1
ความรู้เบื้องต้น

1.1 อะไรคือโปรแกรมคอมพิวเตอร์ (Computer Program)

คอมพิวเตอร์นั้นได้ถูกสร้างขึ้นมาเพื่อจัดการกับข้อมูลต่างๆ ซึ่งคำว่า *ข้อมูล* ในที่นี้ก็รวมถึงเซ็ตของคำสั่งต่างๆ (Instructions) เช่น การบวกลบเลข การเปรียบเทียบมากกว่าน้อยกว่า การเรียงลำดับข้อมูล เป็นต้น ซึ่งอาจจะจัดเก็บไว้บนแผ่นดิสก์ ฮาร์ดดิสก์ แผ่นซีดี รอม (CD-ROM) หรือในหน่วยความจำแบบถาวรหรือชั่วคราวของคอมพิวเตอร์ เพื่อใช้ในการประมวลผลต่อไป

คอมพิวเตอร์ในยุคปัจจุบันสามารถทำตามคำสั่งเหล่านี้ที่เรากำหนดได้อย่างรวดเร็วด้วยความเร็ว หลายสิบล้านคำสั่งในหนึ่งวินาที เราสามารถกล่าวได้ว่า โปรแกรมสำหรับคอมพิวเตอร์ที่แท้จริงก็คือ ลำดับของคำสั่งต่างๆหรือแผนการทำงานนั้นเอง ซึ่งแต่ละคำสั่งจะต้องจัดอยู่ในกลุ่มของคำสั่งที่คอมพิวเตอร์ สามารถเข้าใจและปฏิบัติตามได้

Niklaus Wirth นักคอมพิวเตอร์ชาวสวิส ผู้ที่ได้ชื่อว่าเป็นผู้คิดค้นภาษาปาสคาส (Pascal) และโอเบอรอน (Oberon) เคยกล่าวไว้ว่า โปรแกรมคอมพิวเตอร์ก็คือผลรวมของ *โครงสร้างของข้อมูล* (Data Structure) และ *อัลกอริทึม* (Algorithm) หรือ วิธีการแก้ไขปัญหา ซึ่งถือว่าเป็นคำจำกัดความของคำว่า โปรแกรมคอมพิวเตอร์ที่สั้นและได้ใจความอย่างยิ่ง โครงสร้างของข้อมูลและอัลกอริทึมนั้นจะไม่ขึ้นอยู่กับชนิดของคอมพิวเตอร์ชนิดใดชนิดหนึ่ง เราลองนึกถึงวิธีการแก้ระบบสมการเชิงเส้นหลายตัวแปรซึ่งมีการกำหนดเป็นขั้นตอนที่เป็นแบบแผน และการตั้งระบบสมการก็เปรียบเสมือนกับการกำหนดโครงสร้างของข้อมูลเพื่อแสดงความสัมพันธ์ระหว่างตัวเลขต่างๆ ซึ่งก็คือข้อมูลนั้นเอง เช่น เราสามารถเขียนความสัมพันธ์ของตัวเลขเหล่านั้นให้อยู่ในรูปของเวกเตอร์หรือเมตริกซ์ เพื่อใช้ในการแก้ระบบสมการต่อไป ในบางครั้งการแก้ไขปัญหาทางคอมพิวเตอร์ต้องการโครงสร้างของข้อมูลที่เหมาะสมกับวิธีการแก้ไขปัญหา ความสัมพันธ์ระหว่างทั้งสองสิ่งนี้จะมี

ผลอย่างมากต่อประสิทธิภาพในการทำงานของโปรแกรมที่เราจะนำไปใช้งานหรือถ้าเราเลือกโครงสร้างของข้อมูลที่ไม่เหมาะสม ก็อาจจะทำให้การคิดค้นวิธีการแก้ปัญหายากขึ้นอีก

เราอาจจะตีความของคำว่าโปรแกรมคอมพิวเตอร์ได้ในอีกแง่หนึ่งเมื่อเรามองวิธีการแก้ปัญหาที่เราคิดขึ้นมาและสามารถเขียนมันให้อยู่ในรูปที่เรานำไปใช้งานกับคอมพิวเตอร์ชนิดใดชนิดหนึ่งได้ โดยใช้ภาษาใดภาษาหนึ่งในการเขียนอธิบายโครงสร้างและวิธีการจัดการกับข้อมูลเหล่านั้น

