บรรทัดที่ 17 เป็นการพิมพ์ค่าที่เก็บในตัวแปร x[3] ออกทางหน้าจอ ดังนั้นจึงเห็นว่า x[3] มีค่า 200

ตัวแปรพอยน์เตอร์กับตัวแปรอาร์เรย์มีความสัมพันธ์พิเศษอย่างหนึ่งนั่นคือ ถ้าเขียนชื่อตัวแปรอาร์เรย์โดยไม่ระบุอินเด็กซ์จะทำหน้าที่เหมือนพอยน์เตอร์ชี้ไปยังแอสเดรดของตัวแปรอาร์เรย์ตัวแรก

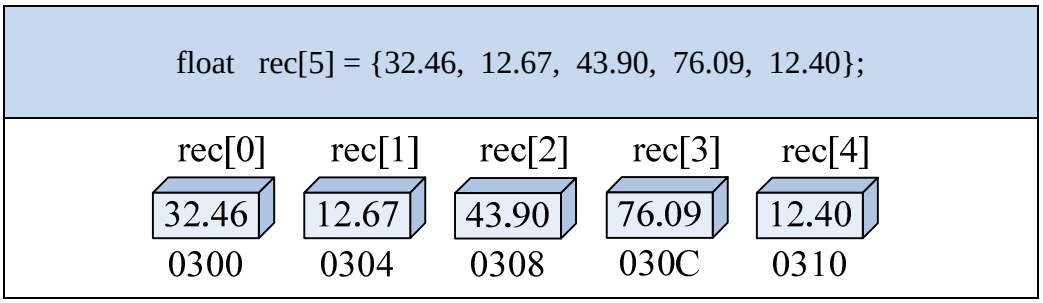
ตัวอย่างโปรแกรมที่ 10.8

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv) {
3     float  rec[5]  = {32.46,  12.67,  43.90,  76.09,  12.40};
4     float   *pt_rec,  new_rec;
5
6     pt_rec = rec;
7     printf("Address of array record = %p\n", pt_record);
8     new_rec = *(pt_record + 3);
9     printf("rec[3] = %f\n",  new_rec);
10    printf("rec[1] = %f\n",  *(pt_rec + 1));
11    pt_rec = &rec[2];
12    printf("Address of rec[2] = %p\n", pt_rec);
13  }
```

ผลการรัน

```
Address of  array record = 00000300
rec[3]  =  76.09
rec[1]  =  12.67
Address of rec[2] = 00000308
```

จากตัวอย่างโปรแกรมที่ 10.8 กำหนดให้ตัวแปรอาร์เรย์ของจำนวนจริง rec มีการใช้ตำแหน่งในหน่วยความจำดังต่อไปนี้

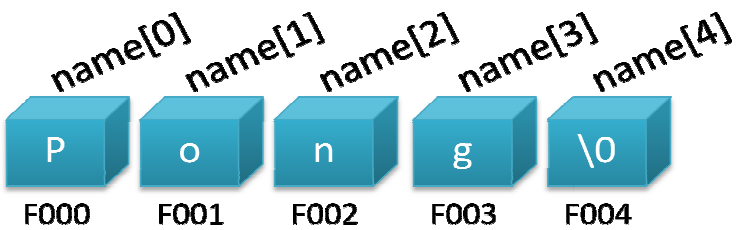


จากตัวอย่างโปรแกรมที่ 10.8 ถ้าเราอ้างถึงชื่อตัวแปรอาร์เรย์ rec โดยไม่ใส่อินเด็กซ์ใดๆต่อท้าย จะหมายถึงตำแหน่งในหน่วยความจำของ rec[0] ดังนั้นสำหรับตัวแปรอาร์เรย์เราสามารถกำหนดตำแหน่งในหน่วยความจำให้กับตัวแปรพอยน์เตอร์ได้ โดยไม่ต้องใช้เครื่องหมาย & ดังนี้

float *pt_rec; pt_rec = rec;	
float new_rec; new_rec = *(pt_rec+3); เนื่องจาก pt_rec เก็บค่าตำแหน่งในหน่วยความจำของ rec[0] ดังนั้น pt_rec+3 จะเป็นค่าตำแหน่งในหน่วยความจำของตัวแปรอาร์เรย์ตัวที่ 4 ซึ่งก็คือ rec[3]	
new_rec = *(pt_record+1);	
pt_rec = &rec[2]; ถ้าต้องการค่าตำแหน่งในหน่วยความจำ ของตัวแปรอาร์เรย์แต่ละตัว สามารถใช้เครื่องหมาย & ได้ตามปกติ	

10.6 ตัวแปรพอยน์เตอร์กับข้อความ

สำหรับข้อมูลชนิดข้อความตามปกติแล้วจะใช้วิธีการสร้างตัวแปรอาร์เรย์ชนิดอักขระขึ้นมาให้มีขนาดเท่ากับจำนวนอักขระทั้งหมดในข้อความบวกเพิ่มอีกหนึ่งสำหรับ \0 (อักขระว่าง) เพื่อบอกให้รู้ว่าเป็นข้อความ อย่างเช่น

char name[5] = “Pong”;	

การที่เขียนชื่อตัวแปรอาร์เรย์ที่เก็บข้อความโดยไม่ระบุอินเด็กซ์ อย่างเช่น name จะหมายถึงค่าตำแหน่งในหน่วยความจำเริ่มต้นของข้อความนั้น (ตัวแปรอาร์เรย์ตัวแรก) ดังนั้นถ้าจะใช้ตัวแปรพอยน์เตอร์กับตัวแปรอาร์เรย์ที่เก็บข้อความก็ไม่จำเป็นต้องใช้เครื่องหมาย & ในการหาค่าตำแหน่งในหน่วยความจำดังแสดงในตัวอย่างโปรแกรมที่ 10.9

ตัวอย่างโปรแกรมที่ 10.9

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv) {
3     char   name[5]  = “Pong”;
4     char  *pt_name;
5
6     pt_name =  name;
7     printf(“Address of name = %p\n”, pt_name);
8     printf(“name = %s\n,  name);
9     printf(“name = %s\n”, pt_name);
10 }
```

ผลการรัน

```
Address of  name  = 0000F000
name = Pong
name = Pong
```

การรับข้อมูลจากคีย์บอร์ดด้วยฟังก์ชั่น scanf( ) ตามปกติแล้วจะต้องใส่เครื่องหมาย & นำหน้าชื่อของตัวแปรเพื่อระบุตำแหน่งในหน่วยความจำของตัวแปรนั้น แต่สำหรับข้อมูลชนิดข้อความจะมีความพิเศษกว่าตัวแปลชนิดอื่นตรงที่ไม่ต้องใส่เครื่องหมาย & นำหน้าชื่อของตัวแปร ซึ่งเราก็ได้รู้เหตุผลแล้วว่าชื่อของตัวแปรที่เก็บข้อมูลชนิดข้อความ หมายถึงตำแหน่งเริ่มต้นในหน่วยความจำของข้อความนั้นเอง

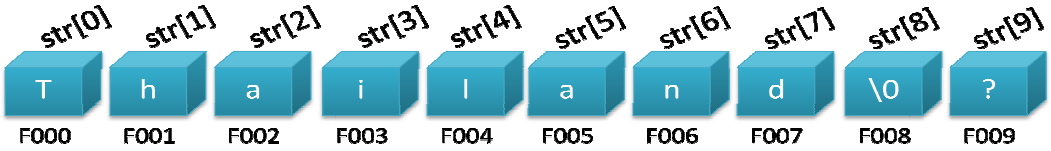
ตัวอย่างโปรแกรมที่ 10.10

```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv) {
3     char   str[10]  = “Thailand”;
4     char  *pt;
5     pt =  str;
6     printf(“%c\n”, pt[2]);
7     printf(“%c\n”, *(pt+1));
8     printf(“%c\n”,*(pt+4));
9  }
```

ผลการรัน

```
a
h
l
```

ตัวอย่างโปรแกรมที่ 10.10 แสดงการใช้งานตัวแปรพอยน์เตอร์กับตัวขระในข้อความ ซึ่งจะแสดงให้เห็นว่าตัวแปรพอยน์เตอร์สามารถอ้างอิงถึงตัวอักษรในข้อความได้ โดยในบรรทัดที่ 3 มีการประกาศตัวแปรแบบข้อความชื่อ str กำหนดค่าเริ่มต้นคือ Thailand ซึ่งการเก็บข้อมูลจะเป็นดังรูปที่ 10.3



รูปที่ 10.3 การเก็บข้อมูลแบบข้อความหรือสตริงก์

- บรรทัดที่ 5 มีการกำหนดค่าให้กับตัวแปรพอยน์เตอร์ เมื่อเป็นชื่อตัวแปรประเภทอาร์เรย์จะหมายถึงแอดเดรสของสมาชิกตัวแรกของอาร์เรย์ซึ่งก็หมายความว่า พอยน์เตอร์จะชี้ไปยัง str[0]
- บรรทัดที่ 6 แสดงค่าตัวอักขระที่อยู่ในตำแหน่งของ pt[2] จริงๆแล้วก็มีความหมายเหมือนกับ *(pt + 2) ซึ่งก็คือค่าที่อยู่ในตำแหน่งของ str[2] ที่มีค่า ‘a’
- บรรทัดที่ 7 – 8 แสดงค่าตัวอักขระที่อยู่ในตำแหน่ง *(pt+1) และ *(pt+4) คือการเลื่อนตำแหน่งพอยน์เตอร์ไป 1 และ 4 ค่าตามลำดับ ซึ่งก็คือ str[1] และ str[4] ที่มีค่า ‘h’ และ ‘l’ ตามลำดับ

10.7 อาร์เรย์ของพอยน์เตอร์

ในกรณีที่ต้องการใช้ตัวแปรพอยน์เตอร์ชนิดเดียวกันเป็นจำนวนมาก แทนที่จะประกาศสร้างตัวแปรพอยน์เตอร์เหล่านั้นทีละตัว เราสามารถสร้างตัวแปรอาร์เรย์ของพอยน์เตอร์ขึ้นมาใช้งานได้เช่นเดียวกับตัวแปรอาร์เรย์ของตัวแปรชนิดอื่นๆ ด้วยรูปแบบดังแสดงต่อไปนี้

type *variable[n];
• type : ชนิดตัวแปร Indirect pointer ที่จะสร้างขึ้น ซึ่งต้องเป็นชนิดเดียวกันกับตัวแปรพอยน์เตอร์ที่จะหาค่าตำแหน่งในหน่วยความจำ • * : เครื่องหมายที่แสดงว่าเป็นตัวแปรพอยน์เตอร์ • variable : ชื่อของตัวแปรอาร์เรย์ของพอยน์เตอร์ที่จะสร้างขึ้น • [n] : ขนาดของอาร์เรย์ของพอยน์เตอร์ที่สร้างขึ้น

ตัวอย่างโปรแกรมที่ 10.11

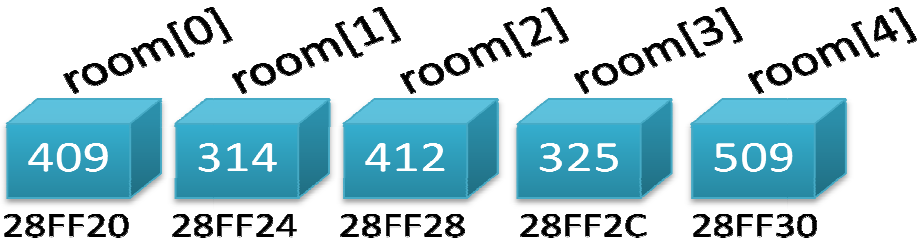
```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv) {
3     int  room[5]  = {409,  314,  412,  325,  509};
4     int  *pt[5],  i;
5     for(i = 0;  i < 5;  i++)
6          pt[i]  = &room[i];
7
8    for(i  = 0;  i <  5;  i++)
9         printf("Address of room[%d] = %p\n", i,  pt[i]);
10 }
```

ผลการรัน

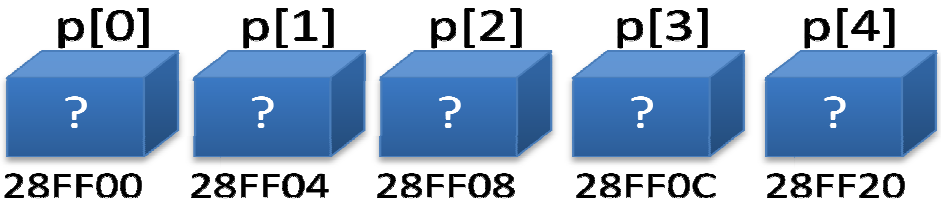
```
Address of room[0] = 0028FF20
Address of room[1] = 0028FF24
Address of room[2] = 0028FF28
Address of room[3] = 0028FF2C
Address of room[4] = 0028FF30
```

จากตัวอย่างโปรแกรมที่ 10.11 แสดงการสร้างตัวแปรอาร์เรย์ของพอยน์เตอร์และการใช้งาน โดยเมื่อสั่งรัน โปรแกรมจะแสดงตำแหน่งในหน่วยความจำของสมาชิกแต่ละตัวในอาร์เรย์ room ที่เก็บในอาร์เรย์ของพอยน์เตอร์ pt ขึ้นบนหน้าจอ การทำงานของโปรแกรมสามารถอธิบายด้วยรูปประกอบได้ดังนี้

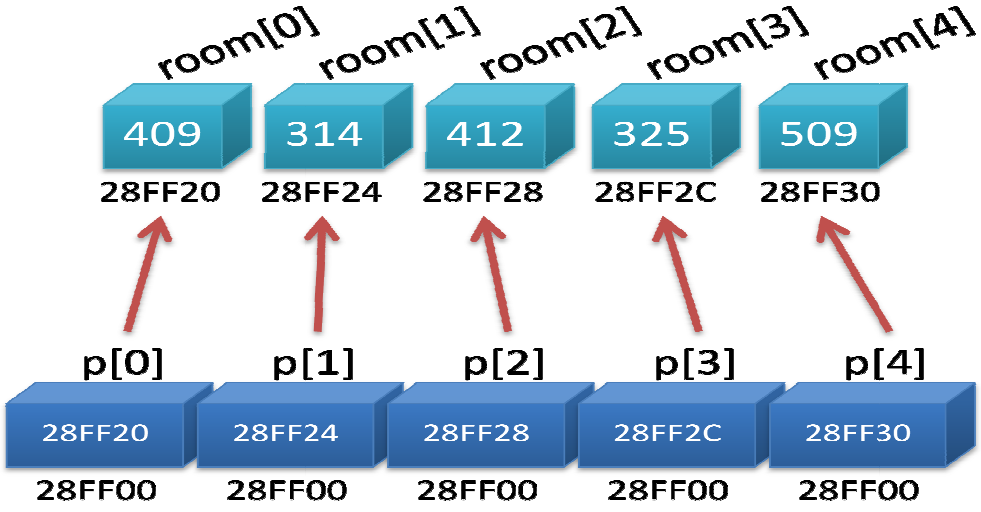
```
int  room[5] = {409, 314, 412, 325, 509};
```




```
int   *pt_r[5];
```




```
for(i = 0;  i < 5;  i++)
        pt[i]  = &room[i];
```



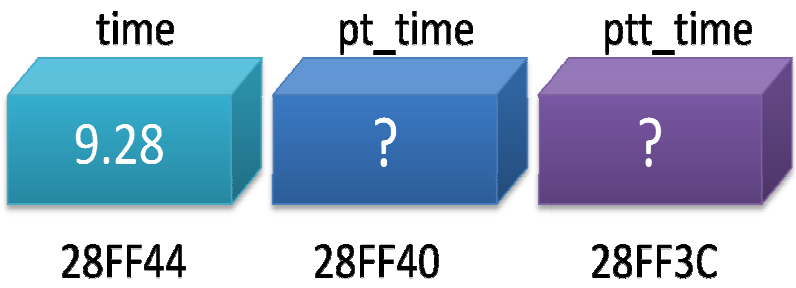
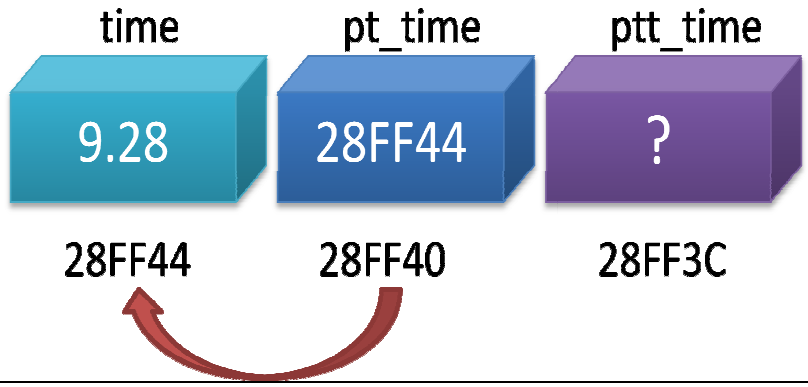
10.8 พอยน์เตอร์ซ้อนพอยน์เตอร์

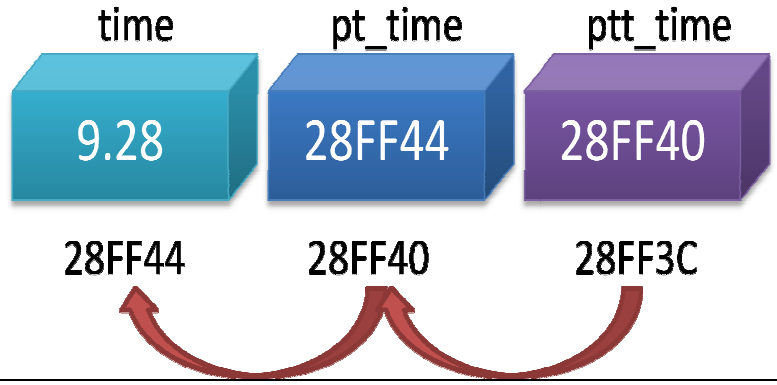
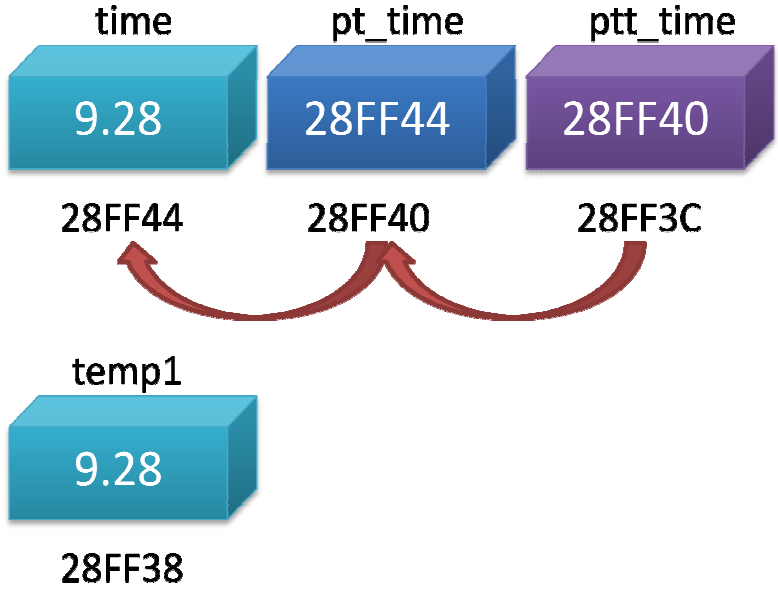
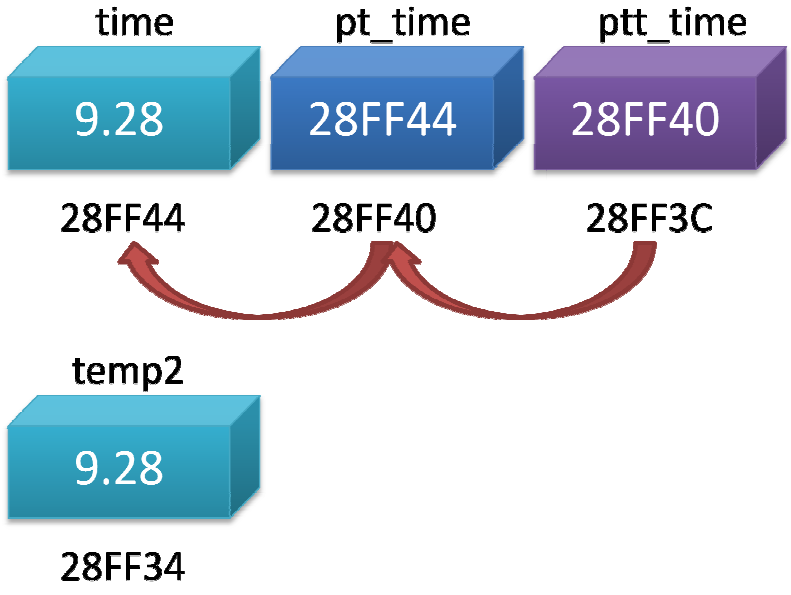
ถึงแม้ว่าตัวแปรพอยน์เตอร์จะเก็บค่าตำแหน่งในหน่วยความจำของตัวแปรชนิดอื่นๆ แต่ตัวแปรพอยน์เตอร์ก็ถือเป็นตัวแปรชนิดหนึ่งเช่นกัน ดังนั้นเมื่อสร้างตัวแปรพอยน์เตอร์ขึ้นมา ตัวแปลภาษาซีก็จะจองพื้นที่ในหน่วยความจำสำหรับตัวแปรพอยน์เตอร์เหล่านั้นเช่นเดียวกับตัวแปรชนิดอื่นๆ ถ้าต้องการตำแหน่งในหน่วยความจำของตัวแปรพอยน์เตอร์ ให้เรียกใช้ตัวแปรชนิดที่เรียกว่า Indirect pointer หรือตัวแปรพอยน์เตอร์ซ้อนพอยน์เตอร์ ซึ่งมีรูปแบบดังนี้

type **variable;
• type : ชนิดตัวแปร Indirect pointer ที่จะสร้างขึ้น ซึ่งต้องเป็นชนิดเดียวกันกับตัวแปรพอยน์เตอร์ที่จะหาค่าตำแหน่งในหน่วยความจำ • ** : เครื่องหมายที่แสดงว่าเป็นตัวแปร Indirect pointer • variable : ชื่อของตัวแปร indirect pointer ที่จะสร้างขึ้น

ตัวอย่างโปรแกรมที่ 10.12 แสดงส่วนของโปรแกรมและแผนภาพแสดงการทำงานของตัวแปรพอยน์เตอร์ซ้อนพอยน์เตอร์

ตัวอย่างโปรแกรมที่ 10.12

float time = 9.28 ; float *pt_time; float **pt_time;	
pt_time = &time;	

ptt_time = &pt_time;	
float temp 1; temp1 = *pt_time;	
float temp 2; temp2 = **ptt_time	

ตัวอย่างโปรแกรมที่ 10.13

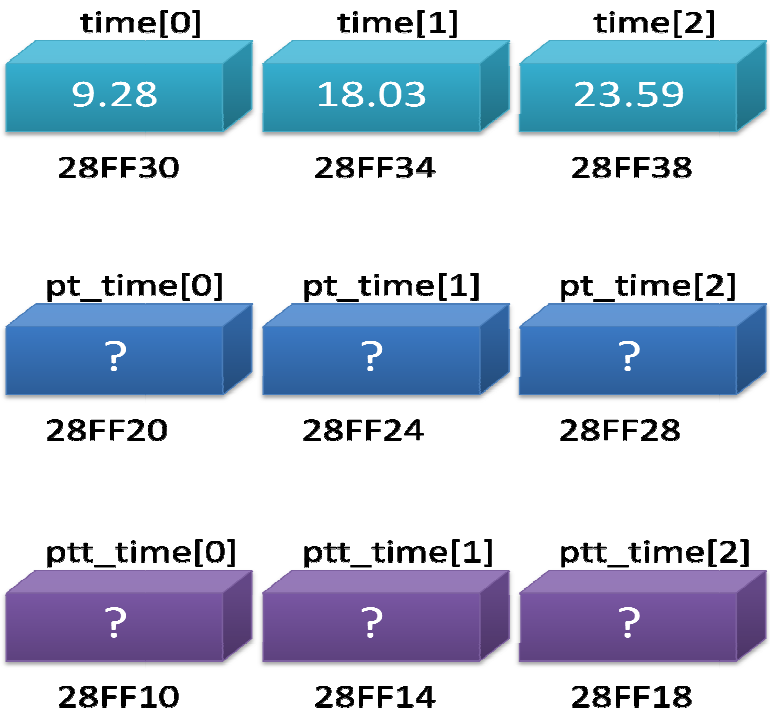
```
1  #include <stdio.h>
2  int  main(int  argc,  char **argv) {
3     float   time[3]  = {9.28,  18.03,  23.59};
4     float  *pt_time[3];
5     float   **ptt_time[3];
6
7     pt_time[0]  = &time[0];
8     pt_time[1]  = &time[1];
9     pt_time[2]  = &time[2];
10
11    ptt_time[0] = &pt_time[0];
12    ptt_time[1] = &pt_time[1];
13    ptt_time[2] = &pt_time[2];
14
15    printf("Address of pt_time[0]  = %p\n", i,  ptt_time[0]);
16    printf("Address of pt_time[1]  = %p\n", i,  ptt_time[1]);
17    printf("Address of pt_time[2]  = %p\n", i,  ptt_time[2]);
18
19    printf("time[2]  = %f\n",  **ptt_time[2]);
      printf("time[1]  = %f\n",  *pt_time[1]);
   }
```

ผลการรัน

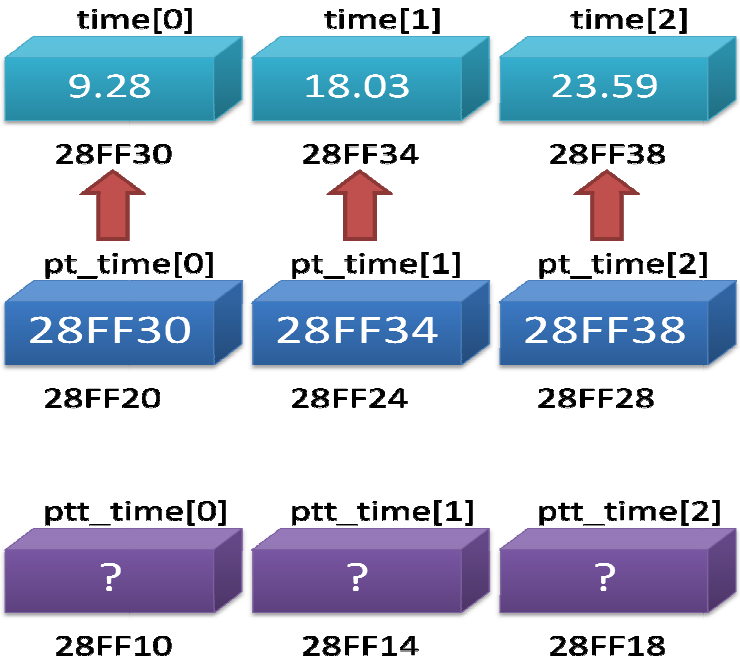
```
Address of pt_time[0]  = 0028FF20
Address of pt_time[1]  = 0028FF24
Address of pt_time[2]  = 0028FF28
time[2]  = 23.590000
time[1]  = 18.030001
```

ตัวอย่างโปรแกรมที่ 10.13 เป็นการเขียนโปรแกรมโดยสร้างตัวแปรอาร์เรย์ของพอยน์เตอร์ และตัวแปรอาร์เรย์ของ indirect pointer เพื่อหาค่าตำแหน่งในหน่วยความจำของตัวแปรพอยน์เตอร์ดังแสดงต่อไปนี้

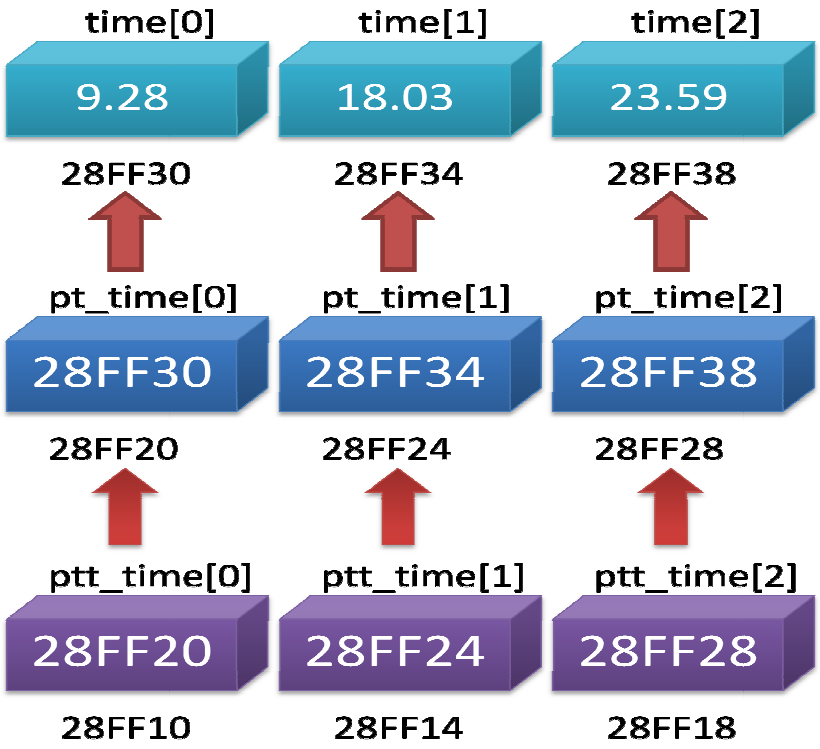
```
float   time[3]  = {9.28,  18.03,  23.59},  *pt_time[3],  **ptt_time[3];
```




```
pt_time[0]  = &time[0];   pt_time[1]  = &time[1];   pt_time[2]  = &time[2];
```



ptt_time[0] = &pt_time[0]; ptt_time[1] = &pt_time[1]; ptt_time[2] = &pt_time[2];



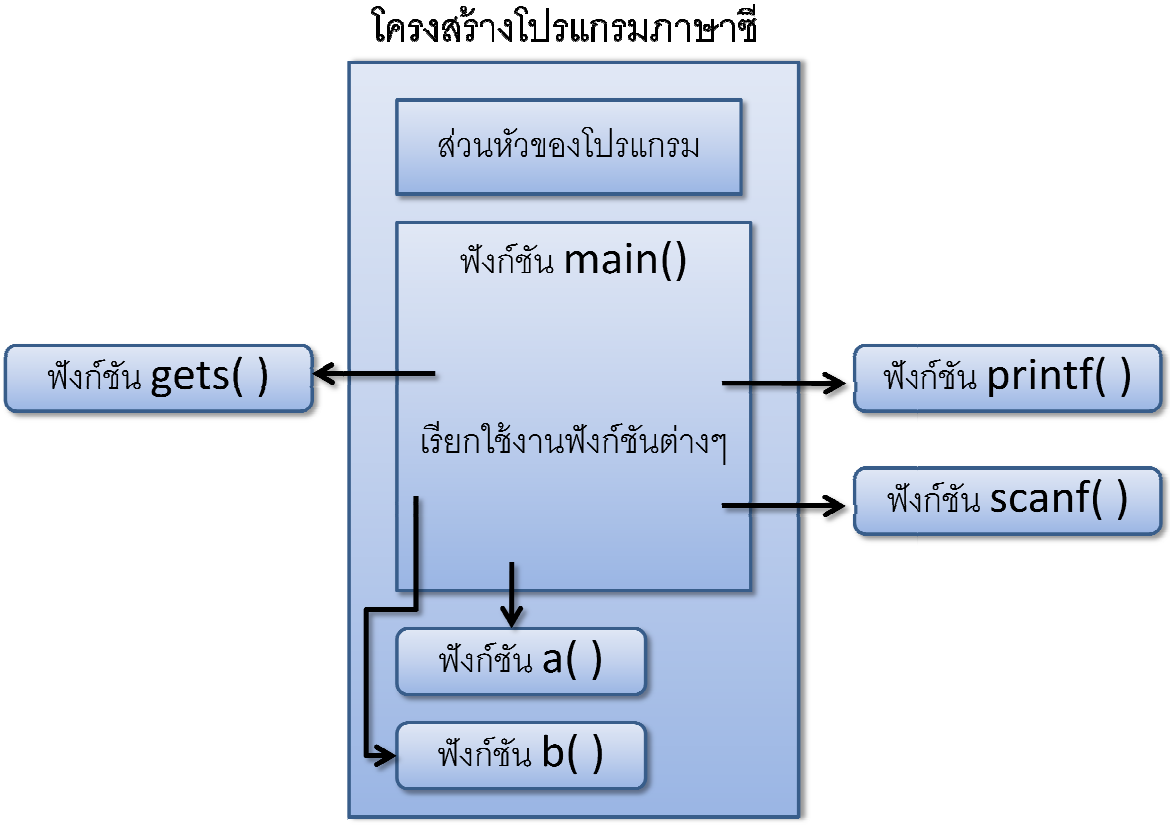
ถ้าเป็นตัวแปรชนิดอื่นๆในภาษาซี อย่างเช่น int, float หรือ char ตัวแปลภาษาซีจะจองพื้นที่ในหน่วยความจำให้เท่ากับขนาดของตัวแปรเหล่านั้นคือ 4, 4 และ 1 ไบต์ ตามลำดับ แต่ถ้าเป็นตัวแปรชนิดพอยน์เตอร์ซึ่งเก็บค่าตำแหน่งในหน่วยความจำ โดยตำแหน่งในหน่วยความจำของคอมพิวเตอร์จะใช้เป็นเลขในระบบฐาน 16 ขนาด 4 ไบต์ (สำหรับระบบ 32-bits) ดังนั้นไม่ว่าเราจะสร้างตัวแปรพอยน์เตอร์ชนิดใด(int, float หรือ char) ตัวแปรพอยน์เตอร์จะมีขนาดเท่ากันทั้งหมดคือ 4 ไบต์

บทที่ 11
ฟังก์ชันในภาษาซี

ในการทำงานบางอย่างจำเป็นต้องใช้คำสั่งมากกว่าหนึ่งคำสั่งเพื่อทำงานนั้นให้สำเร็จ ซึ่งคำสั่งที่เขียนรวมกันไว้เพื่อใช้งานจะเรียกว่า ฟังก์ชัน(function) จากบทที่ผ่านมาเราได้เรียกใช้งานฟังก์ชันในภาษาซีมาบ้างแล้ว เช่น printf( ), scanf( ) เป็นต้น ฟังก์ชันเหล่านี้เป็นฟังก์ชันมาตรฐานใน ภาษาซีหรือไลบรารีฟังก์ชัน

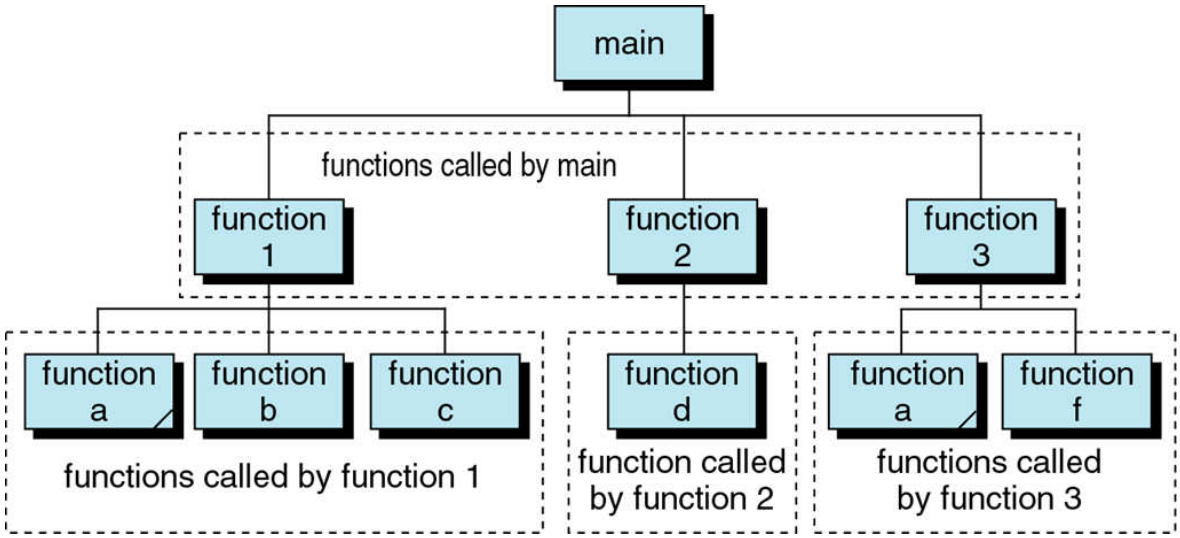
11.1 รูปแบบของฟังก์ชัน

โครงสร้างของโปรแกรมที่เขียนด้วยภาษาซี ภายในโปรแกรมจะประกอบด้วยฟังก์ชันต่างๆ โดยจะไม่มีคำสั่งที่เขียนเดี่ยวๆ อยู่นอกฟังก์ชัน อย่างน้อยที่สุดในโปรแกรมจะต้องมีหนึ่งฟังก์ชันเสมอนั่นคือ ฟังก์ชัน main( ) ซึ่งเป็นฟังก์ชันหลักที่โปรแกรมภาษาซีจะเริ่มต้นทำงานจากจุดนี้ จากนั้นภายในฟังก์ชัน main( ) อาจจะมีการเรียกใช้ฟังก์ชันอื่นๆ ต่อไปอีกดังรูปที่ 11.1

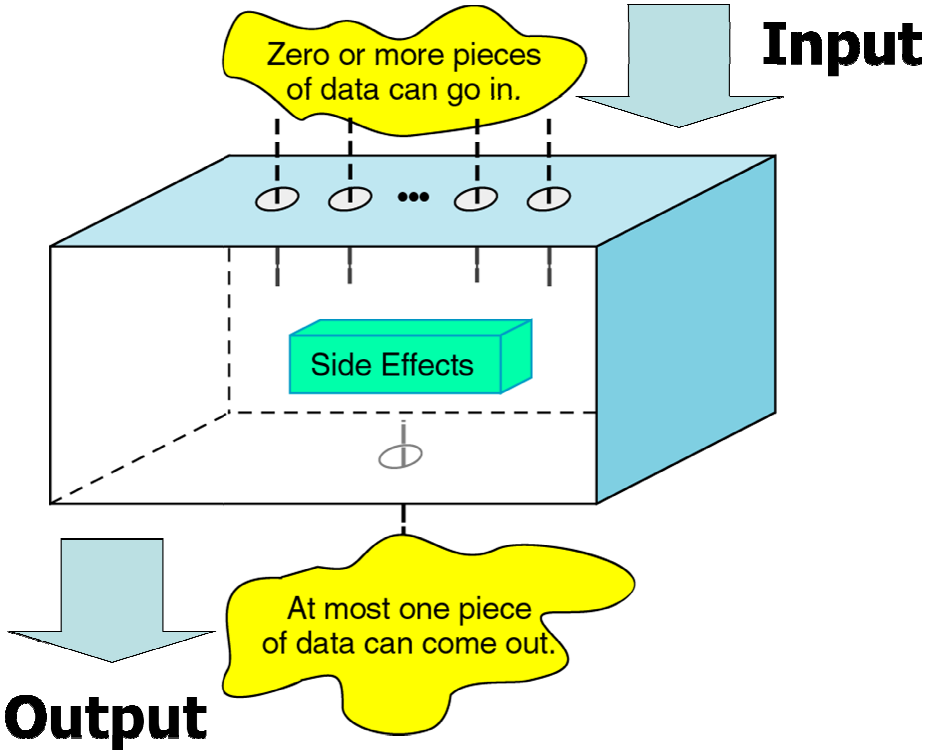


รูปที่ 11.1 โครงสร้างโปรแกรมภาษาซี

อีกทั้งในฟังก์ชันยังสามารถเรียกใช้ฟังก์ชันต่อเนื่องกันไปเรื่อยๆได้อีก โดยมีฟังก์ชัน main( ) เป็นจุดเริ่มต้นของการทำงาน ดังรูปที่ 11.2



รูปที่ 11.2 โครงสร้างโปรแกรมภาษาซีโดยมีฟังก์ชัน main( ) เป็นราก



รูปที่ 11.3 การทำงานของฟังก์ชัน

รูปที่ 11.3 แสดงรูปการทำงานของฟังก์ชันซึ่งฝั่งขาเข้า (input) อาจจะไม่มีข้อมูล หรือมีหลายข้อมูลก็ได้ เมื่อผ่านเข้าฟังก์ชันค่าที่ถูกป้อนเป็น input ต่างๆ จะถูกคำนวณตามคำสั่งที่ฟังก์ชันกำหนดไว้ บางครั้งอาจจะเกิดสิ่งที่เรียกว่า side effects อาจจะทำให้ค่า input เปลี่ยนค่าก็ได้ ผู้พัฒนาโปรแกรมควรระวังในส่วนนี้ด้วย ผลลัพธ์จากฟังก์ชันหรือที่เรียกว่า output จะมีแค่เพียงค่าเดียวเท่านั้น

ฟังก์ชันในภาษาซีสามารถแบ่งออกเป็น 2 ประเภท คือ ไลบรารีฟังก์ชัน (Library Function) หรือเรียกได้อีกอย่างหนึ่งว่า ฟังก์ชันมาตรฐาน และฟังก์ชันที่ผู้ใช้สร้างขึ้นเอง (User defined function)

11.1.1 ไลบรารีฟังก์ชันหรือฟังก์ชันมาตรฐาน

ฟังก์ชันมาตรฐานในภาษาซีเป็นฟังก์ชันที่มีมาให้พร้อมกับตัวแปลภาษาซี(C compiler) ตามข้อกำหนด ANSI C เพื่อใช้งานในด้านต่างๆ โดยเน้นที่งานพื้นฐาน อย่างเช่นฟังก์ชันคำนวณทางคณิตศาสตร์, ฟังก์ชันสำหรับจัดการกับข้อความ หรือฟังก์ชันจัดการเกี่ยวกับเวลา เป็นต้น เพื่อให้ผู้พัฒนาโปรแกรมภาษาซีทำงานได้สะดวกขึ้น การใช้งานฟังก์ชันมาตรฐานปกติแล้วจำเป็นต้องใช้คำสั่ง #include แฟ้มข้อมูล .h หรือที่เรียกว่า header file เข้าไปในโปรแกรมก่อนการเรียกใช้งาน เนื่องจาก header file จะเป็นตัวที่นิยามฟังก์ชัน และค่าต่างๆ ที่ฟังก์ชันจำเป็นต้องใช้ เช่นถ้าต้องการใช้งานฟังก์ชันทางคณิตศาสตร์ ในโปรแกรมจะเป็นต้อง #include <math.h> เข้าไปก่อน ดังแสดงในตัวอย่างโปรแกรมที่ 11.1

ตัวอย่างโปรแกรมที่ 11.1

```
1  #include <stdio.h>                    // #include ก่อนจะเรียกใช้ฟังก์ชันใน stdio.h
2  #include <math.h>                     // และ math.h
3  int main(int argc, char **argv) {
4      int   x, y;
5      printf("Enter number: ");         // ฟังก์ชัน printf( ) ที่อยู่ใน stdio.h
6      scanf("%d", &x);                  // ฟังก์ชัน scanf( ) ที่อยู่ใน stdio.h
7      y = sqrt(x);                      // ฟังก์ชัน sqrt( ) ที่อยู่ใน math.h
8      printf(" Square root of %d = %d\n", x, y);
9  }
```

ผลการรัน

```
Enter number : 4
Square root of 4 = 2
```

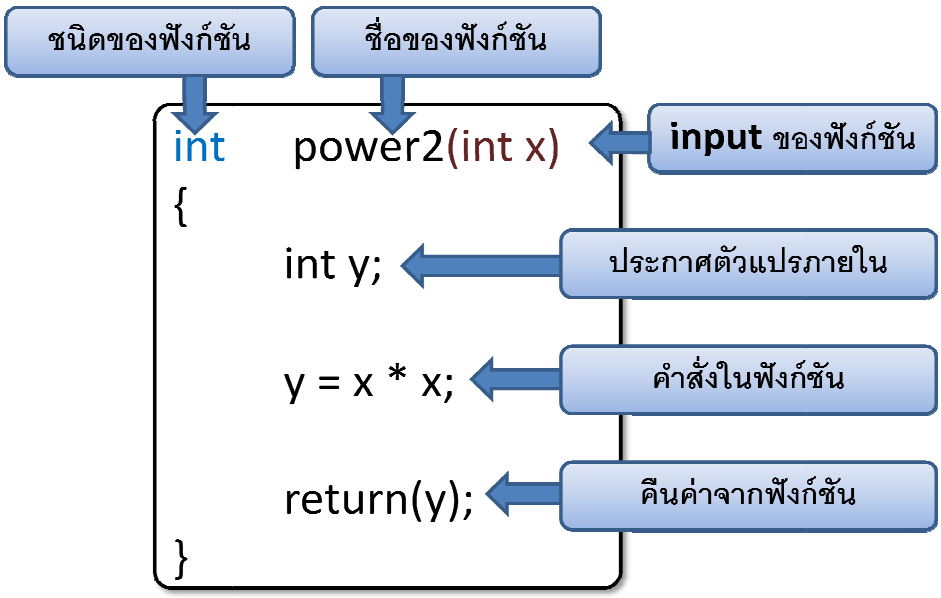
11.1.2 ฟังก์ชันที่ผู้ใช้สร้างขึ้นเอง

ไลบรารีฟังก์ชันในภาษาซี เป็นฟังก์ชันที่สร้างขึ้นมาเพื่อทำงานพื้นฐานทั่วไป ซึ่งในบางครั้งจะไม่มีไลบรารีที่ทำงานได้อย่างที่ต้องการ ดังนั้นผู้พัฒนาโปรแกรมจึงต้องสร้างฟังก์ชันขึ้นมาใช้งานเอง ข้อดีของการสร้างฟังก์ชันขึ้นมาใช้เอง คือกรณีที่มีกลุ่มของชุดคำสั่งที่ต้องใช้บ่อยครั้ง ทำให้เราสามารถเขียนฟังก์ชันขึ้นมาครั้งเดียว แล้วเรียกใช้ได้บ่อยครั้งตามที่ต้องการ โดยไม่ต้องเสียเวลาเขียนกลุ่มของชุดคำสั่งนี้ทุกครั้งที่ใช้

นอกจากนี้การเขียนและใช้งานฟังก์ชันจะทำให้โปรแกรมถูกแบ่งออกเป็นส่วนเล็กๆ ทำให้ง่ายต่อการตรวจสอบและแก้ไขโปรแกรม โดยเฉพาะโปรแกรมทีมีขนาดใหญ่ หรือมีผู้เขียนร่วมกันหลายคน การสร้างฟังก์ชันขึ้นมาใช้งานเอง จำเป็นต้องเขียนให้ตรงกับรูปแบบการประกาศฟังก์ชันที่ภาษาซีกำหนดไว้ ซึ่งรูปแบบการประกาศฟังก์ชันแสดงได้ดังนี้

```
type   function-name (type  arg-1,  type  arg-2, … ) {
          local variable declaration;
          statement-1;
          statement-2;
          ….
          statement-n;
          return(value);
}
```

- type : ชนิดของฟังก์ชัน โดยพิจารณาจากผลลัพธ์ที่ได้จากการทำงานของฟังก์ชัน ถ้าฟังก์ชันไม่ให้ผลลัพธ์การทำงานใดๆ ให้ระบุเป็น void
- function-name : ชื่อของฟังก์ชัน
- type arg-1, type arg-2 : เป็นการกำหนดชนิดของอาร์กิวเมนต์ที่จะส่งเข้ามาให้กับฟังก์ชัน
- local variable declaration : เป็นการประกาศตัวแปรที่ใช้ในฟังก์ชันเท่านั้น
- statement-1 ถึง statement-2 : เป็นคำสั่งการทำงานของฟังก์ชัน
- return(value) : คำสั่งสำหรับคืนค่าเมื่อฟังก์ชันทำงานสิ้นสุดลง ถ้าฟังก์ชันไม่คืนผลลัพธ์ก็สามารถละคำสั่งนี้ได้



รูปที่ 11.4 ตัวอย่างการประกาศฟังก์ชันที่ผู้ใช้สร้างขึ้นเอง

รูปที่ 11.4 เป็นการประกาศฟังก์ชันที่ผู้ใช้สร้างขึ้นเอง โดยฟังก์ชันชื่อ power2 มีการรับอาร์กิวเมนต์หนึ่งค่านั่นก็คือ x ที่เป็นประเภทข้อมูลแบบ int ซึ่งฟังก์ชันนี้จะมีการคืนค่าเป็นประเภทข้อมูลแบบ int ภายในฟังก์ชันมีการประกาศตัวแปรภายในหนึ่งตัวคือ y ที่เป็นข้อมูลประเภท int ชุดคำสั่งในฟังก์ชันนี้มีเพียงชุดคำสั่งเดียวก็คือ y = x * x จากนั้นฟังก์ชันนี้สิ้นสุดด้วยการคืนค่า y ให้กับผู้ที่เรียกฟังก์ชัน

11.2 ประเภทของฟังก์ชัน

ฟังก์ชันในภาษาซี ถ้าใช้การรับ/ส่งค่าของฟังก์ชันเป็นเกณฑ์ สามารถแบ่งออกได้เป็น 3 ประเภท

- ฟังก์ชันที่ไม่มีการรับ/ส่งค่า
- ฟังก์ชันที่มีการรับค่าเข้าไปในฟังก์ชัน
- ฟังก์ชันที่มีการส่งค่ากลับออกจากฟังก์ชัน

11.2.1 ฟังก์ชันที่ไม่มีการรับ/ส่งค่า

ฟังก์ชันประเภทแรกคือ ฟังก์ชันที่ไม่มีการรับ/ส่งค่าใดๆ ซึ่งหมายความว่า การเรียกใช้ฟังก์ชันประเภทนี้ไม่ต้องส่งอาร์กิวเมนต์ใดๆ เข้ามาให้กับฟังก์ชัน และฟังก์ชันจะทำงานโดยไม่มีการส่งผลลัพธ์ใดๆ กลับมาให้กับผู้เรียกใช้ ดังนั้นถือได้ว่าฟังก์ชันประเภทนี้เป็นประเภทที่เขียนง่ายที่สุด ส่วนใหญ่จะใช้ในการแสดงผลข้อความ ดังตัวอย่างโปรแกรมที่ 11.2

ตัวอย่างโปรแกรมที่ 11.2

```
1  #include <stdio.h>                          // #include ก่อนจะเรียกใช้ฟังก์ชันใน stdio.h
2
3  void showProgram( ) {
4        printf(“=== Today Program ===\n”);
5        printf(“1. Mummy\n”);
6        printf(“2. Star war\n”);
7        printf(“3. Spiderman\n\n”);
8  }
9
10 int main(int argc, char **argv) {
11     char ch;
12     printf("Do you want to check showing program (y/n) : ");
13     ch = getchar( );
14     if((ch == ‘Y’) || (ch == ‘y’)) {
15             showProgram( );
16     } else {
17             printf(“Thank you\n”);
18     }
19 }
```

ผลการรัน (ถ้าผู้ใช้ป้อน y หรือ Y)

Do you want to check showing program (y/n) : ***Y***

=== Today Program ===

1. Mummy
2. Star war
3. Spiderman

ผลการรัน (ถ้าผู้ใช้ป้อนตัวอักษรอื่นที่ไม่ใช่ y หรือ Y)

Do you want to check showing program (y/n) ***: N***

Thank you

ตัวอย่างโปรแกรมที่ 11.2 แสดงการสร้างฟังก์ชันที่ไม่มีการรับ/ส่งข้อมูลขึ้นมาเองโดยตัวฟังก์ชันได้ถูกเขียนไว้ในบรรทัดที่ 3 - 8 ซึ่งฟังก์ชันจะไม่มีการคืนค่าเนื่องจากประเภทข้อมูลของฟังก์ชันถูกประกาศเป็น void ฟังก์ชันที่ถูกสร้างขึ้นมานี้ชื่อ showProgram และไม่มีอาร์กิวเมนต์ซึ่งก็หมายความว่าฟังก์ชันจะไม่มีการรับข้อมูลใดๆ เข้าฟังก์ชัน บรรทัดที่ 4 - 7 เป็นคำสั่งภายในฟังก์ชันซึ่งจะเป็นได้ว่าทั้งหมดเป็นฟังก์ชันแสดงผล printf( ) ส่วนบรรทัดที่ 10 - 19 เป็นฟังก์ชัน main( ) จะเป็นจุดเริ่มต้นของโปรแกรม ในฟังก์ชัน main( ) จะเรียกฟังก์ชันที่สร้างขึ้นมาเองในบรรทัดที่ 15 ด้วยคำสั่ง showProgram( );

11.2.2 ฟังก์ชันที่มีการรับค่า

ฟังก์ชันประเภทที่มีการรับค่าหมายถึง ฟังก์ชันที่เวลาจะเรียกใช้งานต้องมีการส่งค่าอาร์กิวเมนต์ไปให้กับฟังก์ชันนั้นด้วย โดยอาร์กิวเมนต์ที่ส่งไปต้องตรงตามข้อกำหนดของฟังก์ชัน อย่างเช่น ฟังก์ชันกำหนดให้ส่งอาร์กิวเมนต์เข้าไป 3 ตัวคือ int, float, long เวลาเรียกใช้งานฟังก์ชันก็ต้องส่งข้อมูลซึ่งอาจจะอยู่ในรูปของนิพจน์ ตัวแปร หรือ ค่าคงที่ก็ได้ แต่ต้องเป็นข้อมูลชนิด int, float, และ long ตามลำดับเท่านั้น

ตัวอย่างโปรแกรมที่ 11.3

```
1  #include <stdio.h>
2  void average(int x, int  y) {
3      printf("Average of %d and %d = %f\n", x, y, (x + y)/2.0);
4  }
5
6  int main(int argc, char **argv) {
7      int a, b;
8      printf("Enter number 1 :  ");  scanf("%d", &a);
9      printf("Enter number 2 :  ");  scanf("%d", &b);
10     average(a, b);
11 }
```

ผลการรัน

```
Enter number 1 : 5
Enter number 2 : 10
Average of 5 and 10 = 7.500000
```

ตัวอย่างโปรแกรมที่ 11.3 แสดงการสร้างฟังก์ชันที่มีการรับข้อมูลขึ้นมาเอง ตัวฟังก์ชันได้ถูกเขียนไว้ในบรรทัดที่ 2 - 4 ซึ่งฟังก์ชันจะไม่มีการคืนค่าเนื่องจากประเภทข้อมูลของฟังก์ชันถูกประกาศเป็น void ฟังก์ชันที่ถูกสร้างขึ้นมานี้ชื่อ average จะรับข้อมูลเข้ามา 2 ค่า คือ x, y ซึ่งมีประเภทข้อมูลแบบจำนวนเต็ม (int) การทำงานของฟังก์ชันจะนำค่า x และ y มาบวกกันจากนั้นหารด้วย 2.0 เพื่อเป็นการหาค่าเฉลี่ยของ x และ y แล้วนำผลลัพธ์ที่ได้มาแสดงออกทางหน้าจอ ฟังก์ชัน main( ) เป็นจุดเริ่มต้นในการทำงานของโปรแกรมอยู่ในบรรทัดที่ 6 - 11 โดยฟังก์ชัน main( ) จะทำการรับค่า 2 ค่าจากทางแป้นพิมพ์ด้วยฟังก์ชัน scanf( ) มาเก็บในตัวแปร a และ b ในบรรทัดที่ 8 และ 9 ตามลำดับ จากนั้นฟังก์ชัน main( ) เรียกใช้งานฟังก์ชัน average( ) ที่บรรทัดที่ 10 เพื่อให้แสดงค่าเฉลี่ย 2 ค่าที่รับมาจากแป้นพิมพ์

ฟังก์ชันที่มีการรับค่าสามารถแบ่งประเภทย่อยลงไปอีกได้ 3 ประเภท ตามลักษณะของอาร์กิวเมนต์ที่ส่งให้กับฟังก์ชัน ได้แก่ ฟังก์ชันที่รับค่าโดยตรง, ฟังก์ชันที่รับค่าเป็นพอยน์เตอร์

11.2.2.1 ฟังก์ชันที่รับค่าโดยตรง

ฟังก์ชันที่รับค่าโดยตรงหมายถึง ฟังก์ชันที่มีการรับค่าตัวแปร ค่าคงที่ หรือนิพจน์ที่ให้ผลออกมาเป็นค่าคงที่ การเรียกใช้งานฟังก์ชันโดยส่งค่าให้โดยตรงในลักษณะนี้จะเรียกว่า call by value ถ้ามีการตั้งชื่อตัวแปรซ้ำกันในฟังก์ชันที่เป็นฝ่ายเรียกและฝ่ายถูกเรียกใช้งาน ตัวแปรเหล่านั้นจะถือว่าเป็นคนละตัวแปรกัน การเปลี่ยนแปลงค่าภายในฟังก์ชันจะไม่มีผลกับตัวแปรที่ใช้ชื่อเดียวกันในอีกฟังก์ชันหนึ่ง

ตัวอย่างโปรแกรมที่ 11.4

```
1  #include <stdio.h>
2  void average(int x, int  y) {
3      printf("Average of %d and %d = %f\n", x, y, (x + y)/2.0);
4  }
5  int main(int argc, char **argv) {
6     average(10, 5);
7  }
```

ผลการรัน

```
Average of 5 and 10 = 7.500000
```

ตัวอย่างโปรแกรมที่ 11.4 จะคล้ายกับตัวอย่างโปรแกรมที่ 11.3 จะเห็นได้ว่าเราสามารถจะส่งผ่านค่าคงที่เข้าฟังก์ชัน average( ) โดยตรงได้

ตัวอย่างโปรแกรมที่ 11.5

```
1  #include <stdio.h>
2  void plus5(int x) {
3      x = x + 5;
4      printf("X in plus5 = %d\n", x);
5  }
6
7  int main(int argc, char **argv) {
8      int   x;
9      printf("Enter number : " );
10     scanf("%d", &x);
11     printf("X in main before calling function = %d\n",  x);
12     plus5( x );
13     printf("X in main after calling function = %d\n",  x);
14 }
```

ผลการรัน

```
Enter number : 10
X in main before calling function = 10
X in plus5 = 15
X in main after calling function = 10
```

ตัวอย่างโปรแกรมที่ 11.5 ได้มีการสร้างฟังก์ชัน plus5( ) โดยจะรับอาร์กิวเมนต์ 1 ตัวคือ x ซึ่งจะเห็นว่าตัวแปร x มีการประกาศทั้งในฟังก์ชัน main( ) และฟังก์ชัน plus5( ) ในฟังก์ชัน plus5( ) จะมีการเอาค่าในตัวแปร x ไปบวก 5 แล้วแสดงผลลัพธ์ออกทางหน้าจอ จากตัวอย่างจะเห็นได้ว่าค่า x หลังจากบวก 5 จะได้ 15 ซึ่งมีการเปลี่ยนแปลงค่าของ x แต่อย่างไรก็ตาม call by value จะไม่ยุ่งเกี่ยวกับค่าในตัวแปรของผู้เรียกดังนั้นเมื่อสิ้นสุดฟังก์ชัน plus5( ) ค่า x ในฟังก์ชัน main( ) ก็ยังมีค่าเดิมอยู่ จะเห็นได้จากผลการรัน “X in main after calling function = 10”

ตัวอย่างโปรแกรมที่ 11.6

```
1  #include <stdio.h>
2
3  void swap(int  a, int  b) {
4      int  tmp;
5     tmp = a;
6     a  =  b;
7     b  =  tmp;
8     printf("In swap : A = %d, B = %d\n", a, b);
9  }
10
11 int main(int argc, char **argv) {
12    int  a = 5;
13    int  b = 10;
14    printf("Before swap A = %d, B = %d\n", a, b);
15    swap(a, b);
16    printf("After swap A = %d, B = %d\n", a, b);
17 }
```

ผลการรัน

```
Before swap A = 5, B = 10
In swap : A = 10, B = 5
After swap A = 5, B = 10
```

ตัวอย่างโปรแกรมที่ 11.6 แสดงให้เห็นการใช้งานของฟังก์ชันที่รับค่าโดยตรง บรรทัดที่ 3 – 9 เป็นฟังก์ชัน swap( ) ที่จะทำการสับเปลี่ยนค่าภายในของตัวแปร 2 ตัวแปร โดยจะทำการรับค่าให้กับ a และ b แล้วทำการสลับค่าที่อยู่ในตัวแปร a และ b จากผลการรันจะเห็นว่าการเรียกใช้ฟังก์ชันแบบนี้จะมีปัญหาเนื่องจากค่านั้นได้ถูกสับเปลี่ยนก็จริงแต่เป็นการเปลี่ยนกันระหว่างตัวแปรในฟังก์ชันซึ่งไม่มีการเกี่ยวพันใดๆ ทั้งสิ้นกับตัวแปรในฟังก์ชัน main( ) ทำให้ค่า a และ b ที่อยู่ในฟังก์ชัน main() ยังคงค่าเดิมอยู่หลังจากเรียกฟังก์ชัน swap( )

11.2.2.2 ฟังก์ชันที่รับค่าเป็นพอยน์เตอร์

นอกจากการส่งค่าโดยตรงหรือที่เรียกว่า call by value แล้ว ยังสามารถที่จะส่งพอยน์เตอร์ หรือแอสเดรดของตัวแปรเป็นอาร์กิวเมนต์ในการเรียกฟังก์ชันได้ โดยใช้เครื่องหมายของตัวแปรพอยน์เตอร์ (&, *) เข้ามาช่วย การส่งตัวแปรพอยน์เตอร์ในการเรียกฟังก์ชันจะเรียกว่า call by reference ถ้ามีการเปลี่ยนแปลงค่าของตัวแปรในฟังก์ชันใดฟังก์ชันหนึ่งก็จะมีผลต่อค่าของตัวแปรที่อยู่ในอีกฟังก์ชัน เนื่องจากตัวแปรทั้ง 2 ฟังก์ชันเป็นตัวแปรเดียวกัน และเก็บข้อมูลอยู่ในหน่วยความจำตำแหน่งเดียวกัน

ตัวอย่างโปรแกรมที่ 11.7

```
1  #include <stdio.h>
2
3  void average(int  *a, int  *b) {
4        printf("Average of %d and %d = %f\n",  *a,  *b,  (*a + *b)/2.0);
5  }
6
7  int main(int argc, char **argv) {
8        int  a = 10;
9        int  b = 5;
10       average(&a,  &b);
11 }
```

ผลการรัน

```
Average of 10 and 5 = 7.500000
```

ตัวอย่างโปรแกรมที่ 11.7 แสดงการสร้างและเรียกใช้งานฟังก์ชันที่มีการรับค่าแบบพอยน์เตอร์ จากบรรทัดที่ 3 จะเห็นว่าอาร์กิวเมนต์ของฟังก์ชันจะเป็นแบบพอยน์เตอร์คือ int *a และ int *b การใช้งานของค่าที่อยู่ในตัวแปรพอยน์เตอร์จะต้องอ้างแบบพอยน์เตอร์คือใช้งาน *a และ *b ในบรรทัดที่ 4 และการคำนวณในฟังก์ชันเพื่อหาค่าเฉลี่ยก็จะนำค่าของ *a มาบวกกับ *b แล้วหารด้วย 2.0 แล้วแสดงผลออกทางหน้าจอ ส่วนบรรทัดที่ 7 – 11 เป็นฟังก์ชัน main( ) ซึ่งจะเห็นในบรรทัดที่ 10 จะเป็นการเรียกฟังก์ชัน average( ) จะสังเกตเห็นได้ว่าจะต้องส่งค่าแอสเดรดของตัวแปรคือ &a และ &b ให้กับฟังก์ชัน

ตัวอย่างโปรแกรมที่ 11.8

```
1  #include <stdio.h>
2
3  void swap(int  *a, int  *b) {
4      int  tmp;
5     tmp = *a;
6     *a  =  *b;
7     *b  =  tmp;
8     printf("In swap : A = %d, B = %d\n", *a, *b);
9  }
10
11 int main(int argc, char **argv) {
12    int  a = 5;
13    int  b = 10;
14    printf("Before swap A = %d, B = %d\n", a, b);
15    swap(&a, &b);
16    printf("After swap A = %d, B = %d\n", a, b);
17 }
```

ผลการรัน

```
Before swap A = 5, B = 10
In swap : A = 10, B = 5
After swap A = 10, B = 5
```

ตัวอย่างโปรแกรมที่ 11.8 แสดงให้เห็นการใช้งานของฟังก์ชันที่รับค่าแบบพอนย์เตอร์ บรรทัดที่ 3 – 9 เป็นฟังก์ชัน swap( ) ที่จะทำการสับเปลี่ยนค่าภายในของตัวแปร 2 ตัวแปร ซึ่งตัวแปร *a และ *b จะชี้ไปยังตัวแปร a และ b ในฟังก์ชัน main( ) ตามลำดับ แล้วทำการสลับค่าที่อยู่ในตัวแปร a และ b จากผลการรันจะเห็นว่าการเรียกใช้ฟังก์ชันแบบนี้สามารถแก้ไขปัญหาที่เกิดขึ้นในตัวอย่างโปรแกรมที่ 11.6 ได้ จะเห็นได้ว่าหลังจากเรียกฟังก์ชัน swap( ) แล้วนั้น ค่าตัวแปร a และ b ในฟังก์ชัน main( ) จะเปลี่ยนไปด้วยจริงๆ

11.2.3 ฟังก์ชันที่มีการส่งค่ากลับออกจากฟังก์ชัน

ฟังก์ชันประเภทสุดท้ายคือ ฟังก์ชันที่มีการส่งค่าผลลัพธ์จากการทำงานกลับออกไปจากฟังก์ชันด้วย โดยส่วนใหญ่แล้วฟังก์ชันประเภทนี้มักจะใช้กับงานประเภทที่ต้องการคำนวณค่า ในการเรียกใช้งานฟังก์ชันประเภทที่มีการส่งค่ากลับ เราจำเป็นจะต้องสร้างตัวแปรที่มีประเภทข้อมูลเดียวกันกับประเภทที่ฟังก์ชันคืนกลับมาเพื่อรับค่าที่คืนกลับมาจากฟังก์ชัน ซึ่งมีการลักษณะการใช้งานดังนี้

variable = function-name(arg-1, arg-2, …)
• variable : ตัวแปรที่สร้างขึ้นเพื่อใช้รับค่าที่คืนกลับมาจากฟังก์ชัน • function-name : ชื่อของฟังก์ชัน • arg-1, arg-2 : ค่าอาร์กิวเมนต์ที่จะส่งเข้ามาไปในฟังก์ชัน

ส่วนค่าที่คืนกลับมาจากฟังก์ชันนั่นจะอยู่ในส่วนของคำสั่ง return ภายในฟังก์ชัน โดยส่วนใหญ่จะเขียนไว้ที่บรรทัดสุดท้ายของฟังก์ชัน ซึ่งจะมีลักษณะการใช้งานดังนี้

return(value)
• return : คำสั่งที่ใช้ส่งค่ากลับออกจากฟังก์ชัน • value : ตัวแปรหรือค่าที่จะส่งกลับออกจากฟังก์ชัน ซึ่งจะต้องเป็นประเภทเดียวกับประเภทข้อมูลของฟังก์ชัน

ตัวอย่างโปรแกรมที่ 11.9

```
1  #include <stdio.h>
2  int  power3(int  x) {
3        return( x * x * x);
4  }
5
6  int  main(int argc, char **argv) {
7       printf("2^3 = %d\n", power3(2));
8  }
```

ผลการรัน

```
2^3 = 8
```

ตัวอย่างโปรแกรมที่ 11.9 แสดงการสร้างและใช้งานฟังก์ชันที่มีการคืนค่าโดยบรรทัดที่ 2 -4 จะเป็นฟังก์ชันชื่อ power3( ) ที่มีการรับค่าอาร์กิวเมนต์ 1 ค่าคือ x และมีการคืนค่าเป็นประเภทข้อมูลแบบ int ซึ่งจะเห็นได้ว่าฟังก์ชันจะทำการคำนวณค่า x คูณกับ 3 ครั้ง (x * x * x) แล้วคืนค่าที่คำนวณได้ออกจากฟังก์ชันด้วยคำสั่ง return ส่วนฟังก์ชัน main( ) ที่เขียนในบรรทัดที่ 6 - 8 จะเป็นการส่งค่า 2 ให้กับฟังก์ชัน แล้วแสดงค่าที่คืนจากฟังก์ชันออกทางหน้าจอ

ตัวอย่างโปรแกรมที่ 11.10

```
1  #include <stdio.h>
2  float  average(int  x, int  y) {
3        return( (x + y)/2.0 );
4  }
5
6  int main(int argc, char **argv) {
7        float  f;
8        printf("Enter number 1: ");  scanf("%d", &a);
9        printf("Enter number 2: ");  scanf("%d", &b);
10       f = average(a, b);
11      printf("Average of %d and %d = %f\n", a, b, f);
12 }
```

ผลการรัน

Enter number 1: ***10***

Enter number 2: ***5***

Average of 10 and 5 = 7.500000

ตัวอย่างโปรแกรมที่ 11.10 แสดงการสร้างและใช้งานฟังก์ชันที่มีการคืนค่าโดยบรรทัดที่ 2 -4 จะเป็นฟังก์ชันชื่อ average( ) ที่มีการรับค่าอาร์กิวเมนต์ 2 ค่าคือ x และ y และมีการคืนค่าเป็นประเภทข้อมูลแบบ float ซึ่งจะเห็นได้ว่าฟังก์ชันจะทำการคำนวณค่า x บวกกับ y และหาร 2 ((x + y)/2.0) แล้วคืนค่าที่คำนวณได้ออกจากฟังก์ชันด้วยคำสั่ง return ส่วนฟังก์ชัน main( ) ที่เขียนในบรรทัดที่ 6 - 12 จะเป็นการรับค่า 2 ค่าคือ a และ b จากแป้นพิมพ์แล้วส่งค่า ให้กับฟังก์ชัน แล้วนำค่าที่คืนกลับมาจากฟังก์ชันเก็บไว้ในตัวแปร f จากนั้นจึงแสดงค่าออกทางหน้าจอ

11.3 การเขียนโปรโตไทป์สำหรับฟังก์ชัน

โปรโตไทป์ (Prototype) คือ คำสั่งอธิบายรายละเอียดของฟังก์ชัน ซึ่งเป็นคำสั่งที่บอกให้ตัวแปรภาษาซีรู้จักกับฟังก์ชันก่อนที่ฟังก์ชันจะถูกเรียกใช้งาน ตามปกติถ้าต้องการให้ตัวแปรภาษาซีแปลความหมายของโปรแกรมให้ถูกต้อง เราจะต้องเขียนฟังก์ชันที่สร้างขึ้นเอง ไว้ก่อนฟังก์ชัน main( ) เนื่องจากถ้ามีการเรียกใช้ฟังก์ชันก่อน ที่จะเจอตัวฟังก์ชัน ตัวแปรภาษาซีจะเตือนว่าไม่รู้จักฟังก์ชัน แต่การเขียนโปรโตไทป์จะทำให้เราสามารถย้ายตำแหน่งของฟังก์ชันไปไว้ส่วนใดของโปรแกรมก็ได้ หรือไว้ในอีกโปรแกรมต้นฉบับก็ได้ โดยไม่จำเป็นต้องเขียนไว้ก่อนฟังก์ชัน main( ) ซึ่งรูปแบบการเขียนโปรโตไทป์จะเหมือนกับการเขียนบรรทัดแรกสุดของฟังก์ชัน คือ

```
type   function-name (type  arg-1,  type  arg-2, … );
```

- type : ชนิดของฟังก์ชัน โดยพิจารณาจากผลลัพธ์ที่ได้จากการทำงานของฟังก์ชัน ถ้าฟังก์ชันไม่ให้ผลลัพธ์การทำงานใดๆ ให้ระบุเป็น void
- function-name : ชื่อของฟังก์ชัน
- type arg-1, type arg-2 : เป็นการกำหนดชนิดของอาร์กิวเมนต์ที่จะส่งเข้ามาให้กับฟังก์ชัน

ตัวอย่างโปรแกรมที่ 11.11

```
1  #include <stdio.h>
2  int  power3(int  x) ;
3
4  int  main(int argc, char **argv) {
5       printf("2^3 = %d\n", power3(2));
6  }
7
8  int  power3(int  x) {
9        return( x * x * x);
10 }
```

ผลการรัน

```
2^3 = 8
```

ตัวอย่างโปรแกรมที่ 11.11 จะให้ผลลัพธ์เช่นเดียวกับตัวอย่างโปรแกรมที่ 11.9 แต่ตัวฟังก์ชันที่สร้างขึ้นนั้น (power3) จะถูกประกาศในบรรทัดที่ 8 – 10 ซึ่งจะเห็นว่าฟังก์ชัน main( ) นั้นถูกประกาศไว้ในบรรทัดที่ 4 – 6 โดยถูกประกาศก่อนฟังก์ชัน power3 ( ) แต่จะเห็นได้ว่ามีการประกาศโปรโตไทป์ของฟังก์ชัน power3( ) เอาไว้แล้วในบรรทัดที่ 2 ทำให้โปรแกรมนี้สามารถทำงานได้โดยไม่มีปัญหา แต่ถ้าโปรโตไทป์ในบรรทัดที่ 2 ไม่มีประกาศ ตัวแปรภาษาซีจะเตือนข้อผิดพลาดในโปรแกรมขึ้น

บทที่ 12
ตัวแปรและขอบเขตการใช้งานสำหรับฟังก์ชัน

เมื่อมีการสร้างฟังก์ชันขึ้นมาใช้งานเพิ่มในโปรแกรม จากเดิมที่เคยมีเพียงฟังก์ชัน main( ) การประกาศตัวแปรในแต่ละฟังก์ชันนั้นจึงต้องมีข้อกำหนดและขอบเขตของตัวแปรเพื่อป้องกันความสับสนในการใช้งาน โดยประเภทของตัวแปรในภาษาซีจะแบ่งตามขอบเขตของการสร้างและการใช้งาน ตัวแปรของเป็น 4 ประเภทใหญ่ๆคือ local, global, extern และ static

12.1 ตัวแปร local

ตัวแปร local จะเรียกอีกอย่างหนึ่งว่า ตัวแปร automatic เป็นตัวแปรประเภทที่สร้างขึ้นมาภายในฟังก์ชัน ขอบเขตการใช้งานตัวแปร local ก็จะอยู่ภายในฟังก์ชันที่สร้างตัวแปรขึ้นมาเท่านั้น จะไม่สามารถถูกเรียกใช้งานจากฟังก์ชันอื่นได้ และถ้ามีการสร้างตัวแปร local แล้วมีการตั้งชื่อซ้ำกันระหว่างฟังก์ชัน ก็ถือว่าเป็นคนละตัวแปรกัน การสั่งให้เปลี่ยนค่าที่ฟังก์ชันใดฟังก์ชันหนึ่งจะไม่ส่งผลกระทบต่ออีกฟังก์ชันหนึ่งที่ใช้ชื่อตัวแปร local เดียวกัน

ตัวอย่างโปรแกรมที่ 12.1

```
1  #include <stdio.h>
2  void  testLocal( ) {
3        int  i;
4        i += 5;
5  }
6  int  main(int argc, char **argv) {
7        int  i = 10;
8        printf("i  = %d\n", i);
9        testLocal( );
10      printf("i  = %d\n", i);
11 }
```

ผลการรัน

```
i = 10
i = 10
```

ตัวอย่างโปรแกรมที่ 12.1 มีการสร้างและใช้งานฟังก์ชันที่ชื่อ testLocal( ) ที่ไม่มีการรับ/ส่งข้อมูล ในตัวของฟังก์ชัน testLocal( ) จะมีการประกาศตัวแปรประเภท local ที่ชื่อ i และภายในฟังก์ชันมีคำสั่ง i += 5 ซึ่งเป็นการเพิ่มค่า i อีก 5 ในขณะที่ฟังก์ชัน main( ) มีการประกาศตัวแปรชื่อ i เหมือนกับในฟังก์ชัน testLocal( ) โดยกำหนดให้ตัวแปร i ในฟังก์ชัน main( ) มีค่าเริ่มต้นเท่ากับ 10 แล้วแสดงค่า i ออกทางหน้าจอ จากนั้นเรียกฟังก์ชัน testLocal( ) และแสดงค่า i ออกทางหน้าจออีกครั้ง จะเห็นจากผลการรันว่าค่า i ภายในฟังก์ชัน main( ) ก่อนและหลังเรียกฟังก์ชัน testLocal( ) ให้ค่าเท่ากัน ดังนั้นค่าของ i ที่ถูกเพิ่มในฟังก์ชัน testLocal( ) จะไม่เกี่ยวข้องกับค่า i ในฟังก์ชัน main( ) เลย

12.2 ตัวแปร global

ตัวแปร global หรือเรียกอีกอย่างหนึ่งว่า ตัวแปร external เป็นตัวแปรประเภทที่สามารถเรียกใช้งานจากฟังก์ชันใดก็ได้ในโปรแกรม โดยการสร้างตัวแปร global จะต้องเขียนคำสั่งประกาศสร้างไว้ ที่ส่วนกำหนดค่าเริ่มต้น ของโปรแกรมต่อจากส่วนของ Preprocessor โดยไม่ได้สร้างไว้ภายในฟังก์ชันใดฟังก์ชันหนึ่งโดยเฉพาะ ขอบเขตการใช้งานตัวแปร global ก็คือตลอดทั้งโปรแกรมสามารถเรียกใช้หรือเขียนคำสั่งเพื่อเปลี่ยนแปลงค่าของตัวแปรประเภท global จากฟังก์ชันใดก็ได้ในโปรแกรม

ตัวอย่างโปรแกรมที่ 12.2

```
1  #include <stdio.h>
2  int   i;
3  void  testGlobal( ) {
4        i += 5;
5  }
6  int  main(int argc, char **argv) {
7        i = 10;
8        printf("i  = %d\n", i);
9        testGlobal( );
10       printf("i  = %d\n", i);
11 }
```

ผลการรัน

```
i = 10
i = 15
```

ตัวอย่างโปรแกรมที่ 12.2 มีการประกาศตัวแปร global ในบรรทัดที่ 2 และมีการสร้างและใช้งานฟังก์ชันที่ชื่อ testGlobal( ) ที่ไม่มีการรับ/ส่งข้อมูล ภายในฟังก์ชันมีคำสั่ง i += 5 ซึ่งเป็นการเพิ่มค่า i อีก 5 ในขณะที่ฟังก์ชัน main( ) จะมีการกำหนดค่าเริ่มต้นเท่ากับ 10 ให้กับตัวแปร i แล้วแสดงค่า i ออกทางหน้าจอ จากนั้นเรียกฟังก์ชัน testGlobal( ) และแสดงค่า i ออกทางหน้าจออีกครั้ง จะเห็นจากผลการรันว่าค่า i ก่อนและหลังเรียกฟังก์ชัน testGlobal( ) ให้ค่าไม่เท่ากันเนื่องจากค่าของตัวแปร i ซึ่งเป็นตัวแปร global ถูกเปลี่ยนค่าระหว่างเรียกใช้ฟังก์ชัน testGlobal( )

12.3 ตัวแปร extern

ตัวแปรประเภท global นั้นจะสามารถใช้ได้จากทุกส่วนของโปรแกรม แต่ปัญหาจะเกิดขึ้นเมื่อมีการเขียนแฟ้มข้อมูลต้นฉบับหลายแฟ้มข้อมูล ยกตัวอย่างเช่น ถ้าในโปรแกรมมีแฟ้มข้อมูลต้นฉบับอยู่ 2 แฟ้มข้อมูล main.c และ function.c ถ้าต้องการประกาศและใช้งานตัวแปรชื่อ x เป็นตัวแปรประเภท global โดยให้ทั้งสองแฟ้มข้อมูลต้นฉบับเห็นเหมือนกัน ถ้าเราประกาศตัวแปรไว้ใน main.c เมื่อคอมไพล์แฟ้มข้อมูลต้นฉบับ function.c จะมีปัญหาเนื่องจากโปรแกรมไม่รู้จักตัวแปร x แต่ถ้าเราประกาศตัวแปรไว้ในแฟ้มข้อมูลต้นฉบับ function.c เมื่อคอมไพล์ก็จะมีปัญหาที่แฟ้มข้อมูลต้นฉบับ main.c เนื่องจากโปรแกรมไม่รู้จักตัวแปร x แต่ถ้าประกาศตัวแปร x นี้ไว้ทั้งสองแฟ้มข้อมูลต้นฉบับก็จะเกิดปัญหาการประกาศตัวแปรซ้ำซ้อนอีก

ดังนั้นจึงจำเป็นต้องมีการใช้ตัวแปร extern เมื่อตัวแปรมีคำว่า extern อยู่ข้างหน้าจะหมายถึงตัวแปรนั้นจะไม่ได้จองหน่วยความจำและเป็นการบอกตัวแปลภาษาซีว่าจะต้องมีตัวแปรตัวนี้ประกาศอยู่ที่ไหนซักแห่งในแฟ้มข้อมูลต้นฉบับซักแฟ้มข้อมูลหนึ่ง

ตัวอย่างโปรแกรมที่ 12.3

```
1  //----- function.c ----
2  #include <stdio.h>
3  int   x = 0;
4
5  void  increaseX( ) {
6     x++;
7  }
8  void   decreaseX( ) {
9     x--;
10 }
```

ตัวอย่างโปรแกรมที่ 12.4

```
1  //----- main.c ----
2  #include <stdio.h>
3  extern int   x;
4
5  int   main(int  argc,  char  **argv) {
6     printf("X = %d\n", x);
7     increaseX( );
8     printf("X = %d\n", x);
9     decreaseX( );
10    printf("X = %d\n", x);
11 }
```

ผลการรัน

```
X  = 0
X  = 1
X  = 0
```

จากตัวอย่างโปรแกรมที่ 12.3 และ ตัวอย่างโปรแกรมที่ 12.4 เป็นแฟ้มข้อมูลต้นฉบับ function.c และ main.c ตามลำดับ

- function.c
  - บรรทัดที่ 3 มีการประกาศตัวแปร global ชื่อ x โดยมีการกำหนดค่าเริ่มต้นคือ 0
  - บรรทัดที่ 5 - 7 เป็นการประกาศฟังก์ชันชื่อ increaseX( ) โดยจะเพิ่มค่าตัวแปร x ไป 1 ค่า
  - บรรทัดที่ 8 - 10 เป็นการประกาศฟังก์ชันชื่อ decreaseX( ) โดยจะลดค่าตัวแปร x ไป 1 ค่า
- main.c
  - บรรทัดที่ 3 มีการประกาศตัวแปรประเภท extern ซึ่งจะไม่มีการจองหน่วยความจำจริงที่ตำแหน่งนี้ แต่เป็นการบอกตัวแปลภาษาซีว่า int x ได้ถูกประกาศไปแล้วในที่นี้คือ int x ได้ถูกประกาศไว้ในแฟ้มข้อมูล function.c
  - บรรทัดที่ 6 - 10 เป็นการแสดงค่าของตัวแปร x ก่อนและหลังการเขียนใช้งานฟังก์ชันต่างๆ

12.4 ตัวแปร static

ตัวแปร static สามารถแบ่งออกได้ 2 ประเภทคือ ตัวแปร static ที่เป็นแบบ local และ ตัวแปร static ที่เป็นประเภท globa

12.4.1 ตัวแปร static แบบ local

ตัวแปร static แบบ local จะเป็นการจองหน่วยความจำในฟังก์ชันแบบถาวร หมายถึงเมื่อสิ้นสุดการทำงานของฟังก์ชันแล้วข้อมูลในหน่วยความจำนั้นจะไม่ถูกคืนให้กับระบบ ทำให้สามารถอ้างอิงค่าที่อยู่ในหน่วยความจำตำแหน่งได้ทุกครั้งที่มีการเรียกใช้ฟังก์ชันอีกครั้ง

ตัวอย่างโปรแกรมที่ 12.5

```
1  #include <stdio.h>
2
3  void  showX( ) {
4      static  int  x  = 0;
5      printf("X = %d\n",  x);
6      x++;
7  }
8  int   main(int  argc,  char  **argv) {
9      showX( );
10     showX( );
11     showX( );
12 }
```

ผลการรัน

```
X  = 0
X  = 1
X  = 2
```

จากตัวอย่างโปรแกรมที่ 12.5 มีการประกาศฟังก์ชันชื่อ showX( ) ซึ่งมีการประกาศตัวแปรประเภท static (static int x) โดยกำหนดค่าเริ่มต้นเป็น 0 ในบรรทัด 4 ภายในฟังก์ชัน จากนั้นฟังก์ชันจะแสดงค่า x ออกทางหน้าจอ และเพิ่มค่า x ไป 1 ค่าก่อนจะออกจากฟังก์ชัน การกำหนดค่าเริ่มต้นในตัวแปรประเภท static จะถูกทำเพียงครั้งแรกที่มีการเรียกใช้งานฟังก์ชัน ถ้ามีการเรียกใช้ฟังก์ชันในครั้งถัดไป ตัวแปลภาษาซีจะไม่สนใจการกำหนดค่าเริ่มต้นอีกแล้ว

ในบรรทัดที่ 8 - 12 เป็นการประกาศฟังก์ชัน main( ) ซึ่งจะเห็นได้ว่ามีการเรียกใช้ฟังก์ชัน showX( ) ทั้งหมด 3 ครั้ง เมื่อดูจากผลการรันจะเห็นได้ว่าค่า x จะเพิ่มค่าขึ้นเรื่อยๆ โดยตัวแปร x สามารถจดจำค่าเก่าที่เก็บไว้ได้ แล้วเพิ่มค่าขึ้นทุกครั้งเมื่อมีการเรียกใช้ฟังก์ชัน showX( )

12.4.1 ตัวแปร static แบบ global

ในการเขียนโปรแกรมใหญ่ๆ บางครั้งจำเป็นต้องเขียนแฟ้มข้อมูลต้นฉบับมากกว่า 1 แฟ้มข้อมูล อีกทั้งยังอาจมีการใช้ตัวแปรประเภท global บ่อยครั้ง แต่เนื่องด้วยตัวแปร global จะถูกเห็นจากทุกแฟ้มข้อมูลต้นฉบับ ในบางครั้งเราอาจต้องการให้ตัวแปร global ถูกเห็นจำกัดอยู่เพียงแค่ในแฟ้มข้อมูลต้นฉบับเพียงแฟ้มข้อมูลเดียว เพื่อป้องกันการเขียนชื่อตัวแปร global ซ้ำกันในแฟ้มข้อมูลต้นฉบับอื่น เราจะใช้คำว่า static อยู่หน้าชื่อตัวแปร global เพื่อจำกัดขอบเขตของตัวแปรอยู่ที่แฟ้มข้อมูลต้นฉบับนั้นเพียงแฟ้มข้อมูลต้นฉบับเดียว

ตัวอย่างโปรแกรมที่ 12.6

```
1  //----- function.c ----
2  #include <stdio.h>
3  static  int   x = 0;
4
5  void  increaseX( ) {
6     x++;
7     printf("X in function.c = %d\n",  x);
8  }
9
10 void   decreaseX( ) {
11    x--;
12   printf("X in function.c = %d\n",  x);
13 }
```

ตัวอย่างโปรแกรมที่ 12.7

```
1  //----- main.c ----
2  #include <stdio.h>
3  int   x = 100;
4  int   main(int  argc,  char  **argv) {
5     printf("X in main.c = %d\n", x);
6     increaseX( );
7     decreaseX( );
8     printf("X in main.c = %d\n", x);
9  }
```

ผลการรัน

```
X in main.c = 100
X in function.c = 1
X in function.c = 0
X in main.c = 100
```

จากตัวอย่างโปรแกรมที่ 12.6 และ ตัวอย่างโปรแกรมที่ 12.7 เป็นแฟ้มข้อมูลต้นฉบับ function.c และ main.c ตามลำดับ

- function.c
  - บรรทัดที่ 3 มีการประกาศตัวแปร global ชื่อ x แบบ static โดยมีการกำหนดค่าเริ่มต้นคือ 0
  - บรรทัดที่ 5 - 8 เป็นการประกาศฟังก์ชันชื่อ increaseX( ) โดยจะเพิ่มค่าตัวแปร x ไป 1 ค่าแล้วแสดงค่า x ออกทางหน้าจอ
  - บรรทัดที่ 10 - 13 เป็นการประกาศฟังก์ชันชื่อ decreaseX( ) โดยจะลดค่าตัวแปร x ไป 1 ค่าแล้วแสดงค่า x ออกทางหน้าจอ
- main.c
  - บรรทัดที่ 3 มีการประกาศตัวแปรประเภท global โดยกำหนดค่าเริ่มต้น 100
  - บรรทัดที่ 5 แสดงค่า x ออกทางหน้าจอซึ่งจะเป็น x ในแฟ้มข้อมูลนี้คือ 100
  - บรรทัดที่ 6 เรียกใช้งานฟังก์ชัน increaseX( ) ซึ่งจะแสดงค่า x ของแฟ้มข้อมูล function.c ซึ่งจะได้ค่า 1

- บรรทัดที่ 7 เรียกใช้งานฟังก์ชัน decrease( ) ซึ่งจะแสดงค่า x ของแฟ้มข้อมูล function.c ซึ่งจะได้ค่า 0
- บรรทัดที่ 8 แสดงค่า x ออกทางหน้าจอซึ่งจะเป็นค่าของ x ที่เป็นตัวแปร global ของแฟ้มข้อมูลต้นฉบับ main.c ซึ่งจะเห็นได้ว่าจะได้ค่าเป็น 100

จากตัวอย่างโปรแกรมทั้งสองโปรแกรมจะเห็นได้ว่า ตัวแปร x ที่เป็นตัวแปรประเภท global ที่อยู่ใน function.c และ main.c จะเป็นตัวแปรคนละตัวกัน เนื่องจากมีการประกาศเป็นประเภท static เพื่อเป็นการจำกัดขอบเขตของตัวแปรประเภท global

12.5 ลำดับการใช้งานตัวแปรของฟังก์ชัน

ถ้าหากมีการใช้งานตัวแปร local และ global ร่วมกันก็สามารถทำได้ ภาษาซีให้ความสำคัญกับตัวแปรประเภท local ก่อนประเภท global ดังตัวอย่างโปรแกรมที่ 12.8

ตัวอย่างโปรแกรมที่ 12.8

```
1  #include <stdio.h>
2  int   x = 100;
3  void   Func1( ) {
4        int  x  = 5;
5        printf("X in Func1 = %d\n",  x);
6  }
7  void  Func2(int  x) {
8        printf("X in Func2 = %d\n", x);
9  }
10 int   main(int  argc,  char  **argv) {
11       Func1( );
12       Func2( 20 );
13       printf("X in main = %d\n",  x);
14 }
```

ผลการรัน

```
X in Func1 = 5
X in Func2 = 20
X in main = 100
```

บทที่ 13
ฟังก์ชันมาตรฐาน

ภาษาซีมีฟังก์ชันมาตรฐานอยู่เป็นจำนวนมาก ทำให้สะดวกต่อผู้พัฒนาโปรแกรม ฟังก์ชันมาตรฐานจะเก็บอยู่ส่วนที่เรียกกันว่า ไลบรารี(Library) ซึ่งก่อนที่จะสามารถเรียกใช้ฟังก์ชันที่อยู่ในไลบรารีได้นั้นจำเป็นจะต้องมีการอ้างถึงแฟ้มข้อมูล header (.h) ที่บรรจุข้อมูลและค่านิยามต่างๆ ของฟังก์ชันมาตรฐานก่อนในส่วนของ Preprocessor directive (#include) เพื่อที่โปรแกรมต้นฉบับจะได้รู้จักกับฟังก์ชันมาตรฐานที่จะถูกเรียกใช้ถัดไป

13.1 ฟังก์ชันมาตรฐานทางคณิตศาสตร์

เนื่องจากการคำนวณทางคณิตศาสตร์เป็นเรื่องพื้นฐาน และใช้งานกันบ่อยที่สุด จึงมีความจำเป็นต้องทราบถึงฟังก์ชันมาตรฐานที่สามารถช่วยในการคำนวณค่าทางคณิตศาสตร์ การใช้งานฟังก์ชันทางคณิตศาสตร์เราจะต้องทำการ include แฟ้มข้อมูลชื่อ math.h ในโปรแกรมก่อนเสมอ (#include <math.h>) โดยตัวแปรที่นำมาใช้กับฟังก์ชันทางคณิตศาสตร์จะต้องเป็นตัวแปรชนิดเลขจำนวนจริงละเอียด 2 เท่า (double) และผลลัพธ์ที่ได้จากฟังก์ชั่นก็จะเป็นชนิดเลขจำนวนจริงละเอียด 2 เท่าเช่นเดียวกัน ฟังก์ชั่นทางคณิตศาสตร์ที่ควรทราบมีดังตารางที่ 13.1 ถึง 13.5

ตารางที่ 13.1 ฟังก์ชันมาตรฐานทางคณิตศาสตร์(ตรีโกณมิติ)

ฟังก์ชันมาตราฐาน	คำอธิบาย
sin(x)	ฟังก์ชันสำหรับหาค่า sine ของมุม x ที่มีหน่วยเป็นเรเดียน
cos(x)	ฟังก์ชันสำหรับหาค่า cos ของมุม x ที่มีหน่วยเป็นเรเดียน
tan(x)	ฟังก์ชันสำหรับหาค่า tan ของมุม x ที่มีหน่วยเป็นเรเดียน
asin(x)	ฟังก์ชันสำหรับหาค่า arcsin ของค่า x จะคืนค่ามุมที่มีหน่วยเป็นเรเดียน
acos(x)	ฟังก์ชันสำหรับหาค่า arccos ของค่า x จะคืนค่ามุมที่มีหน่วยเป็นเรเดียน
atan(x)	ฟังก์ชันสำหรับหาค่า arctan ของค่า x จะคืนค่ามุมที่มีหน่วยเป็นเรเดียน

ค่ามุมปกติที่ใช้สอนกันในวิชาคณิตศาสตร์ตรีโกณมิติส่วนใหญ่จะอยู่ในรูปของ องศาดีกรี แต่ในภาษาซีจะใช้ค่ามุมที่เป็นเรเดียน ดังนั้นจึงควรเข้าใจการแปลงองศาดีกรีให้เป็นองศาเรเดียน และจากองศาเรเดียนให้เป็นองศาดีกรี ซึ่งปกติแล้ว

180 องศาดีกรี คือ π องศาเรเดียน (โดย π คือ 3.14159…)

ดังนั้นการแปลงมุมองศาดีกรีให้เป็นมุมองศาเรเดียน สามารถทำได้โดยคำนวณ

$$\text{องศาเรเดียน} = \frac{\text{องศาดีกรี} \times \pi}{180}$$

และการแปลงมุมองศาเรเดียนให้เป็นมุมองศาดีกรี สามารถทำได้โดยคำนวณ

$$\text{องศาดีกรี} = \frac{\text{องศาเรเดียน} \times 180}{\pi}$$

ตัวอย่างโปรแกรมที่ 13.1 แสดงการใช้งานฟังก์ชันมาตรฐาน sin( ), cos( ) และ tan( ) ในบรรทัดที่ 4 เป็นการคำนวณเพื่อแปลงองศาดีกรี ให้เป็นองศาเรเดียน จากนั้นเรียกใช้ฟังก์ชันมาตรฐานโดยป้อนค่ามุมดีกรี 30 องศาเข้าฟังก์ชันและแสดงผลลัพธ์ออกทางจอภาพ

ตัวอย่างโปรแกรมที่ 13.1

```
1  #include <stdio.h>
2  #include <math.h>
3  int main(int argc, char **argv) {
4      double  radian = 3.14159 / 180;
5      printf("Sin(30)  = %f\n",  sin(radian * 30));
6      printf("Cos(30) = %f\n",  cos(radian * 30));
7      printf("Tan(30)  = %f\n",  tan(radian * 30));
8  }
```

ผลการรัน

```
Sin(30)  = 0.500000
Cos(30) = 0.866026
Tan(30) = 0.577350
```

ตัวอย่างโปรแกรมที่ 13.2

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char **argv) {
5      double  degree = 180/3.14159;
6
7      printf("arc sin(0.5)  = %f\n",  degree * asin(0.5));
8      printf("arc cos(0.866026) = %f\n",  degree * acos(0.866026));
9      printf("arc tan(0.577350)  = %f\n",  degree * atan(0.577350));
10 }
```

ผลการรัน

```
 arc sin(0.5)  = 30.000025
arc cos(0.866026) = 29.999957
arc tan(0.577350) = 30.000014
```

จากตัวอย่างโปรแกรมที่ 13.2 แสดงตัวอย่างการทำงานของฟังก์ชันมาตรฐาน asin( ), acos( ) และ atan( ) ซึ่ง

- บรรทัดที่ 4 เป็นการคำนวณค่าตัวแปลงจากมุมเรเดียนให้เป็นมุมดีกรี
- บรรทัดที่ 5 - 7 เป็นการเรียกใช้ฟังก์ชันเพื่อหาค่ามุมเรเดียนและแปลงเป็นดีกรีก่อนแสดงผล

ผลลัพธ์ที่ควรจะได้ควรจะเป็น 30 องศาดีกรี แต่จะเห็นได้ว่าค่าที่ได้จริงไม่ใช่ 30 จริงๆ แต่ใกล้เคียงกับ 30 มาก ทั้งนี้เป็นเพราะค่า π ที่ใส่คือ 3.14159 ยังไม่ละเอียดพอ และค่าที่ป้อนเข้าฟังก์ชัน asin(), acos(), และ atan() ก็ยังไม่ละเอียดมากพอ ซึ่งผลที่ผิดพลาดมีค่าน้อยมากจนสามารถละทิ้งไม่เอามาคิดได้

ตารางที่ 13.2 ฟังก์ชันมาตรฐานทางคณิตศาสตร์(ลอการิธึม)

ฟังก์ชันมาตราฐาน	คำอธิบาย
exp(x)	ฟังก์ชันสำหรับหาค่า e^x
log(x)	ฟังก์ชันสำหรับหาค่า log ฐาน e ของ x
log10(x)	ฟังก์ชันสำหรับหาค่า log ฐาน 10 ของ x

ตารางที่ 13.2 รวมฟังก์ชันมาตรฐานในภาษาซีเพื่อคำนวณเกี่ยวกับลอการิธึม ซึ่งฟังก์ชันที่สำคัญจะมี 3 ฟังก์ชัน คือ exp( ), log( ) และ log10( ) วิธีใช้งานแสดงในตัวอย่างโปรแกรมที่ 13.3

ตัวอย่างโปรแกรมที่ 13.3

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char **argv) {
5      printf("exp(1)  = %f\n",  exp(1));
6      printf("log(exp(10)) = %f\n",  log(exp(10)));
7      printf("log10(100) = %f\n",  log10(100));
8  }
```

ผลการรัน

```
 exp(1)  = 2.718282
log(exp(10)) = 10.000000
log10(100) = 2.000000
```

ตัวอย่างโปรแกรมที่ 13.3 มีรายละเอียดการทำงานดังนี้

- บรรทัดที่ 5 เป็นการเรียกฟังก์ชัน exp( ) เพื่อคำนวณแสดงค่า e^1 ซึ่งจะได้ผลลัพธ์ 2.718282
- บรรทัดที่ 6 เป็นการเรียกฟังก์ชันซ้อนกัน โดยการคำนวณ exp(10) ก่อนแล้วนำผลลัพธ์ที่ได้ส่งไปให้ฟังก์ชันกับ log( ) ซึ่งตามสูตรคณิตศาสตร์ $\log_e(e^x) = x$ ซึ่งจะเห็นผลของการรันได้ 10.000000 ซึ่งตรงกับสูตรคณิตศาสตร์
- บรรทัดที่ 7 เป็นการเรียกฟังก์ชัน log10( ) โดยเป็นการหาค่า log ฐาน 10 ของค่าที่ป้อนผ่านฟังก์ชันซึ่งก็คือ 100 ทำให้ได้ผลลัพธ์คือ 2.000000 เนื่องจาก $10^2 = 100$

ตารางที่ 13.3 ฟังก์ชันมาตรฐานทางคณิตศาสตร์(ยกกำลังและค่าสัมบูรณ์)

ฟังก์ชันมาตราฐาน	คำอธิบาย
pow(x, y)	ฟังก์ชันสำหรับหาค่าเลขยกกำลัง x^y
sqrt (x)	ฟังก์ชันสำหรับหาค่ารากที่สองของ x
fabs(x)	ฟังก์ชันสำหรับหาค่าสัมบูรณ์ของ x

ตัวอย่างโปรแกรมที่ 13.4

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char **argv) {
5      printf("pow(5,3)  = %f\n",  pow(5, 3));
6      printf("sqrt(144) = %f\n",  sqrt(144));
7      printf("fabs(-10)  = %f\n",  fabs(-10));
8  }
```

ผลการรัน

```
pow(5,3)   = 125.000000
sqrt(144)  = 12.000000
fabs(-10)  = 10.000000
```

ตารางที่ 13.3 เป็นฟังก์ชันมาตรฐานที่ทำงานเกี่ยวกับการคำนวณเลขยกกำลังและหาค่าสัมบูรณ์ ตัวอย่างโปรแกรมที่ 13.4 แสดงการใช้งานฟังก์ชันมาตรฐานเหล่านี้โดยมีรายละเอียดการทำงานดังนี้

- บรรทัดที่ 5 เป็นการเรียกฟังก์ชัน pow( ) เพื่อคำนวณค่า 5^3 ซึ่งทำให้ได้ผลลัพธ์ 125
- บรรทัดที่ 6 เป็นการเรียกฟังก์ชัน sqrt( ) เพื่อหารากที่สองของค่า 144 ซึ่งผลลัพธ์ที่ได้คือ 12
- บรรทัดที่ 7 เป็นการเรียกฟังก์ชัน fabs( ) เพื่อหาค่าสัมบูรณ์ของ -10 ผลลัพธ์ที่ได้คือ 10

ตารางที่ 13.4 ฟังก์ชันมาตรฐานทางคณิตศาสตร์(ยกกำลังและค่าสัมบูรณ์)

ฟังก์ชันมาตรฐาน	คำอธิบาย
floor(x)	ฟังก์ชันสำหรับปัดเศษทศนิยมของ x ออกให้เป็นจำนวนเต็ม
ceil(x)	ฟังก์ชันสำหรับปัดเศษทศนิยมของ x ขึ้นให้เป็นจำนวนเต็ม
round(x)	ฟังก์ชันสำหรับปัดเศษทศนิยมของ x ให้เป็นจำนวนเต็ม ถ้าจุดทศนิยมมากกว่าหรือเท่ากับ .5 จะปัดขึ้นนอกนั้นปัดลง

ตัวอย่างโปรแกรมที่ 13.5

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char **argv) {
5     double  n1 = 5.2;
6     double  n2  = 5.5;
7     double  n3  = 5.8;
8
9      printf("N1 :  floor = %f,  round = %f, ceil = %f\n", floor(n1), round(n1), ceil(n1));
10     printf("N2 :  floor = %f,  round = %f, ceil = %f\n", floor(n2), round(n2), ceil(n2));
11     printf("N3 :  floor = %f,  round = %f, ceil = %f\n", floor(n3), round(n3), ceil(n3));
12 }
```

ผลการรัน

```
N1 :  floor = 5.000000,  round = 5.000000, ceil = 6.000000
N2 :  floor = 5.000000,  round = 6.000000, ceil = 6.000000
N3 :  floor = 5.000000,  round = 6.000000, ceil = 6.000000
```

ตัวอย่างโปรแกรมที่ 13.5 จะเห็นได้ว่าได้ประกาศตัวแปรประเภท double ไว้ด้วยกันทั้งหมด 3 ตัว คือ n1 มีค่า 5.2, n2 มีค่า 5.5 และ n3 มีค่า 5.8 บรรทัดที่ 9 - 11 ได้แสดงค่าของตัวแปร n1, n2 และ n3 โดยใช้ฟังก์ชัน floor( ), round( ) และ ceil( ) ตามลำดับ ผลการรันที่ได้คือ ทุกตัวแปรเมื่อผ่านฟังก์ชัน floor จะเป็นการปัดเศษลง, round จะปัดเศษลงสำหรับ n1 เนื่องจากจุดทศนิยมน้อยกว่า .5 สำหรับ n2, n3 จะเป็นการปัดขึ้น สุดท้ายคือฟังก์ชัน ceil จะเป็นการปัดเศษขึ้นทั้งหมด

ตารางที่ 13.5 ฟังก์ชันมาตรฐานทางคณิตศาสตร์(การสุ่มตัวเลข)

ฟังก์ชันมาตรฐาน	คำอธิบาย
rand()	ฟังก์ชันสุ่มตัวเลขเป็นจำนวนเต็มในย่านตั้งแต่ 0 ถึง 32767

ตัวอย่างโปรแกรมที่ 13.6

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char **argv) {
5      int i;
6      for(i = 0; i < 10; i++) {
7          printf("%d\n", rand());
8      }
9  }
```

ผลการรัน

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
```

ในภาษาซีสามารถจะเรียกฟังก์ชันมาตรฐานเพื่อทำการสุ่มตัวเลขได้ซึ่งก็คือฟังก์ชัน rand( ) รายละเอียดของฟังก์ชันแสดงในตารางที่ 13.5 การใช้งานฟังก์ชันแสดงในตัวอย่างโปรแกรมที่ 13.6 จะเห็นได้ว่าเป็นการวนลูปเพื่อแสดงค่าที่ได้จากการสุ่มทั้งหมด 10 ค่าออกทางหน้าจอ จากผลการรันค่าที่ได้จะสุ่มขึ้นมาโดยไม่ซ้ำกัน

13.2 ฟังก์ชันมาตรฐานเกี่ยวกับตัวอักขระและข้อความ

อักขระและข้อความเป็นอีกส่วนประกอบในโปรแกรมที่มีการใช้งานกันบ่อยครั้ง ดังนั้นจึงควรรู้จักฟังก์ชันมาตรฐานต่างๆที่ทำงานกับตัวอักขระและข้อความ เพื่อที่จะทำงานได้สะดวกมากขึ้น โดยฟังก์ชันมาตรฐานที่น่าสนใจเกี่ยวกับตัวอักขระจะรวมไว้ในตารางที่ 13.6 และฟังก์ชันมาตรฐานที่น่าสนใจเกี่ยวกับข้อความจะรวมไว้ในตารางที่ 13.7

ตารางที่ 13.6 ฟังก์ชันมาตรฐานที่จัดการกับตัวอักขระ

ฟังก์ชันมาตรฐาน	คำอธิบาย
tolower(ch)	ฟังก์ชันที่ใช้แปลงตัวอักษร ch ให้เป็นตัวพิมพ์เล็ก
toupper(ch)	ฟังก์ชันที่ใช้แปลงตัวอักษร ch ให้เป็นตัวพิมพ์ใหญ่
islower(ch)	ฟังก์ชันที่ใช้ตรวจสอบตัวอักษร ch ว่าเป็นตัวพิมพ์เล็กหรือไม่ คืนค่าจริง(ค่าไม่เท่ากับศูนย์) ถ้า ch เป็นตัวพิมพ์เล็ก นอกนั้นคืนค่าเท็จหรือศูนย์
isupper(ch)	ฟังก์ชันที่ใช้ตรวจสอบตัวอักษร ch ว่าเป็นตัวพิมพ์ใหญ่หรือไม่ คืนค่าจริง(ค่าไม่เท่ากับศูนย์) ถ้า ch เป็นตัวพิมพ์ใหญ่ นอกนั้นคืนค่าเท็จหรือศูนย์
isalnum(ch)	ฟังก์ชันจะให้ค่ากลับคืนออกมาเป็นจริง(ค่าไม่เท่ากับศูนย์) ถ้าตัวแปร ch มีค่าเป็น 'A' - 'Z' , 'a' - 'z' , และ '0' – '9' นอกนั้นจะให้ค่าเป็นเท็จหรือศูนย์
isalpha(ch)	ฟังก์ชันจะให้ค่ากลับคืนออกมาเป็นจริง(ค่าไม่เท่ากับศูนย์) ถ้าตัวแปร ch มีค่าเป็น 'A' - 'Z' , และ 'a' - 'z' นอกนั้นจะให้ค่าเป็นเท็จหรือศูนย์
isdigit(ch)	ฟังก์ชันจะให้ค่ากลับคืนออกมาเป็นจริง(ค่าไม่เท่ากับศูนย์) ถ้าตัวแปร ch มีค่าเป็น '0' - '9' นอกนั้นจะให้ค่าเป็นเท็จหรือศูนย์
ispunct(ch)	ฟังก์ชันจะให้ค่ากลับคืนออกมาเป็นจริง(ค่าไม่เท่ากับศูนย์) ถ้าตัวแปร ch มีค่าเป็นตัวเชื่อมต่างๆ เช่น #, <, @, > และอื่นๆ นอกนั้นจะให้ค่าเป็นเท็จหรือศูนย์

ตัวอย่างโปรแกรมที่ 13.7

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[]) {
5      int i;
6      char ch[12] = "Hello World";
7
8      for(i = 0; i < 11; i++) {
9          printf("%c", tolower(ch[i]));
10     }
11     printf("\n");
12     for(i = 0; i < 11; i++) {
13         printf("%c", toupper(ch[i]));
14     }
15 }
```

ผลการรัน

```
hello world
HELLO WORLD
```

ตัวอย่างโปรแกรมที่ 13.7 แสดงการใช้งานฟังก์ชัน tolower() และ toupper( ) ซึ่งโปรแกรมได้ประกาศตัวแปรประเภทข้อความไว้ในบรรทัดที่ 7 ซึ่งเป็นคำว่า “Hello World” ซึ่งมีตัวอักษรตัวพิมพ์เล็กและตัวพิมพ์ใหญ่ผสมกัน จากนั้น

- บรรทัดที่ 8 – 10 เป็นการวนลูปเพื่อนำตัวอักษรในข้อความออกมาทีละ 1 ตัวอักษรและผ่านเข้าฟังก์ชัน tolower( ) เพื่อแปลงตัวอักษรให้เป็นตัวพิมพ์เล็ก ถ้าตัวอักษรที่ผ่านเข้ามาในฟังก์ชันไม่ใช่ตัวพิมพ์ใหญ่ ก็จะไม่มีอะไรเกิดขึ้นฟังก์ชันจะคืนค่าตัวอักษรนั้นออกไป ทำให้ตัวพิมพ์เล็กและช่องว่างไม่มีการแปลงข้อมูลเกิดขึ้น ดังนั้นผลลัพธ์ที่ได้คือ “hello world”
- บรรทัดที่ 12 -14 ทำงานเช่นเดียวกันกับในบรรทัดที่ 8 – 10 แต่เป็นการแปลงตัวอักษรให้เป็นตัวพิมพ์ใหญ่ด้วยฟังก์ชัน toupper( ) ผลลัพธ์ที่ได้คือ “HELLO WORLD”

ตัวอย่างโปรแกรมที่ 13.8

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[]) {
5      char ch1 = 'A';
6      char ch2 = 'a';
7      char ch3 = '#';
8
9      printf("%c : uppercase? %s, lowercase? %s\n", ch1
10                                    , isupper(ch1)?"yes":"no "
11                                    , islower(ch1)?"yes":"no ");
12
13     printf("%c : uppercase? %s, lowercase? %s\n", ch2
14                                     , isupper(ch2)?"yes":"no "
15                                     , islower(ch2)?"yes":"no ");
16
17     printf("%c : uppercase? %s, lowercase? %s\n", ch3
18                                     , isupper(ch3)?"yes":"no "
19                                    , islower(ch3)?"yes":"no ");
20 }
```

ผลการรัน

```
 A : uppercase? yes, lowercase? no
a : uppercase? no , lowercase? yes
# : uppercase? no , lowercase? no
```

ตัวอย่างโปรแกรมที่ 13.8 แสดงการใช้งานฟังก์ชัน isupper( ) และ islower( ) เพื่อตรวจสอบตัวอักษรว่าเป็นตัวพิมพ์เล็กหรือตัวพิมพ์ใหญ่ จะเห็นว่าในโปรแกรมจะประกาศตัวแปร 3 ตัวคือ ch1, ch2 และ ch3 ที่มีค่าคือ 'A', 'a' และ '#' ตามลำดับ ผลลัพธ์ที่ได้คือ A เป็นตัวพิมพ์ใหญ่, a เป็นตัวพิมพ์เล็ก และ # ไม่ได้เป็นทั้งตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก (uppercase?no , lowercase? no)

ตัวอย่างโปรแกรมที่ 13.9

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[]) {
5      int i;
6      char ch[] ={'P', '8', '#'};
7
8      for(i = 0; i < 3; i++) {
9          printf("%c :",  ch[i]);
10         printf("AlphaNumber? %s", isalnum(ch[i])?"yes   ":"no    ");
11         printf("Alphabet? %s",        isalpha(ch[i])?"yes   ":"no    ");
12         printf("Number? %s",          isdigit(ch[i])?"yes   ":"no    ");
13         printf("Punctuation? %s\n",   ispunct(ch[i])?"yes   ":"no    ");
14     }
15 }
```

ผลการรัน

```
P :AlphaNumber? yes   Alphabet? yes   Number? no    Punctuation? no
8 :AlphaNumber? yes   Alphabet? no    Number? yes   Punctuation? no
# :AlphaNumber? no    Alphabet? no    Number? no    Punctuation? yes
```

ตัวอย่างโปรแกรมที่ 13.9 แสดงการใช้งานฟังก์ชัน isalnum( ), isalpha( ), isdigit( ), และ ispunct( ) เพื่อตรวจสอบตัวอักษรว่าเป็นตัวอักษรลักษณะไหน ตัวหนังสือ, ตัวเลข หรือ ตัวสัญลักษณ์ จะเห็นว่าในโปรแกรมจะประกาศตัวแปรอาร์เรย์ ch ที่มีสมาชิก 3 ตัวคือ ‘P’, ‘8’ และ ‘#’

ผลลัพธ์ที่ได้จะเห็นว่า

- P เป็นตัวอักษร จึงมีผลให้ isalnum, isalpha คืนค่าจริง และ isdigit, ispunct คืนค่าเท็จ
- 8 เป็นตัวเลข จึงมีผลให้ isalnum, isdigit คืนค่าจริง และ isalpha, ispunct คืนค่าเท็จ
- # เป็นสัญลักษณ์ จึงมีผลให้ ทุกฟังก์ชันที่ทำการตรวจสอบคืนค่าเท็จ ยกเว้นฟังก์ชัน ispunct( ) คืนค่าจริง

ตารางที่ 13.7 ฟังก์ชันมาตรฐานที่จัดการกับข้อความ

ฟังก์ชันมาตรฐาน	คำอธิบาย
strcpy(str1, str2)	ฟังก์ชันสำหรับคัดลอก (copy) ข้อความจากตัวแปร str2 ไปเก็บยังตัวแปร str1
strcat(str1, str2)	ฟังก์ชันสำหรับเชื่อมต่อข้อความโดยการนำข้อความในตัวแปร str2 ไปต่อท้ายข้อความในตัวแปร str1 โดยผลลัพธ์จะเก็บไว้ในตัวแปร str1
strcmp(str1, str2)	ฟังก์ชันสำหรับเปรียบเทียบข้อความในตัวแปร str1 และ str2 ถ้าข้อความเหมือนกันจะคืนค่าศูนย์
strlen(str)	ฟังก์ชันสำหรับหาความยาวของข้อความใน str

ตารางที่ 13.7 แสดงฟังก์ชันมาตรฐานที่ใช้ในการจัดการกับข้อความ ซึ่งจะมีฟังก์ชันหลักๆ อยู่ทั้งหมด 4 ฟังก์ชันคือ strcpy( ), strcat( ), strcmp( ), และ strlen( ) การใช้งานการฟังก์ชันทั้ง 4 จะแสดงในตัวอย่างโปรแกรมที่ 13.10 และ 13.11

ตัวอย่างโปรแกรมที่ 13.10

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char *argv[]) {
4      char str1[] = “C Language”;
5      char str2[] = “ is easy”;
6      char str3[24];
7      strcpy(str3,  str1);
8      strcat(str1,  str2);
9      printf("%s\n",  str1);
10     printf("%s\n",  str3);
11 }
```

ผลการรัน

```
C Language
C Language is easy
```

จากตัวอย่างโปรแกรมที่ 13.10 แสดงการใช้งานฟังก์ชัน strcpy( ) และ strcat ( ) โดยโปรแกรมมีการประกาศตัวแปรประเภทข้อความ str1 โดยมีการกำหนดค่าเริ่มต้นให้คือ “C Language”, ตัวแปรประเภทข้อความ str2 โดยมีการกำหนดค่าเริ่มต้นให้คือ “ is easy” และ ตัวแปรประเภทข้อความ str3 ที่ไม่มีการกำหนดค่าเริ่มต้นให้

- บรรทัดที่ 7 เป็นการใช้ฟังก์ชัน strcpy เพื่อคัดลอกค่าในตัวแปร str1 มาเก็บไว้ยังตัวแปร str3 ซึ่งทำให้สิ้นสุดบรรทัดนี้ค่าในตัวแปร str3 จะมีค่าเท่ากับ “C Language”
- บรรทัดที่ 8 เป็นการใช้ฟังก์ชัน strcat เพื่อนำข้อความที่อยู่ใน str2 (“ is easy”) ไปเชื่อมต่อที่ท้ายข้อความใน str1 (“C Language”) เป็นเก็บผลลัพธ์ไว้ที่ str1 ดังนั้นเมื่อสิ้นสุดบรรทัดนี้ตัวแปร str1 จะเก็บค่า “C Language is easy”
- บรรทัดที่ 9 – 10 แสดงค่าของตัวแปร str1 และ str3 ออกทางหน้าจอด้วยฟังก์ชัน printf( ) ซึ่งจะได้ผลลัพธ์คือ “C Language” และ “C Language is easy” ตามลำดับ

ตัวอย่างโปรแกรมที่ 13.11

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char *argv[]) {
4      char str1[] = “C Language”;
5      char str2[] = “c language”;
6      if(!strcmp(str1,  str2)) {
7              printf(“str1 = str2\n”);
8      } else {
9             printf(“str1 != str2\n”);
10     }
11     printf("Length of str1 = %d\n",  strlen(str1));
12 }
```

ผลการรัน

```
str1 != str2
Length of str1 = 10
```

จากตัวอย่างโปรแกรมที่ 13.11 แสดงการใช้งานฟังก์ชัน strcmp( ) และ strlen ( ) โดยโปรแกรมมีการประกาศตัวแปรประเภทข้อความ str1 และตัวแปรประเภทข้อความ str2 โดยมีการกำหนดค่าเริ่มต้นคือ “C Language” และ “c language” ตามลำดับ

- บรรทัดที่ 6 – 10 เป็นการเปรียบเทียบข้อมูลในตัวแปรประเภทข้อความระหว่าง str1 และ str2 ซึ่งถ้าเท่ากันจะแสดงคำว่า “str1 = str2” และถ้าไม่เท่ากันจะแสดงคำว่า “str1 != str2” เนื่องด้วยในภาษาซีข้อมูลที่เป็นตัวพิมพ์เล็กมีค่าไม่เท่ากับข้อมูลที่เป็นตัวพิมพ์ใหญ่ดังนั้นผลลัพธ์ของเงื่อนไขนี้จึงแสดงคำว่า “str1 != str2” ออกทางหน้าจอ
- บรรทัดที่ 11 เป็นการเรียกใช้ฟังก์ชัน strlen( ) เพื่อแสดงความยาวของข้อความที่เก็บไว้ในตัวแปร str1 ซึ่งในแปรแกรม str1 เก็บคำว่า “C Language” เมื่อนับจำนวนตัวอักษรจะได้ 9 ตัว รวมกับช่องว่างอีก 1 ตัว จึงทำให้ผลลัพธ์ของฟังก์ชันมีค่าเท่ากับ 10

13.3 ฟังก์ชันอื่นทั่วๆไป

นอกจากฟังก์ชันสำหรับการคำนวณทางคณิตศาสตร์และการจัดการกับตัวอักขระและข้อความแล้ว ในภาษาซียังมีฟังก์ชันอื่นๆ ให้ใช้งานอีกมากมาย จะขอยกตัวอย่างสองฟังก์ชันคือ ฟังก์ชันการหาขนาดพื้นที่หน่วยความจำที่ตัวแปรใช้(sizeof) และ ฟังก์ชันสำหรับการออกจากการทำงานของ โปรแกรม (exit) คำอธิบายของทั้งสองฟังก์ชันแสดงในตารางที่ 13.8 การใช้งานฟังก์ชัน sizeof( ) จะแสดงในตัวอย่างโปรแกรมที่ 13.12 และการใช้งานฟังก์ชัน exit จะแสดงในตัวอย่างโปรแกรมที่ 13.13

ตารางที่ 13.8 ฟังก์ชันมาตรฐานที่จัดการกับข้อความ

ฟังก์ชันมาตรฐาน	คำอธิบาย
sizeof(x)	ฟังก์ชันสำหรับหาขนาดพื้นที่หน่วยความจำที่ใช้เก็บข้อมูลของตัวแปร x โดยมีหน่วยที่คืนเป็นไบต์
exit(n)	ฟังก์ชันสำหรับการออกจากการทำงานของโปรแกรม โดย n จะเป็นค่าที่ฟังก์ชัน main จะคืนให้กับระบบ ปกติถ้าเป็นการออกจากโปรแกรมโดยไม่มีค่าผิดพลาดจะกำหนดให้ n = 0

ตัวอย่างโปรแกรมที่ 13.12

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char *argv[]) {
4      double x;
5      printf(“Size of short = %d\n”, sizeof(short));
6      printf(“Size of int     = %d\n”, sizeof(int));
7      printf(“Size of long  = %d\n”, sizeof(long));
8      printf(“Size of float     = %d\n”, sizeof(float));
9      printf("Size of double = %d\n",  sizeof(double));
10     printf(“Size of x          = %d\n”, sizeof(x));
11 }
```

ผลการรัน

```
Size of short = 2
Size of int     = 4
Size of long  = 4
Size of float     = 4
Size of double = 8
Size of x         = 8
```

จากตัวอย่างโปรแกรมที่ 13.12 แสดงการใช้งานของฟังก์ชัน sizeof( ) โดยในโปรแกรมตั้งแต่บรรทัดที่ 5 – บรรทัดที่ 9 เป็นการหาค่าหน่วยความจำของประเภทข้อมูลชนิด short, int, long, float, และ double ตามลำดับ จากผลการรันจะเห็นว่า ตัวแปรประเภท short, int, long, float และ double ใช้หน่วยความจำ 2 ไบต์, 4 ไบต์, 4 ไบต์, 4 ไบต์ และ 8 ไบต์ตามลำดับ

บรรทัดที่ 10 ของโปรแกรมแสดงให้เห็นว่า สามารถที่จะหาหน่วยความจำที่ใช้ของตัวแปรได้เลย โดยโปรแกรมประกาศตัวแปรประเภท double ชื่อ x หลังจากการใช้ sizeof(x) จะพบว่าตัวแปร x ใช้พื้นที่หน่วยความจำ 8 ไบต์ ซึ่งจะเท่ากับประเภทข้อมูลของตัวแปรที่เป็น double

ตัวอย่างโปรแกรมที่ 13.13

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char *argv[]) {
4      printf("Hello\n");
5      printf("Hello\n");
6      printf("Hello\n");
7      exit(0);
8      printf("Hello\n");
9      printf("Hello\n");
10 }
```

ผลการรัน

```
Hello
Hello
Hello
```

จากตัวอย่างโปรแกรมที่ 13.13 จะเป็นโปรแกรมที่แสดงให้เห็นการทำงานของฟังก์ชัน exit() อย่างง่ายๆ โดยในโปรแกรมบรรทัดที่ 4 – 6 และ บรรทัดที่ 8 – 9 จะเป็นการแสดงคำว่า “Hello” ออกทางหน้าจอด้วยฟังก์ชัน printf( ) โดยปกติเมื่อมีคำสั่ง printf( ) ทั้งหมด 5 คำสั่งจะมีการ แสดงผลคำว่า “Hello” ทั้งหมด 5 ครั้ง แต่เนื่องด้วยบรรทัดที่ 7 เป็นคำสั่ง exit(0) ซึ่งเป็นคำสั่งให้จบ การทำงานของโปรแกรม ดังนั้นฟังก์ชันที่ 8 – 9 จะไม่เคยถูกเรียกใช้งานเลย จึงมีเหลือแค่การแสดงผลบนหน้าจอของฟังก์ชัน printf() ของบรรทัดที่ 4 - 6

บทที่ 14
สตรัคเจอร์ และยูเนียน

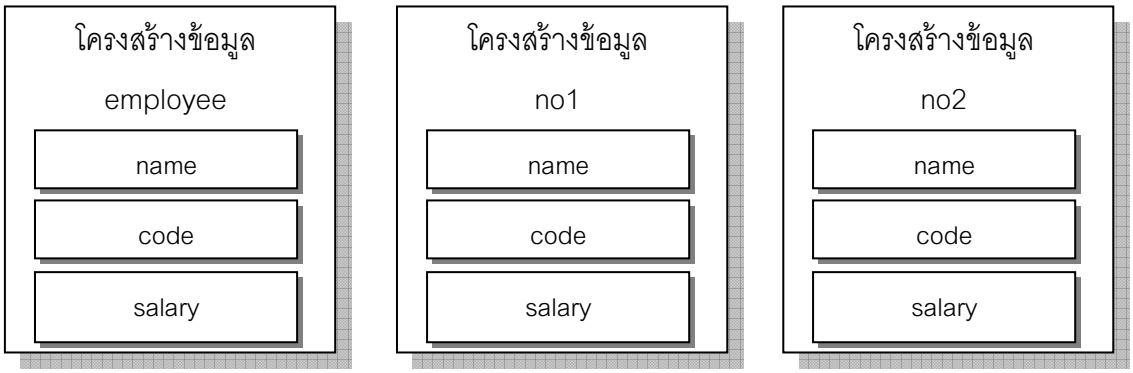
ถ้ามีข้อมูลอยู่จำนวนหนึ่งซึ่งเป็นข้อมูลต่างชนิดกันแต่มีความเกี่ยวข้องกัน เช่น ข้อมูลของนักศึกษาซึ่งอาจจะประกอบไปด้วย ชื่อ, นามสกุล, อายุ, เพศ, และชั้นเรียน การเก็บข้อมูลของนักศึกษาจะต้องสร้างตัวแปรขึ้นมา 5 ตัวต่อนักศึกษาหนึ่งคน ถ้านักศึกษามีจำนวนมากก็จำเป็นต้องสร้างตัวแปรมากขึ้นตามไปด้วยซึ่งอาจจะเกิดความสับสนในการเรียกใช้งานตัวแปรเหล่านั้น การนำโครงสร้างข้อมูลหรือสตรัคเจอร์มาใช้จะช่วยให้การทำงานในลักษณะนี้ง่ายขึ้น

14.1 สตรัคเจอร์ (Structure)

สตรัคเจอร์ หรือโครงสร้างข้อมูล (Structure) เป็นการกำหนดชนิดของตัวแปรขึ้นมาใหม่ โดยนำตัวแปรชนิดพื้นฐานในภาษาซี อย่างเช่น int, char, และ float มาประกอบกันเป็นโครงสร้างของตัวแปร ชนิดใหม่ ยกตัวอย่างเช่น ต้องการสร้างตัวแปรชนิดใหม่โดยใช้ชื่อ employee เพื่อเก็บข้อมูลของ พนักงาน ภายในประกอบด้วยตัวแปรพื้นฐานดังต่อไปนี้

```
char  name[24];      //เก็บชื่อพนักงาน
int   code;          //เก็บรหัสพนักงาน
float salary;        //เก็บเงินเดือนพนักงาน
```

ต่อไปเมื่อต้องการจะเก็บข้อมูลของพนักงานแต่ละคน ก็เพียงแต่ประกาศตัวแปรชนิด employee ขึ้นมา โดยอาจจะใช้ชื่อตัวแปร no1, no2, หรือ อื่นๆ ซึ่งตัวแปรที่ประกาศขึ้นมาจากโครงสร้างข้อมูลในลักษณะนี้จะเรียกว่าตัวแปรชนิดโครงสร้าง (Struct) ดังนั้นตัวแปร no1 และ no2 จึงเป็นตัวแปร ชนิดโครงสร้าง และภายในตัวแปร no1 และ no2 จะมีโครงสร้างภายในเหมือนโครงสร้างข้อมูลของ employee ทุกประการดังแสดงในรูป 14.1

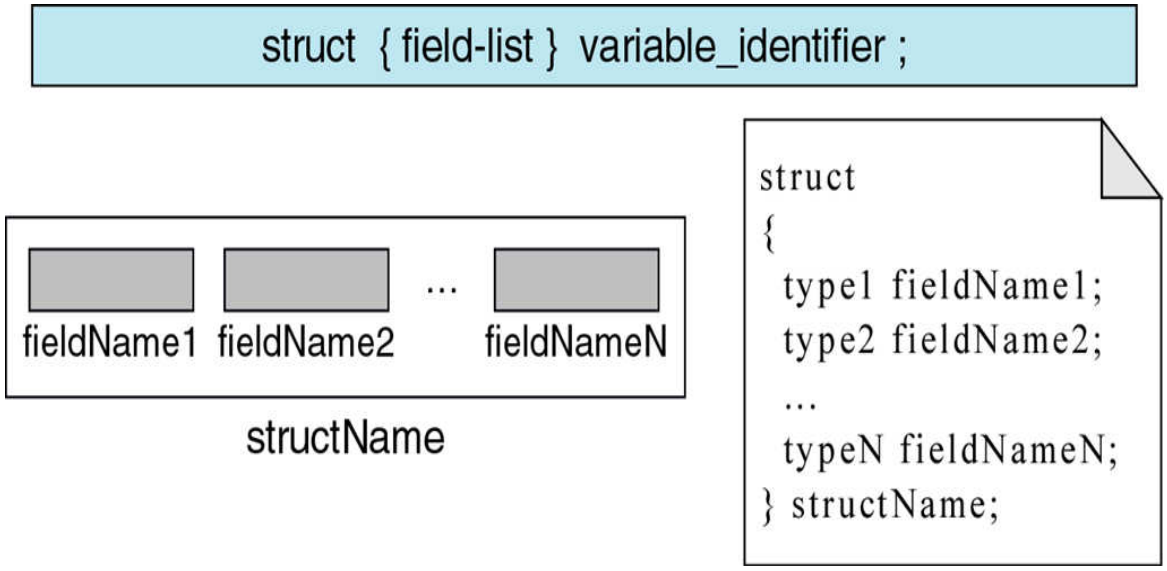


รูปที่ 14.1 โครงสร้างข้อมูล employee และตัวแปรชนิดโครงสร้าง

14.1.1 การประกาศตัวแปรชนิดโครงสร้าง (Struct)

ในภาษาซีสามารถประกาศตัวแปรชนิดโครงสร้างได้ 3 วิธี

วิธีที่ 1 เป็นการประกาศตัวแปรโดยตรง รูปแบบการใช้งานดังรูปที่ 14.2

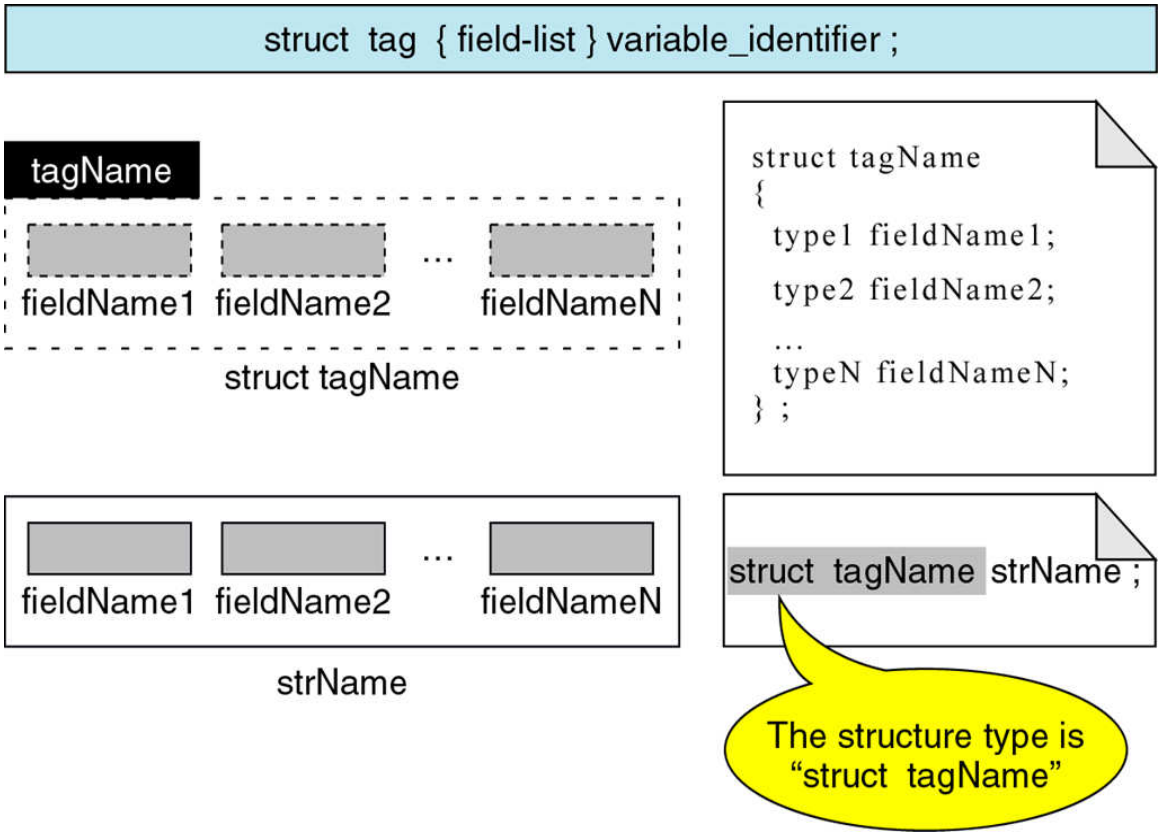


รูปที่ 14.2 การประกาศตัวแปรชนิดโครงสร้างวิธีที่ 1

วิธีนี้เป็นการประกาศตัวแปรชนิดโครงสร้างแบบง่ายดีสุด เป็นการประกาศโดยตรงให้กับตัวแปรเลย ตัวอย่างการใช้งานมีลักษณะดังต่อไปนี้

```
struct {
    char   name[24];
    int    code;
    float  salary;
} no1,  no2;
```

เป็นการประกาศตัวแปรชนิดโครงสร้าง no1 และ no2 ซึ่งโครงสร้างภายใน จะสามารถเก็บค่า name ที่มีประเภทข้อมูลแบบข้อความ , code ที่สามารถเก็บข้อมูลประเภทจำนวนเต็ม และ salary ซึ่งใช้ในการเก็บข้อมูลประเภทจำนวนจริงได้

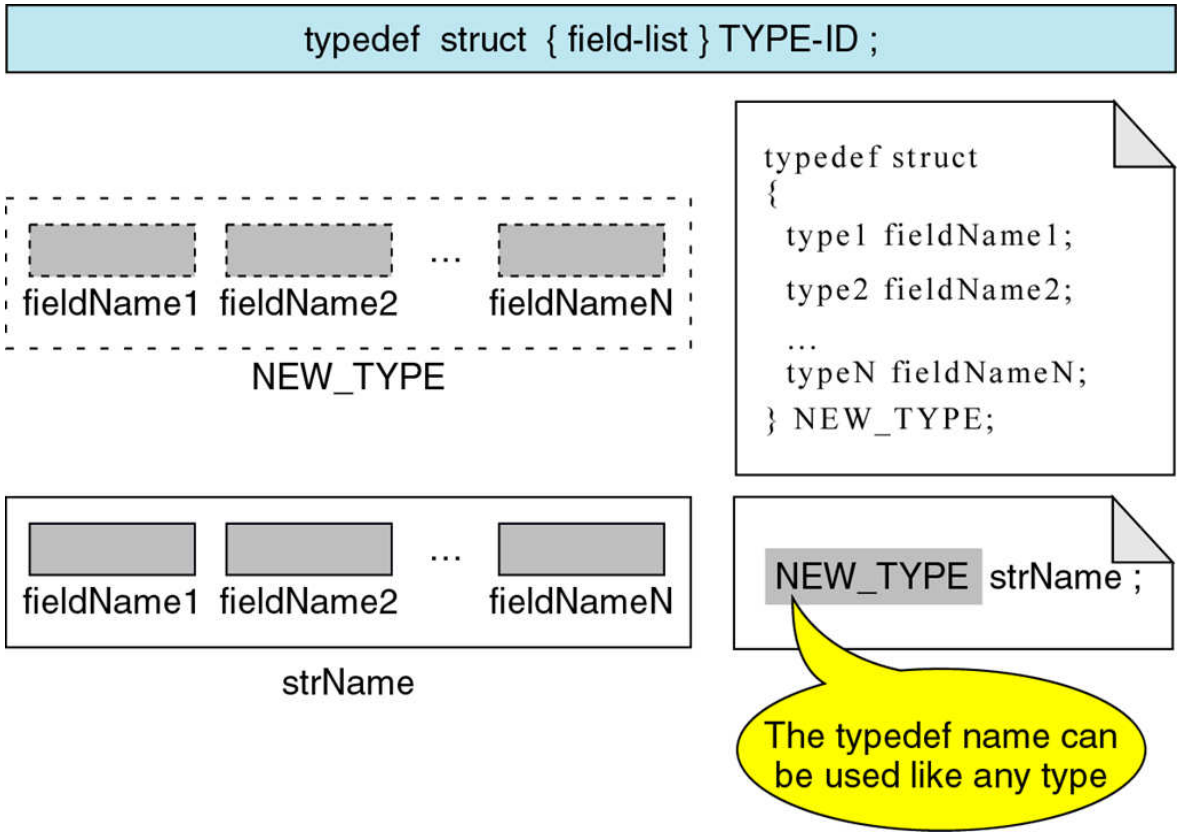


รูปที่ 14.3 การประกาศตัวแปรชนิดโครงสร้างวิธีที่ 2

วิธีที่ 2 สามารถประกาศตัวแปรชนิดโครงสร้างได้ดังรูปที่ 14.3 ซึ่งเป็นการกำหนดชื่อ tag หรือชื่อ ให้กับประเภทโครงสร้างข้อมูลที่สร้างขึ้นมาใหม่ ดังนั้นเมื่อต้องการประกาศตัวแปรที่มีประเภทข้อมูลแบบนี้ก็สามารถทำได้ง่ายขึ้น เพียงแค่ประกาศ struct ตามด้วยชื่อ tag และชื่อตัวแปรที่ต้องการ ลักษณะการทำงานในการสร้างโครงสร้างข้อมูล employee สามารถ ทำได้ดังนี้

```
struct  employee {
    char   name[24];
    int     code;
    float   salary;
} ;

struct  employee  no1,  no2;
```



รูปที่ 14.4 การประกาศตัวแปรชนิดโครงสร้างวิธีที่ 3

วิธีที่ 3 สามารถประกาศตัวแปรชนิดโครงสร้างได้ดังรูปที่ 14.4 ซึ่งเป็นการใช้ตัวดำเนินการ typedef ซึ่งเป็นการกำหนดชื่อใหม่ให้กับประเภทตัวแปร มีข้อดีคือต่อไปสามารถเรียกใช้งานได้ง่ายและสะดวกขึ้น การสร้างโครงสร้างนั้นเพียงแต่ใช้ตัวดำเนินการ typedef อยู่ก่อนการประกาศ struct และสิ้นสุดด้วยการกำหนดชื่อประเภทข้อมูลใหม่ให้กับ struct ตัวนี้ การประกาศตัวแปรโครงสร้างสามารถประกาศโดยใช้ ชื่อประเภทข้อมูลชนิดใหม่ ตามด้วยชื่อของตัวแปรลักษณะการทำงาน สามารถทำได้ดังนี้

```
typedef struct {
    char   name[24];
    int    code;
    float  salary;
} employee;

employee  no1,  no2;
```

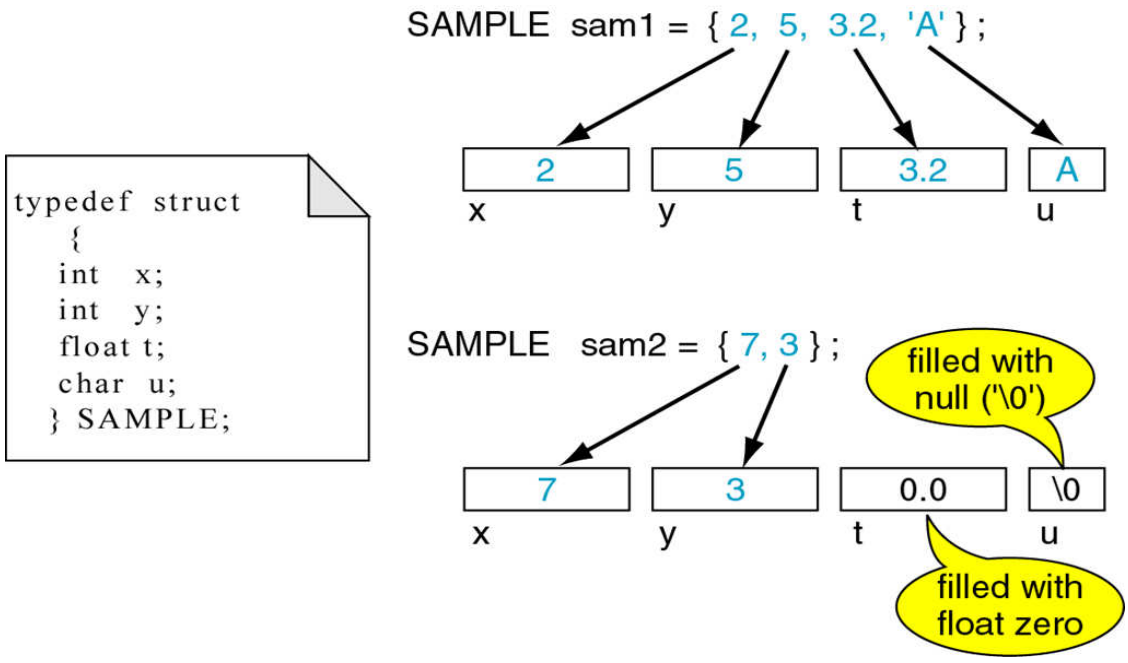
วิธีที่ 1	วิธีที่ 2	วิธีที่ 3
struct { char name[24]; int code; float salary; } no1, no2;	struct employee { char name[24]; int code; float salary; } ; struct employee no1, no2;	typedef struct { char name[24]; int code; float salary; } employee; employee no1, no2;

รูปที่ 14.5 เปรียบเทียบวิธีการประกาศตัวแปรโครงสร้างทั้ง 3 วิธี

รูปที่ 14.5 แสดงการประกาศชนิดโครงสร้าง no1 และ no2 ที่มีโครงสร้างข้อมูลคือ name ที่เป็นข้อความ, code เป็นจำนวนเต็ม และ salary เป็นจำนวนจริงในรูปแบบต่างๆ ทั้ง 3 วิธี

14.1.2 การอ้างอิงข้อมูลสำหรับตัวแปรโครงสร้าง

การกำหนดข้อมูลเริ่มต้นให้กับตัวแปรโครงสร้างสามารถกำหนดได้คล้ายกับการกำหนดค่าตัวแปรเริ่มต้นให้กับตัวแปรอาร์เรย์ รูปที่ 14.6 แสดงการกำหนดค่าเริ่มต้นให้กับตัวแปรชนิดโครงสร้าง



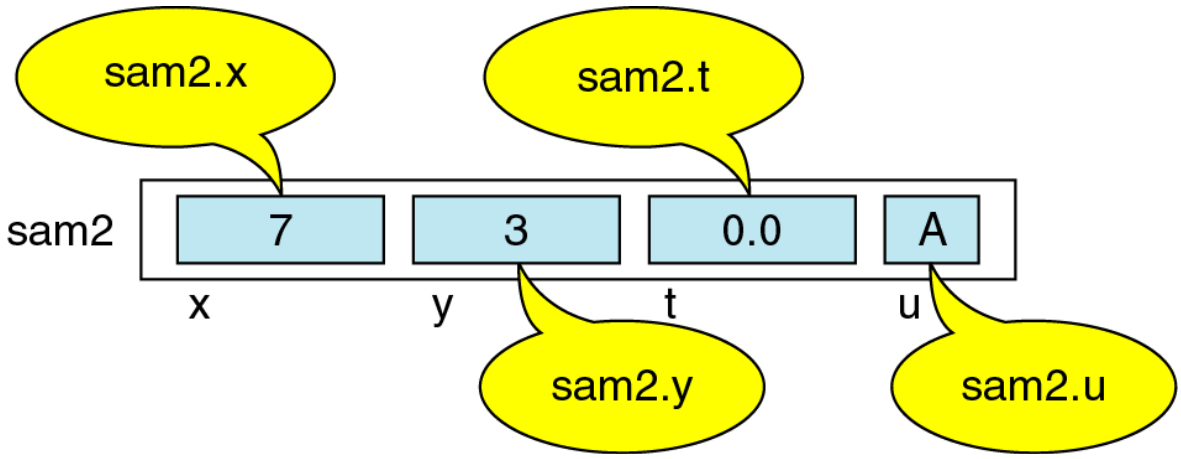
รูปที่ 14.6 การกำหนดค่าเริ่มต้นให้กับตัวแปรชนิดโครงสร้าง

รูปที่ 14.6 มีการประกาศประเภทของชนิดข้อมูลใหม่ชื่อ SAMPLE อันที่จริงแล้วก็คือประเภทข้อมูลชนิดโครงสร้าง ที่โครงสร้างภายในประกอบด้วยตัวแปร x, y, t, และ u ซึ่งเป็นประเภทข้อมูลชนิดจำนวนเต็ม, จำนวนเต็ม, จำนวนจริง, และ ตัวขระ ตามลำดับ จากตัวอย่างมีการประกาศตัวแปรสองตัว คือ sam1 และ sam2 ด้วยคำสั่ง SAMPLE sam1 และ SAMPLE sam2 จะเห็นได้ว่า sam1 มีการกำหนดค่าเริ่มต้นคือ { 2, 5, 3.2, 'A' } ในที่นี้จะทำให้ตัวแปรภายในเก็บค่าต่างๆ ดังนี้ x เก็บค่า 2, y เก็บค่า 5, t เก็บค่า 3.2 และ u เก็บค่า 'A' ในขณะที่ sam2 มีการกำหนดค่าเริ่มต้น คือ { 7, 3 } ส่งผลให้ x มีค่า 7 และ y มีค่า 3 สำหรับตัวแปรภายใน t และ y ไม่มีการกำหนดค่าเริ่มต้น เพราะฉนั้นในส่วนนี้ภาษาซีจะใส่ค่าเริ่มต้น 0.0 กับตัวแปร t และ '\0' ให้กับตัวแปร u

เมื่อมีการสร้างตัวแปรประเภทโครงสร้างซึ่งมีตัวแปรภายในแล้วนั้น จำเป็นต้องทราบถึงวิธีการเข้าถึงข้อมูลต่างๆ ตัวแปรชนิดพื้นฐานที่เป็นส่วนประกอบภายในตัวแปรชนิดโครงสร้างจะใช้คำเรียกว่า *สมาชิก* โดยการอ้างถึงสมาชิกในตัวแปรชนิดโครงสร้างจะมีรูปแบบดังนี้

struct Name.variableName;
• Name : ชื่อของตัวแปรชนิดโครงสร้าง • . : เครื่องหมายจุดใส่คั้นระหว่างชื่อของตัวแปรชนิดโครงสร้างและตัวแปรสมาชิก • variableName : ชื่อของตัวแปรสมาชิก

รูปที่ 14.7 แสดงวิธีการเข้าถึงข้อมูลภายในของตัวแปร sam2 ที่มีประเภทข้อมูลแบบโครงสร้าง SAMPLE ที่ประกอบไปด้วย sam2.x, sam2.y, sam2.t, sam2.u



รูปที่ 14.7 วิธีการอ้างอิงตัวแปรสมาชิกในตัวแปรชนิดโครงสร้าง

ตัวอย่างโปรแกรมที่ 14.1

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      struct  product {
4              int     code;
5              char  productName[64];
6              float   price;
7      } computer;
8
9      computer.code = 1000;
10     strcpy(computer.productName,  “Core 2 duo”);
11     computer.price = 18000.00;
12 }
```

ตัวอย่างโปรแกรมที่ 14.1 แสดงการสร้างตัวแปรโครงสร้างและการอ้างอิงค่าของสมาชิกภายในตัวแปรโครงสร้าง

- บรรทัดที่ 3 – 7 เป็นการประกาศตัวแปรชนิดโครงสร้างชื่อ computer เป็นตัวแปร ที่มีโครงสร้างภายในแบบ struct product โดยสมาชิกภายในตัวแปรชนิดโครงสร้างนั้น ถูกประกาศในบรรทัดที่ 4 – 6 ซึ่งก็คือ
    - code ตัวแปรประเภทจำนวนเต็ม
    - productName ตัวแปรประเภทข้อความ
    - price ตัวแปรประเภทจำนวนจริง
- บรรทัดที่ 9 เป็นการอ้างอิงสมาชิกภายในชื่อ code ของตัวแปร computer สามารถอ้างอิงผ่าน computer.code และใส่ค่า 1000 เข้าไปในตัวแปร
- บรรทัดที่ 10 เป็นการอ้างอิงสมาชิกภายในชื่อ productName ของตัวแปร computer สามารถอ้างอิงผ่าน computer.productName และใส่ค่า “Core 2 duo” เข้าไปในตัวแปร
- บรรทัดที่ 11 เป็นการอ้างอิงสมาชิกภายในชื่อ price ของตัวแปร computer สามารถอ้างอิงผ่าน computer.price และใส่ค่า 18000.00 เข้าไปในตัวแปร

ตัวอย่างโปรแกรมที่ 14.2

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      typedef struct  {
4              int     code;
5              char  name[64];
6              float   salary;
7      } Employee;
8
9     Employee   emp;
10    printf("Enter code : " );   scanf("%d",  &(emp.code));
11    printf("Enter name : ");   gets(emp.name);
12    printf("Enter salary : ");   scanf("%f",  &(emp.salary));
13
14   printf("%d\t%s\t%f\n",  emp.code,  emp.name,  emp.salary);
15  }
```

ผลการรัน

Enter code : ***1000***

Enter name : ***Choopan Rattanapoka***

Enter salary : ***20000***

1000 Choopan Rattanapoka 20000.000000

รายละเอียดการทำงานของตัวอย่างโปรแกรมที่ 14.2 มีดังนี้

- บรรทัดที่ 3 – 7 ประกาศตัวแปรชนิดใหม่ชื่อ Employee เป็นตัวแปรชนิดโครงสร้างที่ประกอบไปด้วยสมาชิกภายในคือ code, name, และ salary ที่มีประเภทข้อมูลคือ จำนวนเต็ม, ข้อความ และ จำนวนจริงตามลำดับ
- บรรทัดที่ 9 ประกาศตัวแปร emp ประเภทข้อมูลคือ Employee
- บรรทัดที่ 10 - 12 เป็นการรับค่าจากผู้ใช้ผ่านทางแป้นพิมพ์เข้ามาเก็บไว้ยังสมาชิกในตัวแปรชนิดโครงสร้าง
- บรรทัดที่ 14 พิมพ์ข้อมูลที่เก็บไว้ในสมาชิกของตัวแปรโครงสร้างออกทางหน้าจอ

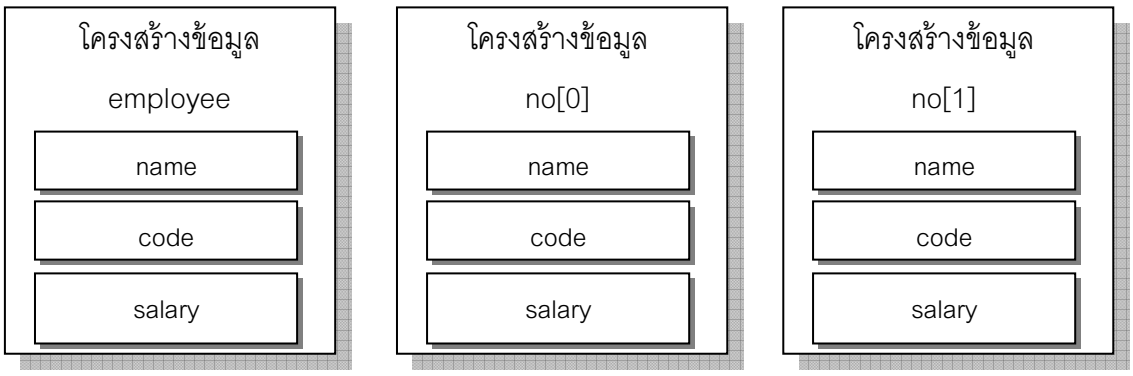
14.1.3 อาร์เรย์ของตัวแปรโครงสร้าง

ในกรณีที่ต้องการใช้ตัวแปรชนิดโครงสร้างแบบเดียวกันหลายๆตัว ผู้พัฒนาโปรแกรมสามารถสร้างตัวแปรอาร์เรย์ชนิดโครงสร้างขึ้นมาได้ เช่นเดียวกันกับการสร้างตัวแปรอาร์เรย์กับชนิดข้อมูลพื้นฐานอื่นๆ ดังที่แสดงต่อไปนี้

```
struct  employee {
    char   name[24];
    int     code;
    float   salary;
} ;

struct  employee  no[2];
```

เมื่อเขียนคำสั่งดังตัวอย่างข้างต้น ก็เท่ากับเราสร้างตัวแปรชนิดโครงสร้างขึ้นมาทั้งหมด 2 ตัว คือ no[0] และ no[1] โดยที่ตัวแปรแต่ละตัวนั้นจะมีสมาชิกภายในเหมือนกับโครงสร้างข้อมูล employee ทุกประการดังรูปที่ 14.8



รูปที่ 14.8 โครงสร้างข้อมูล employee และตัวแปรชนิดโครงสร้างแบบอาร์เรย์

การอ้างอิงข้อมูลของสมาชิกในตัวแปรโครงสร้างแบบอาร์เรย์ก็สามารถอ้างอิงได้ตามปกติ ยกตัวอย่างเช่น ต้องการเข้าถึงข้อมูลของสมาชิก code ในตัวแปรโครงสร้าง no[0] ก็สามารถอ้างอิงในลักษณะ no[0].code ได้เลย

ตัวอย่างโปรแกรมที่ 14.3

```
1  #include<stdio.h>
2  int main(int argc, char **argv) {
3      int  i;
4      typedef struct  {
5              int     code;
6              char  name[64];
7      } Employee;
8
9     Employee   emp[3];
10     for(i = 0; i < 3; i++) {
11             printf("Enter code : " );   scanf("%d",  &(emp[i].code));
12             printf("Enter name : ");   gets(emp[i].name);
13             printf("\n");
14     }
15     for(i = 0;  i < 3; i++)
16             printf("%d:%d\t%s\n",  i, emp[i].code,  emp[i].name);
17 }
```

ผลการรัน

Enter code : ***1000***

Enter name : ***Choopan Rattanapoka***

Enter code : ***1001***

Enter name : ***Somchai Songchauy***

Enter code : ***3000***

Enter name : ***Siriluk Peungrod***

0:1000 Choopan Rattanapoka

1:1001 Somchai Songchauy

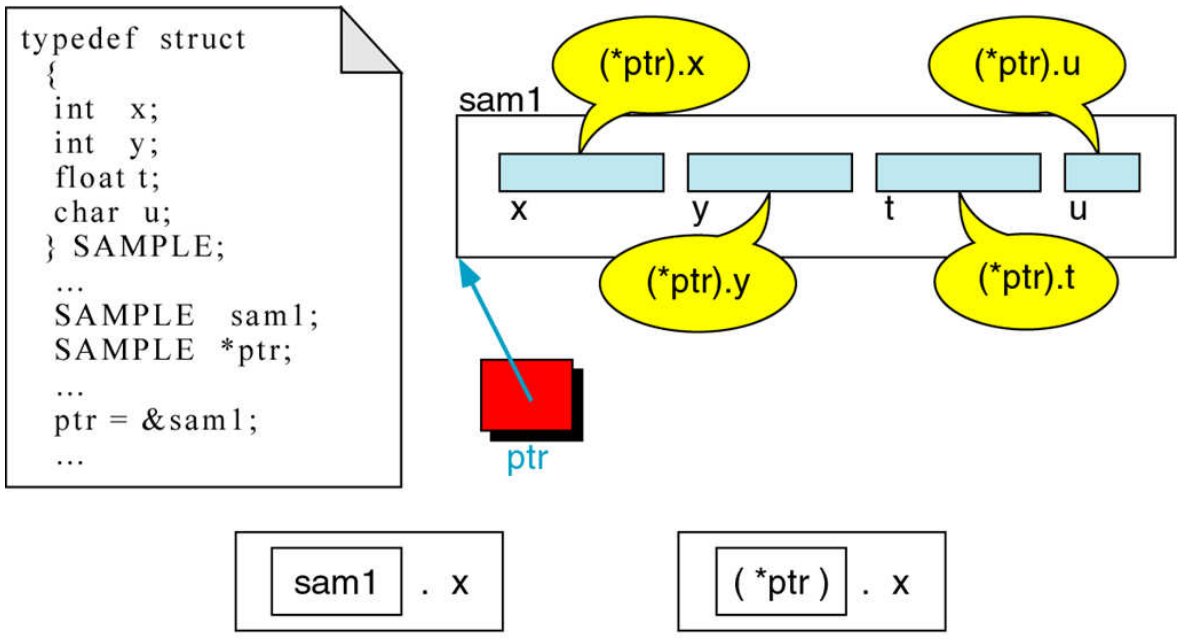
2:3000 Siriluk Peungrod

รายละเอียดการทำงานของตัวอย่างโปรแกรมที่ 14.3 มีดังนี้

- บรรทัดที่ 4 – 7 ประกาศตัวแปรชนิดใหม่ชื่อ Employee เป็นตัวแปรชนิดโครงสร้างที่ประกอบไปด้วยสมาชิกภายในคือ code และ name ที่มีประเภทข้อมูลคือ จำนวนเต็ม, ข้อความ ตามลำดับ
- บรรทัดที่ 9 ประกาศตัวแปร emp ประเภทข้อมูลคือ Employee แบบอาร์เรย์ที่มีสมาชิก 3 ตัว
- บรรทัดที่ 10 - 14 เป็นการรับค่าจากผู้ใช้ผ่านทางแป้นพิมพ์เข้ามาเก็บไว้ยังสมาชิกในตัวแปรชนิดโครงสร้าง เป็นการวนลูปทั้งหมด 3 รอบโดยจะเก็บข้อมูลให้กับ emp[0] , emp[1] และ emp[2] ตามลำดับ
- บรรทัดที่ 15-16 เป็นการวนลูปจำนวน 3 รอบเพื่อพิมพ์ค่าที่เก็บอยู่ในสมาชิกของตัวแปรโครงสร้าง emp[0], emp[1] และ emp[2] ตามลำดับ

14.1.4 พอยน์เตอร์ของตัวแปรโครงสร้าง

เช่นกันกับอาร์เรย์ เราสามารถที่จะสร้างตัวแปรพอยน์เตอร์ของตัวแปรโครงสร้างได้ ซึ่งการใช้งานส่วนใหญ่จะเป็นการผ่านค่าตัวแปรเข้าไปในฟังก์ชัน



รูปที่ 14.9 การใช้งานพอยน์ของตัวแปรโครงสร้าง

จากรูปที่ 14.9 เป็นตัวอย่างการประกาศตัวแปรโครงสร้างแบบธรรมดาและแบบพอยน์เตอร์ ซึ่งจะสังเกตได้ว่าจะประกาศเหมือนการประกาศตัวแปรประเภทพอยน์เตอร์ปกติคือ

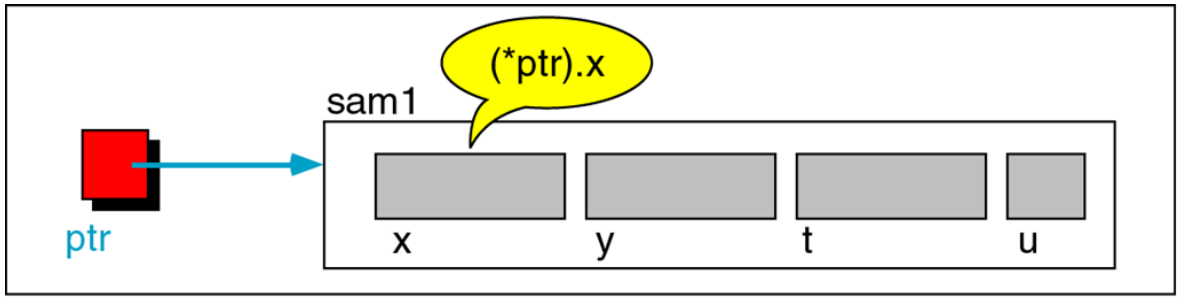
```
SAMPLE   *ptr;
```

เมื่อมีการใส่ค่าแอสเดรดให้กับตัวแปรพอยต์เตอร์จะส่งผลให้พอยน์เตอร์ชี้ไปยังตำแหน่งหน่วยความจำนั้น จากโปรแกรมจะเห็นได้ว่ามีการใส่ค่าแอสเดรดของตัวแปร sam1 ให้กับตัวแปรพอยน์เตอร์ ptr

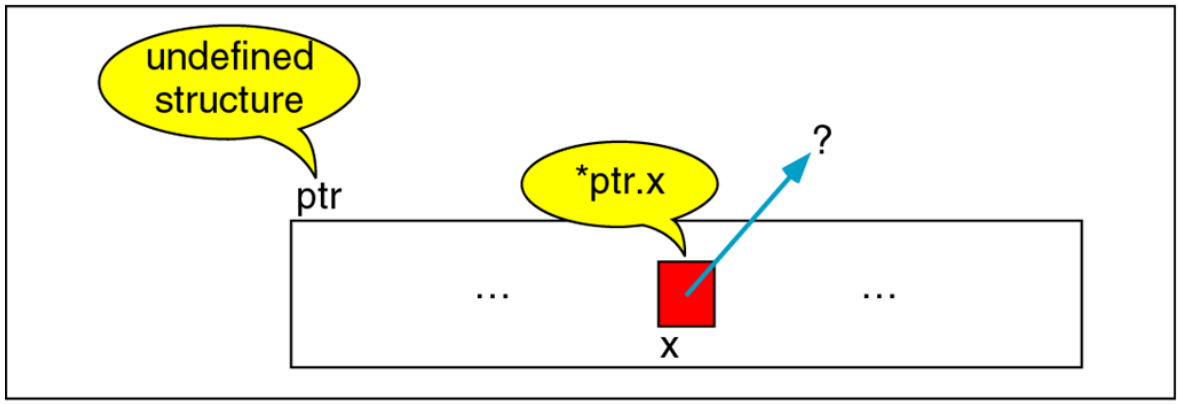
```
ptr = &sam1
```

ทำให้การอ้างอิงข้อมูลของสมาชิกตัวแปรโครงสร้างสามารถจะอ้างอิงได้ 2 วิธี คือการอ้างอิงผ่านตัวแปร sam1 และ การอ้างอิงผ่านตัวแปรพอยน์เตอร์ ptr ในการอ้างอิงสมาชิกผ่านตัวแปร sam1 สามารถอ้างอิงผ่าน sam1.x, sam1.y, sam1.t และ sam1.u แต่ถ้าต้องการอ้างอิงผ่านตัวแปรพอยน์เตอร์จะต้องอ้างอิงผ่าน (*ptr).x, (*ptr).y, (*ptr).t และ (*ptr).u

ข้อควรระวัง (*ptr).x จะไม่เท่ากับ *ptr.x ดังรูปที่ 14.10 เนื่องจาก (*ptr) หมายถึงตำแหน่งที่ ptr ชี้อยู่ ต่อด้วย .x คือไปหาสมาชิก x แต่ถ้าใช้ *ptr.x จะเหมือนกับการใช้ *(ptr.x) ซึ่งหมายถึงการอ้างอิงถึงค่าที่ ptr.x ชี้อยู่ซึ่งจะทำให้ค่าที่ได้ผิดพลาดจากค่าที่ต้องการ

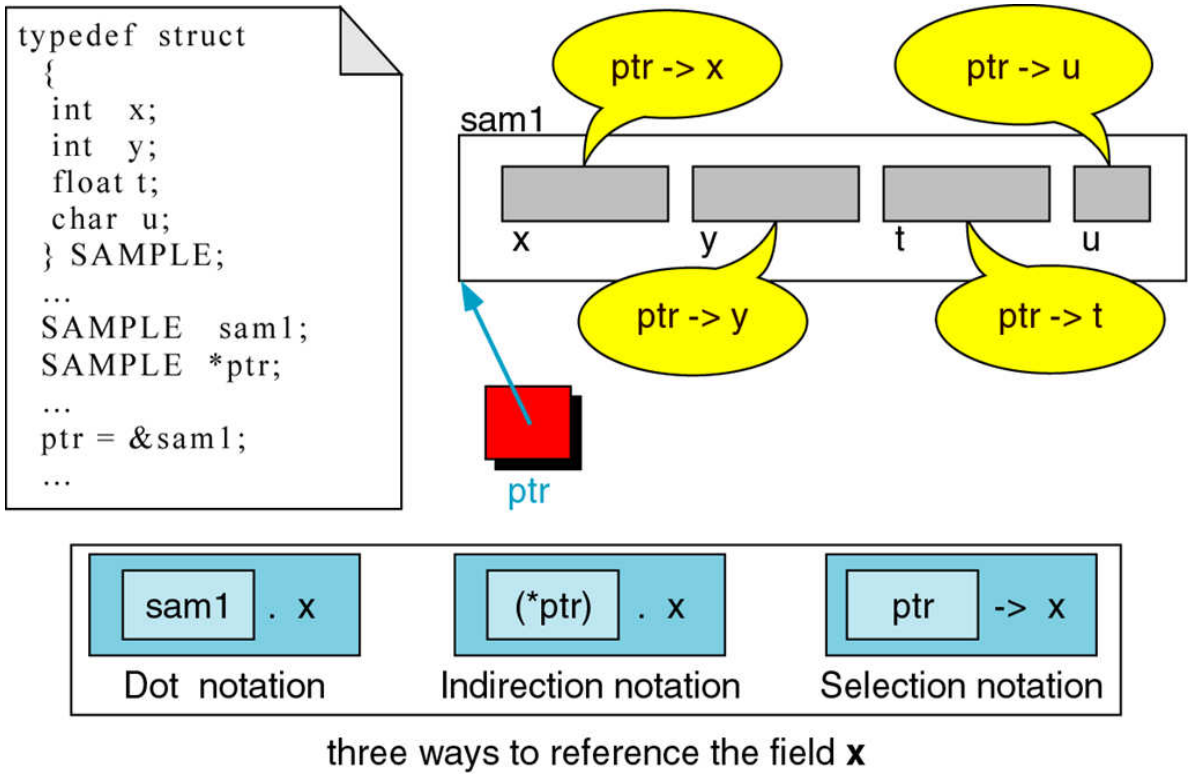


The correct reference



The wrong way to reference the component

รูปที่ 14.10 การอ้างพอยน์เตอร์ของตัวแปรประเภทโครงสร้าง



รูปที่ 14.11 การอ้างอิงค่าสมาชิกของพอยน์เตอร์ของตัวแปรประเภทโครงสร้าง

จากรูปที่ 14.11 แสดงวิธีการอ้างอิงค่าสมาชิกของตัวแปรประเภทโครงสร้าง ซึ่งสามารถทำได้ทั้งหมด 3 วิธี

- Dot notation เป็นการอ้างอิงสมาชิกภายในตัวแปรโครงสร้างที่ไม่ใช่พอยน์เตอร์โดยจะใช้ “.” ต่อด้วยชื่อตัวแปรสมาชิกภายใน จากตัวอย่าง sam1.x
- Indirection notation เป็นการอ้างอิงสมาชิกภายในตัวแปรโครงสร้างที่เป็นพอยน์เตอร์ โดยชี้ไปยังตำแหน่งที่ตัวแปรโครงสร้างนั้นอยู่ ซึ่งก็คือ (*ptr) ตามตัวอย่าง และตามด้วย “.” ต่อด้วยชื่อตัวแปรสมาชิกภายใน จากตัวอย่าง (*ptr).x
- Selection notation เป็นการอ้างอิงซึ่งให้ค่าเหมือนกับการใช้วิธี indirection notation แต่วิธีการเขียนจะสะดวกกว่าในการเขียนโปรแกรม โดยการเอาวงเล็บ และ * ออกจากการเขียนแบบ indirection notation และใช้เครื่องหมาย “->” แทน “.” จากตัวอย่าง ptr -> x

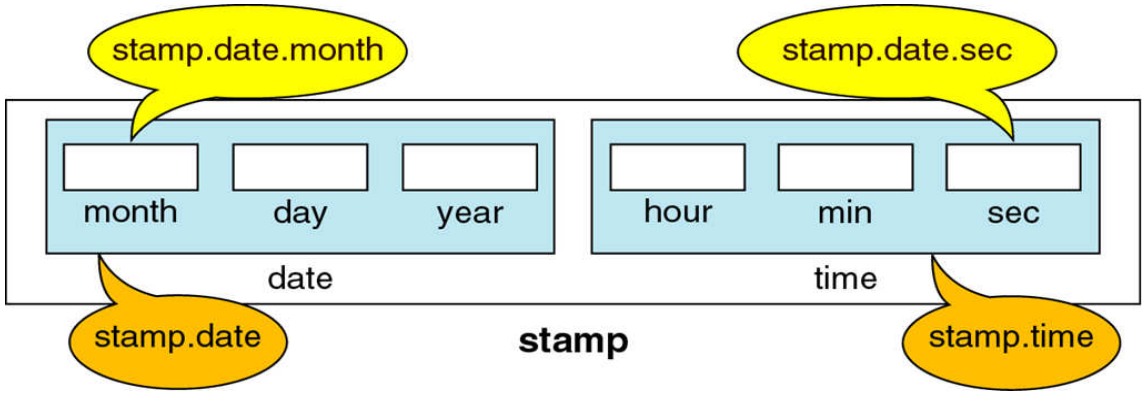
14.1.5 ตัวแปรโครงสร้างในตัวแปรโครงสร้าง

ในภาษาซีนอกจากตัวแปรโครงสร้างแบบพื้นฐาน แบบอาร์เรย์ และแบบพอยน์เตอร์แล้วนั้น เรายังสามารถจะสร้างตัวแปรโครงสร้างที่มีสมาชิกภายในเป็นตัวแปรโครงสร้างได้อีกด้วย ยกตัวอย่างเช่น

โครงสร้างข้อมูล DATE	โครงสร้างข้อมูล TIME	โครงสร้างข้อมูล STAMP
typedef struct { int month; int day; int year; } DATE;	typedef struct { int hour; int min; int sec; } TIME;	typedef struct { DATE date; TIME time; } STAMP; STAMP stamp;

- โครงสร้างข้อมูล DATE จะประกอบด้วยสมาชิกภายในคือ month, day, และ year ที่เป็นตัวแปรประเภทจำนวนเต็ม
- โครงสร้างข้อมูล TIME จะประกอบด้วยสมาชิกภายในคือ hour, min, และ sec ที่เป็นตัวแปรประเภทจำนวนเต็ม
- โครงสร้างข้อมูล STAMP ซึ่งจะประกอบด้วยสมาชิกภายในคือ date และ time ที่เป็นตัวแปรโครงสร้าง DATE และ ตัวแปรโครงสร้าง TIME ตามลำดับ

การอ้างอิงสมาชิกของตัวแปรโครงสร้างภายในตัวแปรโครงสร้างสามารถทำได้แบบไม่ยุ่งยากมากโดยการเรียกเป็นลำดับชั้น ตัวอย่างในรูปที่ 14.12 ยกตัวอย่างเช่น ต้องการอ้างอิงถึงสมาชิกชื่อ year ที่อยู่ในตัวแปรโครงสร้าง date ที่อยู่ที่ภายในตัวแปรโครงสร้าง stamp อีกทีสามารถอ้างอิงได้ คือ stamp.date.year



รูปที่ 14.12 การอ้างอิงสมาชิกภายในตัวแปรโครงสร้างในตัวแปรโครงสร้าง

14.1.6 ตัวแปรโครงสร้างกับฟังก์ชัน

ตัวแปรชนิดโครงสร้างสามารถนำไปใช้งานร่วมกับฟังก์ชันได้ ไม่ว่าจะเป็นการผ่านตัวแปรโครงสร้างทั้งหมด หรือ เพียงแค่สมาชิกภายในตัวแปรโครงสร้าง

ตัวอย่างโปรแกรมที่ 14.4

```
1  #include <stdio.h>
2  #include <string.h>
3
4  struct num_string {
5          int i;
6          char st[10];
7  };
8
9  void output(struct  num_string  a) {
10         printf("x.i = %d\n",   a.i);
11         printf("x.st = %s\n", a.st);
12 }
13
14 int  main(int  argc,  char  **argv) {
15         struct    num_string   x;
16         x.i  = 100;
17         strcpy(x.st, "COMPUTER");
18         output(x);
19 }
```

ผลการรัน

```
x.i = 100
x.st = COMPUTER
```

จากตัวอย่างโปรแกรมที่ 14.4 เป็นการสร้างฟังก์ชันที่รับตัวแปรโครงสร้างเข้ามาเพื่อแสดงค่าที่ภายในออกมาที่หน้าจอ

- บรรทัดที่ 4 - 7 เป็นการประกาศโครงสร้างข้อมูลชื่อ struct num_string โดยมีสมาชิกภายในคือ i และ st ซึ่งเป็นตัวแปรประเภทจำนวนเต็ม และข้อความตามลำดับ
- บรรทัดที่ 9 - 12 เป็นการสร้างฟังก์ชันชื่อ output ซึ่งรับตัวแปรโครงสร้างเข้ามา และใช้ฟังก์ชัน printf เพื่อแสดงค่าสมาชิกภายในตัวแปรโครงสร้างออกทางหน้าจอ
- บรรทัดที่ 14 - 19 เป็นฟังก์ชัน main โดย
  - บรรทัดที่ 15 ประกาศตัวแปรโครงสร้างชื่อ x โดยมีโครงสร้างแบบ struct num_string
  - บรรทัดที่ 16 กำหนดค่า 100 ให้กับสมาชิก i ของตัวแปรโครงสร้าง x
  - บรรทัดที่ 17 กำหนดค่า “COMPUTER” ให้กับสมาชิก st ของตัวแปรโครงสร้าง x
  - บรรทัดที่ 18 เรียกฟังก์ชัน output เพื่อแสดงค่าสมาชิกภายในตัวแปร x ออกหน้าจอ

การผ่านตัวแปรโครงสร้างดังตัวอย่างโปรแกรมที่ 14.4 จะมีข้อจำกัดคือ ฟังก์ชันจะไม่สามารถเปลี่ยนค่าของสมาชิกภายในตัวแปรโครงสร้างได้ เนื่องจากตัวฟังก์ชันจะทำการจองเนื้อที่ของหน่วยความจำขึ้นมาใหม่แล้วทำการก๊อบปี้ค่าต่างๆ เข้าไป ทำให้ถึงแม้จะเปลี่ยนค่าของสมาชิกภายในตัวแปรโครงสร้างก็จะเป็นการเปลี่ยนค่าที่หน่วยความจำตำแหน่งใหม่ ไม่ใช่ตำแหน่งที่แท้จริงของตัวแปรโครงสร้าง ดังนั้นเมื่อออกจากฟังก์ชัน ค่าต่างๆ ที่ถูกแก้ไขนั้นจะสูญหายไป

ดังนั้นถ้าต้องการจะให้ฟังก์ชันสามารถแก้ไขค่าของสมาชิกภายในตัวแปรโครงสร้างได้นั้น จำเป็นต้องใช้พอยน์เตอร์เข้ามาช่วย โดยการส่งตัวแปรโครงสร้างเข้าฟังก์ชันที่ต้องการแก้ไขข้อมูลสมาชิกจะต้องส่งเป็นแอสเดรดเข้ามา และตัวฟังก์ชันเองจะต้องรับค่ามาเป็นพอยน์เตอร์ของตัวแปรโครงสร้าง ซึ่งดูได้จากการทำงานของโปรแกรมตัวอย่างที่ 14.5 ซึ่ง

- บรรทัดที่ 3 - 6 เป็นการประกาศโครงสร้างข้อมูล struct num_string
- บรรทัดที่ 8 - 11 เป็นการประกาศฟังก์ชัน output เพื่อแสดงค่าภายในสมาชิกออกทางหน้าจอ
- บรรทัดที่ 13 - 16 เป็นการประกาศฟังก์ชัน modify ซึ่งเป็นฟังก์ชันที่สามารถทำการแก้ไขค่าของตัวแปรโครงสร้างที่ผ่านเข้ามาในฟังก์ชัน
- บรรทัดที่ 18 - 24 เป็นการทำงานของฟังก์ชัน main โดยจะเรียกใช้ฟังก์ชัน modify และ output ตามลำดับ

ตัวอย่างโปรแกรมที่ 14.5

```
1  #include <stdio.h>
2  #include <string.h>
3  struct  num_string {
4          int i;
5          char st[10];
6  };
7
8  void  output(struct  num_string  a) {
9          printf("x.i = %d\n",   a.i);
10         printf("x.st = %s\n", a.st);
11 }
12
13 void  modify(struct num_string *a,  int  i,  char  *st) {
14         a->i = i;
15         strcpy(a->st,  st);
16 }
17
18 int  main(int  argc,  char  **argv) {
19         struct  num_string  x;
20         x.i = 100;
21         strcpy(x.st, "COMPUTER");
22         modify(&x, 50, “PC”);
23         output(x);
24 }
```

ผลการรัน

```
x.i = 50
x.st = PC
```

14.2 ยูเนียน

ยูเนียน (union) เป็นข้อมูลอีกชนิดหนึ่งในภาษาซี ซึ่งคล้ายกับโครงสร้างข้อมูลคือ มีโครงสร้างภายในประกอบด้วยตัวแปรชนิดพื้นฐานในภาษาซีเช่นกัน แต่ส่วนที่แตกต่างกันก็คือ สมาชิกของข้อมูลชนิดยูเนียนจะใช้พื้นที่ในหน่วยความจำร่วมกัน โดยเปลี่ยนกันใช้พื้นที่ในตำแหน่งนั้นคนละช่วงเวลา ซึ่งเป็นการประหยัดเนื้อที่ในหน่วยความจำ รูปแบบการประกาศตัวแปรยูเนียนจะคล้ายกับโครงสร้างข้อมูลดังแสดงต่อไปนี้

```
union   name {
        type    var_1;
        type    var_2;
        …
        type   var_n;
};
```

- union : คำสั่งเพื่อสร้างข้อมูลแบบยูเนียน
- name : ชื่อของโครงสร้างข้อมูลแบบยูเนียน
- type : ประเภทของข้อมูล
- var_1 - var_n : ชื่อของตัวแปรสมาชิก

ตัวอย่างการประกาศตัวแปรแบบ union ที่ชื่อ employee พร้อมทั้งประกาศตัวแปรโครงสร้างยูเนียนที่ชื่อ emp

```
union  employee {
    char   name[24];
    int      code;
    float   salary;
 } emp;
```

การอ้างอิงค่าของสมาชิกภายในของยูเนียนสามารถทำได้เหมือนกับของสตรัคเจอร์ทุกประการ คือใช้ตัวดำเนินการ “.” เพื่อเข้าหาสมาชิกภายใน เช่น emp.code เพื่ออ้างอิงถึงสมาชิก code ของตัวแปรยูเนียน emp

บทที่ 15
การติดต่อกับแฟ้มข้อมูล

จากบทที่ผ่านมาเมื่อต้องการนำข้อมูลมาประมวลผลร่วมกับโปรแกรม จะใช้วิธีการรับข้อมูลผ่านทางแป้นพิมพ์ และส่งผลการดำเนินการออกทางจอภาพ ซึ่งในทางปฏิบัติ สามารถอ่านข้อมูลที่เก็บอยู่ในแฟ้มข้อมูลเข้ามาใช้ในโปรแกรม หรือ ส่งผลจาการดำเนินการออกไปเขียนลงในแฟ้มข้อมูลได้ในบทนี้จะกล่าวถึงวิธีการเขียนโปรแกรมด้วยภาษาซี เพื่อติดต่อและทำงานกับแฟ้มข้อมูล

15.1 หลักการทำงานร่วมกับแฟ้มข้อมูล

การประมวลผลร่วมกับแฟ้มข้อมูลในภาษาซีนั้น ไม่ได้กระทำโดยตรงกับแฟ้มข้อมูล แต่จะกระทำผ่านบัฟเฟอร์ (Buffer) บัฟเฟอร์คือ พื้นที่ในหน่วยความจำหลักที่ตัวแปลภาษาซีกันไว้สำหรับการทำงานกับแฟ้มข้อมูล โดยแต่ละแฟ้มข้อมูลจะมีบัฟเฟอร์เฉพาะ และไม่ใช้ร่วมกับแฟ้มข้อมูลอื่นๆ นอกจากนี้ในการประมวลผลร่วมกับแฟ้มข้อมูล ยังมีส่วนประกอบที่สำคัญอีกส่วนหนึ่งนั่นคือ *ตัวชี้ตำแหน่งแฟ้มข้อมูล* (File pointer) โดยใช้เป็นตัวบอกตำแหน่งภายในแฟ้มข้อมูลว่าในขณะนั้นประมวลผลอยู่ ณ ตำแหน่งใด ทำให้สามารถกำหนดได้ว่าจะอ่านหรือเขียนข้อมูลโดยเริ่มต้นจากจุดใด และสิ้นสุดที่จุดใด เมื่อเราเขียนคำสั่งเปิดแฟ้มข้อมูลในภาษาซี ตัวชี้ตำแหน่งแฟ้มข้อมูล จะชี้ไปยังจุดเริ่มต้นของแฟ้มข้อมูลนั้นๆ

ชนิดของแฟ้มข้อมูลจะสามารถแบ่งออกเป็น 2 ประเภทคือ เท็กซ์ไฟล์ (Text file) และ ไบนารีไฟล์ (Binary file)

- **เท็กซ์ไฟล์ (Text file)** เป็นแฟ้มข้อมูลที่เก็บข้อมูลอยู่ในรูปแบบของรหัส ASCII นั่นก็คือการเก็บเป็นตัวอักษรธรรมดานั่นเอง ทำให้เราสามารถอ่านข้อมูลในแฟ้มข้อมูลประเภทนี้ได้รู้เรื่อง สำหรับเท็กซ์ไฟล์นั้นจะมีการเปลี่ยนรหัสขึ้นบรรทัดใหม่ \n ให้เป็น Carriage return หรือ Line feed ก่อนที่จะบันทึกลงแฟ้มข้อมูล เมื่ออ่านข้อมูลจากแฟ้มข้อมูลกลับมาก็จะเปลี่ยน Carriage return หรือ Line feed กลับมาเป็น \n ตัวอย่างของแฟ้มข้อมูลประเภทเท็กซ์ไฟล์ ได้แก่ แฟ้มข้อมูลที่มีนามสกุล .txt .c หรือ .bat เป็นต้น

- **ไบนารีไฟล์ (Binary file)** เป็นแฟ้มข้อมูลที่เก็บข้อมูลในรูปแบบของเลขฐานสอง ดังนั้นเราจึงไม่สามารถอ่านข้อมูลในแฟ้มข้อมูลประเภทนี้รู้เรื่อง แฟ้มข้อมูลประเภทนี้จะไม่มีการเปลี่ยนรหัสขึ้นบรรทัดใหม่ \n เป็น Carriage return หรือ Line feed ตัวอย่างของแฟ้มข้อมูลประเภทนี้ ได้แก่ แฟ้มข้อมูลที่มีนามสกุล .exe .mp3 หรือ .avi เป็นต้น

การจัดการกับแฟ้มข้อมูลในภาษาซีนั้น สามารถแบ่งออกเป็นกลุ่มลักษณะการทำงานที่สำคัญได้ใหญ่ๆ 4 กลุ่มการทำงานคือ

- เปิดแฟ้มข้อมูล
- ปิดแฟ้มข้อมูล
- อ่านแฟ้มข้อมูล
- เขียนแฟ้มข้อมูล

15.2 การเปิดแฟ้มข้อมูล

ในการประมวลผลร่วมกันกับแฟ้มข้อมูลนั้น การดำเนินการแรกที่จะต้องกระทำก่อนงานอื่นๆ คือ การเปิดแฟ้มข้อมูลเพื่อเลือกแฟ้มข้อมูลที่จะประมวลผลและสร้างตัวชี้ตำแหน่งแฟ้มข้อมูลขึ้นมา ซึ่งการเปิดแฟ้มข้อมูลนั้น จะต้องใช้ 2 คำสั่งด้วยกัน คำสั่งแรกเป็นการสร้างตัวแปรพอยน์เตอร์ชนิด FILE เพื่อใช้เป็นตัวชี้ตำแหน่งแฟ้มข้อมูล และ คำสั่งที่สองคือ คำสั่งเปิดแฟ้มข้อมูลซึ่งจะให้ค่าตัวชี้ตำแหน่งแฟ้มข้อมูลกับตัวแปรพอยน์เตอร์ที่สร้างขึ้นจากคำสั่งแรก ซึ่งมีการใช้งานดังนี้

```
FILE  *fp;
fp  = fopen("name",  "mode");
```

- FILE : ประเภทข้อมูลที่ใช้ทำงานเกี่ยวกับแฟ้มข้อมูล
- *fp : ชื่อตัวแปรประเภทพอยน์เตอร์ที่จะใช้เป็นตัวชี้ตำแหน่งแฟ้มข้อมูล
- fopen : เป็นคำสั่งที่ใช้เพื่อเปิดแฟ้มข้อมูล
- name : ชื่อแฟ้มข้อมูลที่ต้องการเปิด
- mode : โหมดของการเปิดแฟ้มข้อมูล

โหมดของการเปิดแฟ้มข้อมูล ถูกใช้เพื่อเป็นการระบุวัตถุประสงค์ที่เปิดแฟ้มข้อมูลนี้ ซึ่งในภาษาซีจะมีโหมดของการเปิดแฟ้มข้อมูล ตามตารางที่ 15.1

ตารางที่ 15.1 โหมดของการเปิดแฟ้มข้อมูล

โหมดของการเปิดแฟ้มข้อมูล	ความหมาย
r	เปิดแฟ้มข้อมูลเพื่ออ่านอย่างเดียว
w	เปิดแฟ้มข้อมูลเพื่อเขียนลงไปทับข้อมูลเก่าในแฟ้มข้อมูล
a	เปิดแฟ้มข้อมูลเพื่อเขียนข้อมูลต่อท้ายข้อมูลเก่าในแฟ้มข้อมูล
r+	เปิดแฟ้มข้อมูลเพื่ออ่านและเขียน
w+	เปิดแฟ้มข้อมูลเพื่ออ่านและเขียนทับข้อมูลเก่าในแฟ้มข้อมูล
a+	เปิดแฟ้มข้อมูลเพื่ออ่านและเขียนต่อท้ายข้อมูลเก่าในแฟ้มข้อมูล

การเปิดแฟ้มข้อมูลโดยใช้โหมด r นั้นจำเป็นจะต้องมีแฟ้มข้อมูลนั้นอยู่จริงในเครื่องคอมพิวเตอร์ จึงจะสามารถเปิดแฟ้มข้อมูลขึ้นมาอ่านได้ ในขณะที่โหมดของการเปิดแฟ้มข้อมูลอื่นๆ นั้นถ้าแฟ้มข้อมูลยังไม่มีอยู่จะเป็นการสร้างแฟ้มข้อมูลขึ้นมาใหม่ ในการใช้คำสั่งเปิดแฟ้มข้อมูลนั้นถ้าเปิดแฟ้มข้อมูลสำเร็จหรือไม่มีข้อผิดพลาดใดๆ เกิดขึ้น ค่าที่ได้กลับมาคือ ตัวชี้ตำแหน่งแฟ้มข้อมูล แต่ถ้าเปิดแฟ้มข้อมูลไม่สำเร็จหรือมีข้อผิดพลาดเกิดขึ้น ค่าที่ได้กลับมาจะเป็น NULL

ตัวอย่างโปรแกรมที่ 15.1

```
1  FILE  *fp;
2  fp  =  fopen("d:/exam/test.txt", "r");
3  fp  =  fopen("c:/tmp.mp3",  "w");
4  fp  =  fopen("ex22.exe", "a+");
5  fp  =  fopen("c:/documents/mydoc.txt",  "w+");
```

จากตัวอย่างโปรแกรมที่ 15.1 ซึ่งเป็นเพียงส่วนของโปรแกรม

- บรรทัดที่ 1 เป็นการสร้างตัวแปรพอนย์เตอร์ fp ที่เป็นประเภทข้อมูล FILE
- บรรทัดที่ 2 แสดงตัวอย่างของการเปิดแฟ้มข้อมูลชื่อ d:/exam/test.txt เพื่ออ่านข้อมูล
- บรรทัดที่ 3 แสดงตัวอย่างของการเปิดแฟ้มข้อมูลชื่อ c:/tmp.mp3 เพื่อเขียนข้อมูลใหม่
- บรรทัดที่ 4 แสดงตัวอย่างของการเปิดแฟ้มข้อมูลชื่อ ex22.exe เพื่ออ่านและเขียนข้อมูลต่อท้ายแฟ้มข้อมูล
- บรรทัดที่ 5 แสดงตัวอย่างของการเปิดแฟ้มข้อมูลชื่อ c:/documents/mydoc.txt เพื่ออ่านและเขียนข้อมูลทับแฟ้มข้อมูลเก่า

15.3 การปิดแฟ้มข้อมูล

เมื่อทำงานร่วมกับแฟ้มข้อมูลเสร็จสิ้นแล้ว ขั้นตอนสุดท้ายควรจะทำการปิดแฟ้มข้อมูลให้เรียบร้อย เพื่อคืนเนื้อที่ของส่วนที่ใช้เป็นบัฟเฟอร์ของแฟ้มข้อมูลในเครื่อง นอกจากนั้นเมื่อปิดแฟ้มข้อมูล ข้อมูลต่างๆ ที่ยังค้างอยู่ในบัฟเฟอร์จะถูกเขียนกลับลงไปในแฟ้มข้อมูลด้วย รูปแบบของคำสั่งที่ใช้ในการปิดแฟ้มข้อมูลสามารถใช้งานได้ดังนี้

```
fclose( fp );
```

- fclose : เป็นคำสั่งที่ใช้เพื่อปิดแฟ้มข้อมูล
- fp : ตัวชี้ตำแหน่งแฟ้มข้อมูลที่ได้จากการเปิดแฟ้มข้อมูล

เมื่อเรียกใช้ fclose( ) เพื่อปิดแฟ้มข้อมูลแล้ว ถ้าสามารถปิดแฟ้มข้อมูลได้อย่างเรียบร้อยและไม่มีข้อผิดพลาดใดๆ จะได้ค่าศูนย์คืนกลับมา แต่ถ้าการปิดแฟ้มข้อมูลมีปัญหาหรือมเกิดข้อผิดพลาดขึ้น ค่าที่ได้คืนกลับมาจากฟังก์ชัน fclose( ) จะมีค่าที่ไม่ใช่ศูนย์

ตัวอย่างโปรแกรมที่ 15.2

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char **argv) {
4      FILE   *fp;
5      fp = fopen(“d:/test.txt”, “r”);
6      if( fp == NULL ) {
7              printf("File not found\n”);
8              exit(0);
9      }
10     printf(“Open file successfully\n”);
11     if( !fclose(fp) ) {
12             printf(“Close file\n”);
13     }
14 }
```

ตัวอย่างโปรแกรมที่ 15.2 เป็นตัวอย่างของการเขียนโปรแกรมภาษาซีให้สามารถทำงานร่วมกับแฟ้มข้อมูล

- บรรทัดที่ 4 และ 5 เป็นชุดคำสั่งเพื่อใช้ในการเปิดแฟ้มข้อมูลโดยโปรแกรมต้องการจะเปิดแฟ้มข้อมูลชื่อ test.txt ที่อยู่ใน d: โดยเป็นการเปิดแฟ้มข้อมูลเพื่ออ่าน
- บรรทัดที่ 6 - 9 เป็นการตรวจสอบตัวชี้ตำแหน่งแฟ้มข้อมูล ถ้าเป็น NULL แสดงว่าการเปิดแฟ้มข้อมูลมีปัญหา ซึ่งโปรแกรมจะแสดงคำว่า “File not found” ออกทางหน้าจอและออกจากโปรแกรม
- บรรทัดที่ 11 ใช้คำสั่งเพื่อปิดแฟ้มข้อมูลถ้าการปิดแฟ้มข้อมูลเรียบร้อยไม่มีข้อผิดพลาดจะแสดงคำว่า “Close file” ออกทางหน้าจอ

ผลที่เกิดจากการรันของโปรแกรมตัวอย่างที่ 15.2 ในสภาวะเงื่อนไขต่างๆ แสดงในรูปที่ 15.1

ผลการรัน : ถ้าแฟ้มข้อมูล d:/test.txt ไม่มีในเครื่อง File not found
ผลการรัน : ถ้ามีแฟ้มข้อมูล d:/test.txt ในเครื่องและไม่มีปัญหาในการเปิด แต่ มีปัญหาในขั้นตอนการปิดแฟ้มข้อมูล Open file successfully
ผลการรัน : ถ้ามีแฟ้มข้อมูล d:/test.txt ในเครื่องและไม่มีปัญหาในการเปิด และ ไม่มีปัญหาในขั้นตอนการปิดแฟ้มข้อมูล Open file successfully Close file

รูปที่ 15.1 ผลการรันของโปรแกรมตัวอย่างที่ 15.2 ในสภาวะเงื่อนไขต่างๆ

15.4 การอ่านแฟ้มข้อมูล

การอ่านแฟ้มข้อมูลในภาษาซีนั้น โดยปกติจะเป็นการอ่านค่าที่ตัวชี้ตำแหน่งแฟ้มข้อมูลชี้อยู่ ซึ่งหลังจากทำการเปิดแฟ้มข้อมูลตัวชี้ตำแหน่งแฟ้มข้อมูลจะชี้ข้อมูลตัวแรกสุดที่อยู่ในแฟ้มข้อมูลนั้น การอ่านจะเป็นการเลื่อนตัวชี้ตำแหน่งแฟ้มข้อมูลไปเรื่อยๆ จนกระทั่งสิ้นสุดแฟ้มข้อมูล ข้อมูลที่อ่านจากแฟ้มข้อมูลนั้นจะถูกเก็บอยู่ในบัฟเฟอร์ เพื่อให้โปรแกรมภาษาซีสามารถอ่านค่าได้ โดยจะเห็นได้ว่าในภาษาซีนั้น การอ่านแฟ้มข้อมูลไม่ได้กระทำตรงๆ กับแฟ้มข้อมูล แต่เป็นการกระทำผ่านบัฟเฟอร์ การอ่านแฟ้มข้อมูลด้วยภาษาซีจะมีฟังก์ชันที่นิยมใช้กัน ได้แก่ fgetc( ) , fgets( ), และ fscanf( )

15.4.1 ฟังก์ชัน fgetc( )

ฟังก์ชั่น fgetc( ) ใช้สำหรับการอ่านข้อมูลจากแฟ้มข้อมูลออกมาทีละหนึ่งตัวอักขระ โดยการใช้งานคำสั่ง fgetc( ) มีลักษณะดังนี้

```
ch = fgetc( fp );
```

- ch : ตัวแปรประเภทตัวอักขระที่ใช้เก็บค่าข้อมูลที่อ่านจากแฟ้มข้อมูล
- fgetc : ฟังก์ชัน fgetc สำหรับอ่านข้อมูลทีละหนึ่งอักขระ
- fp : ตัวชี้ตำแหน่งแฟ้มข้อมูลที่ต้องการจะอ่านข้อมูล

เนื่องด้วยการใช้ฟังก์ชัน fgetc( ) นั้น สามารถอ่านข้อมูลได้เพียงทีละหนึ่งตัวอักขระ โดยปกติจะมีการใช้งานควบคู่ไปกับคำสั่งประเภทลูป เช่น while หรือ for เพื่อที่จะวนอ่านตัวอักขระไปเรื่อยๆ ตั้งแต่ตัวแรกสุดจนถึงตัวสุดท้ายของแฟ้มข้อมูล

เมื่อมีการใช้คำสั่งประเภทลูปควบคู่ไปกับการอ่านข้อมูลจากแฟ้มข้อมูล จึงมีความจำเป็นต้องทราบจุดสิ้นสุดของการวนลูป ในที่นี้คือจำเป็นต้องทราบว่าได้อ่านแฟ้มข้อมูลจนจบแล้ว ซึ่งการตรวจสอบจุดสิ้นสุดของแฟ้มข้อมูลนั้นสามารถทำได้ 2 วิธี ขึ้นอยู่กับชนิดของแฟ้มข้อมูล

- ในกรณีของเท็กไฟล์อ่านข้อมูลจนถึงจุดสิ้นสุดของแฟ้มข้อมูลแล้ว ค่าที่คืนกลับมาจากฟังก์ชัน fgetc( ) จะได้ค่า EOF ซึ่งสามารถนำมาเปรียบเทียบ ในเงื่อนไข เพื่อออกจากการวนซ้ำได้
- ในกรณีของไบนารีไฟล์ สามารถใช้ฟังก์ชัน feof( ) เพื่อตรวจสอบจุดสิ้นสุดของแฟ้มข้อมูลถ้าตัวชี้ตำแหน่งแฟ้มข้อมูลยังไม่ใช่จุดสิ้นสุดของแฟ้มข้อมูล ฟังก์ชันนี้จะคืนค่า 0 หรือ เท็จ แต่ถ้าถึงจุดสิ้นสุดของแฟ้มข้อมูลแล้ว ฟังก์ชี้จะคืนค่าที่ไม่ใช่ 0 หรือ ค่าความจริงเป็นจริง ลักษณะการใช้งานของฟังก์ชัน feof() มีลักษณะดังนี้

feof(fp);
• feof : เป็นฟังก์ชันเพื่อตรวจสอบจุดสิ้นสุดของแฟ้มข้อมูล • fp : ตัวชี้ตำแหน่งแฟ้มข้อมูลที่ต้องการตรวจสอบ

ตัวอย่างโปรแกรมที่ 15.3

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char **argv) {
4      FILE   *fp;
5      fp = fopen("d:/test.txt", "r");
6      if( fp == NULL ) {
7              printf("File not found\n");
8              exit(0);
9      }
10     while((ch = fgetc(fp)) != EOF) {
11           printf("%c",  ch);
12     }
13     fclose(fp);
14 }
```

ตัวอย่างโปรแกรมที่ 15.3 เป็นโปรแกรมที่ใช้ในการอ่านข้อมูลประเภทเท็กซ์ไฟล์ซึ่งมีลักษณะการทำงานดังนี้

- บรรทัดที่ 4-5 เปิดแฟ้มข้อมูลชื่อ d:/test.txt เป็นแฟ้มข้อมูลประเภทเท็กซ์ และเป็นการเปิดแฟ้มข้อมูลในโหมดอ่านเพียงอย่างเดียว
- บรรทัดที่ 6-9 เป็นการตรวจสอบว่าสามารถเปิดแฟ้มข้อมูลได้สำเร็จหรือไม่ ถ้าไม่สำเร็จจะแสดงคำว่า "File not found" ในบรรทัดที่ 7 ออกหน้าจอ จากนั้นก็ปิดโปรแกรมด้วยคำสั่งในบรรทัดที่ 8
- บรรทัดที่ 10-14 เป็นการเรียกใช้ฟังก์ชัน fgetc เพื่ออ่านข้อมูลจากแฟ้มข้อมูลขึ้นมา 1 ตัวอักขระแล้วเก็บไว้ที่ตัวแปร ch พร้อมทั้งตรวจสอบว่าเป็นจุดสิ้นสุดของแฟ้มข้อมูลหรือยัง ถ้าสิ้นสุดแล้วก็ออกจากลูปแล้วทำการปิดแฟ้มข้อมูล ไม่เช่นนั้นก็แสดงค่าที่อ่านจากแฟ้มข้อมูลได้ออกจากภาพแล้วไปรับข้อมูลตัวถัดไป

ตัวอย่างโปรแกรมที่ 15.4

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char **argv) {
4      FILE   *fp;
5      fp = fopen("binary.exe", "r");
6      if( fp == NULL ) {
7              printf("File not found\n");
8              exit(0);
9      }
10     while( !feof(fp) ) {
11             ch = fgetc(fp);
12             printf("%c",  ch);
13     }
14     fclose(fp);
15 }
```

ตัวอย่างโปรแกรมที่ 15.4 เป็นโปรแกรมที่ใช้ในการอ่านข้อมูลประเภทไบนารีไฟล์ซึ่งมีลักษณะการทำงานดังนี้

- บรรทัดที่ 4-5 เปิดแฟ้มข้อมูลชื่อ binary.exe เป็นแฟ้มข้อมูลประเภทไบนารี และเป็นการเปิดแฟ้มข้อมูลในโหมดอ่านเพียงอย่างเดียว
- บรรทัดที่ 6-9 เป็นการตรวจสอบว่าสามารถเปิดแฟ้มข้อมูลได้สำเร็จหรือไม่ ถ้าไม่สำเร็จจะแสดงคำว่า "File not found" ในบรรทัดที่ 7 ออกทางหน้าจอ จากนั้นปิดโปรแกรมด้วยคำสั่งในบรรทัดที่ 8
- บรรทัดที่ 10-15 เป็นการเรียกใช้ฟังก์ชัน feof( ) ขึ้นมาเพื่อตรวจสอบจุดสิ้นสุดของแฟ้มข้อมูลถ้าตัวชี้ตำแหน่งแฟ้มข้อมูล(fp) ยังไม่ชี้ไปยังตำแหน่งจุดสิ้นสุดของแฟ้มข้อมูล โปรแกรมจะทำการอ่านข้อมูลด้วยฟังก์ชัน fgetc( ) เพื่ออ่านข้อมูลจากแฟ้มข้อมูลขึ้นมา 1ตัวอักขระแล้วเก็บไว้ที่ตัวแปร ch แล้วแสดงค่าที่อ่านจากแฟ้มข้อมูลได้ออกทางจอภาพ แล้วไปตรวจสอบตำแหน่งจุดสิ้นสุดของแฟ้มข้อมูลอีกครั้ง แต่ถ้าถึงจุดสิ้นสุดของแฟ้มข้อมูลแล้วโปรแกรมจะหลุดออกจากลูปแล้วไปปิดแฟ้มข้อมูลในบรรทัดที่ 14

15.4.2 ฟังก์ชัน fgets( )

ฟังก์ชัน fgets( ) เป็นฟังก์ชันที่ใช้ในการอ่านข้อมูลจากแฟ้มข้อมูลต่อเนื่องกันเป็นข้อความ โดยผู้พัฒนาโปรแกรมจะเป็นผู้กำหนดขนาดความยาวของข้อความที่จะอ่านเอง รูปแบบการเรียกใช้ฟังก์ชัน fgets( ) แสดงดังต่อไปนี้

```
fgets( str,  size,  fp );
```

- fgets : เป็นฟังก์ชันเพื่ออ่านข้อมูลแบบข้อความ
- str : ตัวแปรประเภทสตริงก์ ใช้เก็บข้อความที่อ่านจากแฟ้มข้อมูล
- size : ความยาวของข้อความที่ต้องการอ่านจากแฟ้มข้อมูล
- fp : ตัวชี้ตำแหน่งแฟ้มข้อมูลที่ต้องการอ่าน

ตัวอย่างโปรแกรมที่ 15.5

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char **argv) {
4      FILE   *fp;
5      char  str[60];
6      fp = fopen("c:\mytext.txt", "r");
7      if( fp == NULL ) {
8              printf("File not found\n");
9              exit(0);
10     }
11     fgets(str,  50,  fp);
12     printf("%s",  str);
13    fclose(fp);
14 }
```

ตัวอย่างโปรแกรมที่ 15.5 แสดงการใช้งานฟังก์ชัน fgets( ) ในบรรทัดที่ 5 จะเป็นการ ประกาศตัวแปรประเภทสตริงก์ str ส่วนในบรรทัดที่ 11 จะเป็นการอ่านข้อมูลจากแฟ้มข้อมูล c:\mytext.txt ขึ้นมา 50 ตัวอักษร แล้วเก็บไว้ในตัวแปร str จากนั้นนำค่าที่อ่านได้แสดงผลบนจอภาพด้วยฟังก์ชัน printf( ) ในบรรทัด 12

ตัวอย่างโปรแกรมที่ 15.6

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char **argv) {
4      FILE   *fp;
5      char  str[60];
6      fp = fopen("c:\mytext.txt", "r");
7      if( fp == NULL ) {
8              printf("File not found\n");
9              exit(0);
10     }
11     while(!feof(fp)) {
12             fgets(str,  50,  fp);
13             printf("%s",  str);
14     }
15     fclose(fp);
16 }
```

ตัวอย่างโปรแกรมที่ 15.6 แสดงการใช้งานฟังก์ชัน fgets( ) เพื่ออ่านแฟ้มข้อมูลและแสดงข้อความที่อยู่ในแฟ้มข้อมูลออกทางจอภาพ ทั้งแฟ้มข้อมูล ซึ่งมีรายละเอียดการทำงานดังนี้

- บรรทัดที่ 4-6 เปิดแฟ้มข้อมูลชื่อ c:\mytext.txt
- บรรทัดที่ 7-10 เป็นการตรวจสอบว่าสามารถเปิดแฟ้มข้อมูลได้สำเร็จหรือไม่ ถ้าไม่สำเร็จจะแสดงคำว่า "File not found"
- บรรทัดที่ 11 เป็นการตรวจสอบว่าตัวชี้ตำแหน่งแฟ้มข้อมูลถึงจุดสิ้นสุดของแฟ้มข้อมูลแล้วหรือยัง ถ้าสิ้นสุดแล้วก็จะหยุดการวนลูป ข้ามไปทำงานบรรทัดที่ 15 ซึ่งเป็นการปิดแฟ้มข้อมูล ถ้าตัวชี้ตำแหน่งแฟ้มข้อมูลยังไม่ถึงจุดสิ้นสุดของแฟ้มข้อมูล โปรแกรมจะทำงานต่อในบรรทัดที่ 12 และ 13
- บรรทัดที่ 12-13 เป็นการเรียกใช้ฟังก์ชัน fgets( ) เพื่ออ่านค่าจากแฟ้มข้อมูลทีละ 50 ตัวอักษร นำไปเก็บไว้ในตัวแปร str จากนั้นแสดงค่าที่อ่านได้ออกทางหน้าจอด้วยฟังก์ชัน printf( ) ในบรรทัดที่ 13

15.4.3 ฟังก์ชัน fscanf( )

การอ่านข้อมูลจากแฟ้มข้อมูลนอกจากการอ่านเข้ามาเป็นตัวอักขระ หรือ ข้อความแล้วนั้น ยังสามารถกำหนดรูปแบบการอ่านข้อมูลจากแฟ้มข้อมูลเข้ามาเป็นข้อมูลชนิดอื่นๆ ได้อีก เช่น จำนวนเต็ม หรือ จำนวนจริง ซึ่งทำให้สะดวกมากขึ้นในการทำงานร่วมกับแฟ้มข้อมูลบางประเภทที่เก็บค่าตัวเลข เช่น แฟ้มข้อมูลบัญชี หรือ แฟ้มข้อมูลสถิติ เป็นต้น รูปแบบการเรียกใช้งานคำสั่ง fscanf() มีลักษณะดังนี้

```
fscanf( fp,  “format”,  &var );
```

- fp : ตัวชี้ตำแหน่งแฟ้มข้อมูลที่ต้องการอ่าน
- format : รหัสควบคุมรูปแบบ เหมือนกับการใช้งาน scanf( ) เช่น %d, %f
- &var : Address ของตัวแปรซึ่งจะใช้เก็บข้อมูลที่อ่านเข้ามา โดยประเภทของตัวแปรจะต้องสอดคล้องกับ format ที่กำหนด

ตัวอย่างโปรแกรมที่ 15.7

```
1  fscanf(fp,  “%d”,  &salary);
2  fscanf(fp,  “%d %d”,  &day,  &month);
3  fscanf(fp,  “%c  %d”,  &ch,  &num);
4  fscanf(fp,  “%f  %d”,  &score,  &grade);
5  fscanf(fp,  “%s”,  str);
```

จากตัวอย่างโปรแกรมที่ 15.7 ซึ่งเป็นเพียงส่วนหนึ่งของโปรแกรมในการอธิบายลักษณะการใช้งานของฟังก์ชัน fscanf( )

- บรรทัดที่ 1 อ่านข้อมูลจากแฟ้มข้อมูลเป็นจำนวนเต็มมาเก็บไว้ที่ตัวแปร salary
- บรรทัดที่ 2 อ่านข้อมูลจากแฟ้มข้อมูลแบบจำนวนเต็ม 2 ค่า ค่าแรกเก็บไว้ในตัวแปร day และค่าที่สองเก็บไว้ในตัวแปร month
- บรรทัดที่ 3 อ่านข้อมูลจากแฟ้มข้อมูลขึ้นมา 2 ค่า ค่าแรกเป็นแบบตัวอักขระซึ่งจะเก็บไว้ในตัวแปร ch ค่าที่สองเป็นจำนวนเต็มเก็บไว้ในตัวแปร num
- บรรทัดที่ 4 อ่านข้อมูลจากแฟ้มข้อมูลขึ้นมา 2 ค่า ค่าแรกเป็นแบบจำนวนจริงซึ่งจะเก็บไว้ในตัวแปร score ค่าที่สองเป็นจำนวนเต็มเก็บไว้ในตัวแปร grade
- บรรทัดที่ 5 อ่านข้อมูลจากแฟ้มข้อมูลเป็นข้อความแล้วนำมาเก็บไว้ที่ตัวแปร str

ตัวอย่างโปรแกรมที่ 15.8

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char **argv) {
4      FILE   *fp;
5      int  day,  month,  year;
6      fp = fopen(“birthday.txt”, “r”);
7      if( fp == NULL ) {
8              printf("File not found\n”);
9              exit(0);
10     }
11
12     fscanf(fp, “%d  %d  %d”, &day, &month, &year);
13     printf(“day : %d\n”, &day);
14     printf(“month : %d\n”,  &month);
15     printf(“year : %d\n”, year);
16
17     fclose(fp);
18 }
```

กำหนดให้แฟ้มข้อมูล birthday.txt มีข้อมูลดังรูปที่ 15.2

birthday.txt
16 3 2522
2 7 2525
28 6 2528

รูปที่ 15.2 ข้อมูลในแฟ้มข้อมูล birthday.txt

จากตัวอย่างโปรแกรมที่ 15.8 แสดงการทำงานของฟังก์ชัน fscanf( ) โดยเป็นการเปิดแฟ้มข้อมูลที่ชื่อ birthday.txt เพื่ออ่านค่าตัวเลขที่เก็บอยู่ในแฟ้มข้อมูล

- บรรทัดที่ 3-5 เป็นการเปิดแฟ้มข้อมูล
- บรรทัดที่ 7-10 เป็นการตรวจสอบความถูกต้องของการเปิดแฟ้มข้อมูล
- บรรทัดที่ 12 เป็นการเรียกใช้ฟังก์ชัน fscanf( ) เพื่ออ่านค่าจำนวนเต็ม 3 ค่า ค่าแรกจะเก็บไว้ในตัวแปรชื่อ day ค่าที่สองเก็บไว้ในตัวแปรชื่อ month และค่าสุดท้ายจะถูกเก็บไว้ในตัวแปรชื่อ year
- บรรทัดที่ 13-15 จะเป็นการแสดงค่าของตัวแปร day, month,และ year ตามลำดับออกทางหน้าจอ
- บรรทัดที่ 17 จะเป็นการปิดแฟ้มข้อมูล

กำหนดในข้อมูลในแฟ้มข้อมูล birthday.txt มีค่าดังรูปที่ 15.2 ผลจากการรันตัวอย่างโปรแกรมที่ 15.8 จะมีค่าดังนี้

```
day : 16
month : 3
year : 2522
```

จะเห็นได้ว่าข้อมูลที่ถูกแสดงผลจากโปรแกรมจะมีเพียงแค่ชุดเดียวเนื่องจากมีการเรียกใช้งานฟังก์ชั่น fscanf( ) เพียงครั้งเดียว ซึ่งส่งผลให้โปรแกรมอ่านค่าที่มีอยู่ที่บรรทัดแรกสุดของแฟ้มข้อมูล birthday.txt แล้วนำมาแสดงผล

15.5 การเขียนแฟ้มข้อมูล

สำหรับการเขียนข้อมูลลงในแฟ้มข้อมูลนั้น ขั้นแรกจะต้องเปิดแฟ้มข้อมูลในโหมดที่สามารถเขียนข้อมูลลงไปในแฟ้มข้อมูลได้ก่อน อันได้แก่ w, w+, a, a+ หรือ r+ จากนั้นจึงจะสามารถเรียกใช้งานคำสั่งสำหรับเขียนข้อมูลลงไปในแฟ้มข้อมูลได้ โดยข้อมูลที่จะถูกเขียนลงในแฟ้มข้อมูลนั้นจะถูก เขียนลงไปในบัฟเฟอร์ก่อน ในระหว่างที่เขียนข้อมูลตัวชี้ตำแหน่งแฟ้มข้อมูลจะเลื่อนตำแหน่งชี้ไปเรื่อยๆ ตามปริมาณของข้อมูลที่เขียน เมื่อบัฟเฟอร์เต็มหรือมีการสั่งปิดแฟ้มข้อมูล ข้อมูลที่อยู่ในบัฟเฟอร์ถึงจะเขียนลงไปในแฟ้มข้อมูลจริงๆ ฟังก์ชันที่นิยมใช้ในการเขียนแฟ้มข้อมูลได้แก่ fputc() , fputs( ), และ fprintf( )

15.5.1 ฟังก์ชัน fputc( )

ฟังก์ชั่น fputc( ) ใช้สำหรับการเขียนข้อมูลลงไปในแฟ้มข้อมูลทีละหนึ่งตัวอักขระ โดยการใช้งานคำสั่ง fputc( ) มีลักษณะดังนี้

fputc(ch, fp);
• ch : ตัวแปรประเภทตัวอักขระที่ต้องการจะเขียนลงไปในแฟ้มข้อมูล • fp : ตัวชี้ตำแหน่งแฟ้มข้อมูลที่ต้องการจะเขียนข้อมูล

ตัวอย่างโปรแกรมที่ 15.9

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char **argv) {
4      FILE   *fp;
5      char letter;
6      fp = fopen("alphabet.txt", "w");
7      if( fp == NULL ) {
8              printf("Can't open file\n");
9              exit(0);
10     }
11     for(letter = 'A';  letter <= 'Z'; letter++) {
12             fputc(letter,  fp);
13     }
14    fclose(fp);
15 }
```

ตัวอย่างโปรแกรมที่ 15.9 เป็นการแสดงตัวอย่างโปรแกรมที่ทำการสร้างแฟ้มข้อมูล หรือเขียนข้อมูลทับในแฟ้มข้อมูลชื่อว่า alphabet.txt ในบรรทัดที่ 11 จะเป็นการวนลูปตัวแปร letter ที่มีค่าตั้งแต่ตัวอักษร A ถึง Z โดยบรรทัดที่ 12 จะเป็นการเขียนค่าใน letter ลงไปในแฟ้มข้อมูลทีละตัวตั้งแต่ A จนถึง Z จากนั้นก็ปิดแฟ้มข้อมูล รูปที่ 15.3 แสดงข้อมูลในแฟ้มข้อมูล alphabet.txt หลังจากที่รันโปรแกรมตัวอย่างที่ 15.9

alphabet.txt
ABCDEFGHIJKLMNOPQRSTUVWXYZ

รูปที่ 15.3 ข้อมูลในแฟ้มข้อมูล alphabet.txt

15.5.2 ฟังก์ชัน fputs( )

ฟังก์ชัน fputs( ) เป็นฟังก์ชันที่ใช้เขียนข้อความลงในแฟ้มข้อมูล โดยรูปแบบการใช้งานมีดังนี้

fputs(str, fp);
• str : ตัวแปรประเภทข้อความที่ต้องการจะเขียนลงไปในแฟ้มข้อมูล • fp : ตัวชี้ตำแหน่งแฟ้มข้อมูลที่ต้องการจะเขียนข้อมูล

ตัวอย่างโปรแกรมที่ 15.10

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char **argv) {
4      FILE   *fp;
5      char str[64];
6      fp = fopen(“name.txt”, “w”);
7      if( fp == NULL ) {
8              printf("Can’t open file\n”);
9              exit(0);
10     }
11     for(n = 0;  n < 5;  n++) {
12             printf(“Enter name : “);
13             gets(str);
14              fputs(str);
15     }
16    fclose(fp);
17 }
```

ตัวอย่างโปรแกรมที่ 15.10 มีการทำงานคร่าวๆ ดังนี้โปรแกรมจะทำการเปิดแฟ้มข้อมูลชื่อ name.txt เพื่อเขียน จากนั้นจะทำการวนลูปจำนวน 5 รอบ เพื่อรับค่าจากแป้นพิมพ์ และ นำค่าที่รับมานั้นเขียนลงไปในแฟ้มข้อมูล โดยรายละเอียดของโปรแกรมมีดังนี้

- บรรทัดที่ 4 – 6 เป็นการเปิดแฟ้มข้อมูลชื่อ name.txt เพื่อเขียนข้อมูล
- บรรทัดที่ 7 – 10 เป็นการตรวจสอบว่าแฟ้มข้อมูลสามารถเปิดเพื่อเขียนได้หรือไม่ถ้ามีปัญหาจะแสดงผลว่า Can't open file และปิดโปรแกรม
- บรรทัดที่ 11 เป็นการตั้งลูปให้วนทั้งหมด 5 รอบ
- บรรทัดที่ 12 แสดงคำว่า Enter name: ออกทางหน้าจอ แบบไม่ขึ้นบรรทัดใหม่
- บรรทัดที่ 13 เรียกใช้คำสั่ง gets( ) เพื่อรับค่าจากแป้นพิมพ์แล้วนำค่าที่รับมาเก็บไว้ในตัวแปร str
- บรรทัดที่ 14 เรียกฟังก์ชัน fputs( ) เพื่อนำค่าที่เก็บไว้ในตัวแปร str ลงไปในแฟ้มข้อมูล
- บรรทัดที่ 16 ปิดการใช้งานแฟ้มข้อมูล

รูปที่ 15.4 แสดงผลการรันของตัวอย่างโปรแกรมที่ 15.10 โดยกำหนดให้ผู้ใช้ป้อนชื่อ Somchai, Somsri, Somsak, Sompong, และ Somphot ตามลำดับ และรูปที่ 15.5 แสดงข้อมูลที่อยู่ในแฟ้มข้อมูล name.txt หลังจากโปรแกรมทำงานเสร็จสิ้น

```
Enter name : Somchai
Enter name : Somsri
Enter name : Somsak
Enter name : Sompong
Enter name : Somphot
```

รูปที่ 15.4 ผลการรันของตัวอย่างโปรแกรมที่ 15.10

name.txt
Somchai
Somsri
Somsak
Sompong
Somphot

รูปที่ 15.5 ข้อมูลที่อยู่ในแฟ้มข้อมูล name.txt หลังจากรันตัวอย่างโปรแกรมที่ 15.10

15.5.3 ฟังก์ชัน fprintf( )

นอกจากการเขียนตัวอักขระและข้อความแล้ว ภาษาซียังสามารถจะเขียนค่าตัวเลขจำนวนเต็มหรือ จำนวนจริง ลงไปยังแฟ้มข้อมูลได้อีกด้วย โดยใช้ฟังก์ชัน fprintf( ) ซึ่งสามารถกำหนดรูปแบบการเขียนข้อมูลลงในแฟ้มข้อมูลได้ เหมือนกับการใช้งานฟังก์ชัน printf( ) เพื่อแสดงผลออกทางหน้าจอ การใช้งานฟังก์ชัน fprintf มีลักษณะดังต่อไปนี้

fprintf(fp, “format”, var);
• fp : ตัวชี้ตำแหน่งแฟ้มข้อมูลที่ต้องการเขียน • format : รหัสควบคุมรูปแบบ เหมือนกับการใช้งาน printf( ) เช่น %d, %f • var : ชื่อตัวแปรซึ่งจะใช้เก็บข้อมูลที่ต้องการเขียนค่าลงในแฟ้มข้อมูล โดยประเภทของตัวแปรจะต้องสอดคล้องกับ format ที่กำหนด

ตัวอย่างโปรแกรมที่ 15.11

```
1  fprintf(fp,  “%d”,  number);
2  fscanf(fp,  “%d %d”,  day,  month);
3  fscanf(fp,  “name : %s  age : %d”,  name,  age);
4  fscanf(fp,  “name : %s\nsalary : %f”,  name,  salary);
```

จากตัวอย่างโปรแกรมที่ 15.11 เป็นเพียงส่วนหนึ่งของโปรแกรมในการอธิบายลักษณะการใช้งานของฟังก์ชัน fprintf( )

- บรรทัดที่ 1 เขียนค่าจำนวนเต็มที่เก็บอยู่ในตัวแปร number ลงแฟ้มข้อมูล
- บรรทัดที่ 2 เขียนค่าจำนวนเต็ม 2 ค่า ที่เก็บอยู่ในตัวแปร day และ month ตามลำดับลงแฟ้มข้อมูล
- บรรทัดที่ 3 เขียนคำว่า name : ตามด้วยข้อความที่เก็บในตัวแปร name และ คำว่า age : ตามด้วยค่าจำนวนเต็มที่เก็บในตัวแปร age ลงในแฟ้มข้อมูล
- บรรทัดที่ 4 เขียนคำว่า name : ตามด้วยข้อความที่เก็บในตัวแปร name จากนั้นขึ้นบรรทัดใหม่ และ คำว่า salary : ตามด้วยค่าจำนวนจริงที่เก็บในตัวแปร salary ลงในแฟ้มข้อมูล

ตัวอย่างโปรแกรมที่ 15.12

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(int argc, char **argv) {
4          FILE *fp;
5          int I = 25;
6          char grade = 'A';
7          if((fp = fopen("test.txt", "w")) == NULL) {
8                  printf("Can not open file");  return 1;
9          }
10         fprintf(fp, "I = %d , grade = %c\n", I, grade);
11         fclose(fp);
12 }
```

ตัวอย่างโปรแกรมที่ 15.12 แสดงการใช้งานของฟังก์ชัน fprintf โดยโปรแกรมจะทำการเปิดแฟ้มข้อมูลชื่อ test.txt เพื่อเขียน และบรรทัดที่ 10 โปรแกรมจะทำการบันทึกคำว่า "I = " ตามด้วยค่าจำนวนเต็มที่เก็บไว้ในตัวแปร I ตามด้วยคำว่า ", grade = " และค่าตัวอักขระที่เก็บไว้ในตัวแปร grade ซึ่งเมื่อรันโปรแกรมแล้ว ข้อมูลในแฟ้มข้อมูล test.txt จะมีค่าดังรูปที่ 15.6

test.txt
I = 25, grade = A

รูปที่ 15.6 ข้อมูลที่อยู่ในแฟ้มข้อมูล test.txt หลังจากรันตัวอย่างโปรแกรมที่ 15.12

15.6 ฟังก์ชันอื่น ๆ ที่เกี่ยวกับการทำงานร่วมกับแฟ้มข้อมูล

นอกจากฟังก์ชันการทำงานพื้นฐานในการทำงานร่วมกับแฟ้มข้อมูลแล้ว ในภาษาซียังมีฟังก์ชันที่ใช้ในการทำงานร่วมกับแฟ้มข้อมูลอื่นๆ ให้ใช้อีกมากมาย เช่น

- fread( ), fwrite( ) สำหรับอ่านและเขียนแฟ้มข้อมูลแบบไบนารี ตามลำดับ
- remove( ) สำหรับลบแฟ้มข้อมูล
- rename( ) สำหรับเปลี่ยนชื่อแฟ้มข้อมูล
- fseek( ) สำหรับเลื่อนตัวชี้ตำแหน่งของแฟ้มข้อมูล