การที่เราจะเริ่มเขียนโปรแกรมคอมพิวเตอร์ทุกครั้งจะต้องมีการวางแผนในการคิดวิธีการแก้ปัญหาหรืออัลกอริทึมก่อน เพื่อจะให้แน่ใจว่าโปรแกรมที่เราเขียนขึ้นสามารถใช้แก้ปัญหาได้จริง เราลองมาพิจารณาดูตัวอย่างของปัญหาและเขียนขั้นตอนการทำงานของโปรแกรม เช่น เราต้องการหาผลรวมของเลขจำนวนนับเริ่มต้นจากเลขหนึ่งตามด้วย สอง สาม สี่ และนับไปเรื่อยๆ ไปจนถึงหนึ่งร้อยโดยใช้วิธีบวกเลขแต่ละจำนวนเข้าด้วยกัน เราใช้ตัวแปร (Variable) หรือ ตัวเก็บค่าผลลัพธ์ที่เราต้องการโดยให้ชื่อว่า `sum` เป็นตัวเก็บค่าของผลรวมที่เกิดจากการบวกตัวเลขในแต่ละครั้ง และใช้ตัวแปร `i` แทนตัวเลขจำนวนนับที่ เราใช้นับตั้งแต่หนึ่งขึ้นไป และเราจะนับจำนวนถึงตัวเลขตัวสุดท้ายคือ หนึ่งร้อย โดยใช้ตัวแปร `N` แทนตัวเลขตัวนี้

เงื่อนไข

`1 ≤ i ≤ N, N=100` (`i` จะมีค่าอยู่ระหว่าง 1 และ 100 ในระหว่างการนับ)

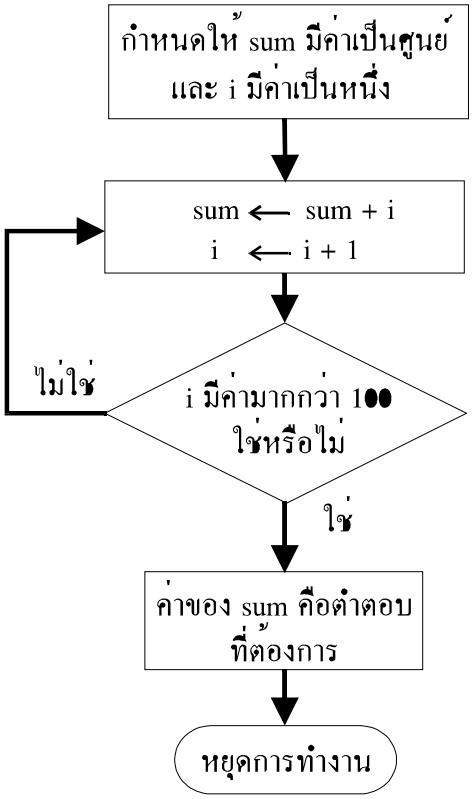
วิธีการ

1. กำหนดตัวแปร `N` ให้มีค่าเป็นหนึ่งร้อย และไม่เปลี่ยนค่าของ `N` ในระหว่างการนับ
2. กำหนดตัวแปร `i` เริ่มต้นให้มีค่าเป็นหนึ่ง
3. กำหนดค่าผลรวม `sum` ให้มีค่าเป็นศูนย์ ก่อนที่เราจะเริ่มนับ เพราะถ้าไม่มีการเริ่มนับผลรวมของตัวเลขที่เกิดจากการนับ ย่อมมีค่าเป็นศูนย์
4. ให้บวกตัวเลขค่าของ `i` ในขณะนั้นเข้ากับค่าผลรวม `sum` และก็ให้เพิ่มค่าของ `i` ขึ้นอีกหนึ่ง ซึ่งก็คือ ค่าของตัวเลขจำนวนนับตัวถัดไป และ `i` ทำหน้าที่เป็นตัวแปรที่บอกเราว่า มีการนับถึงเลขใดแล้ว
5. ถ้าตัวแปร `i` มีค่าเกิน `100` ก็ให้หยุดนับ และผลรวมของตัวเลขในขณะนั้น คือ ผลลัพธ์ที่เราต้องการ ถ้า `i` ยังมีค่าไม่เกิน `100` ก็ให้กลับไปทำขั้นตอนที่สี่ซ้ำอีก

เริ่มนับตั้งแต่ `i = 1` ไปจนถึง `i = N` ซึ่ง `N` มีค่าเท่ากับ `100` โดยใช้วิธีการทำขั้นตอนที่สี่ และห้าซ้ำ หลายๆ ครั้งเมื่อเงื่อนไขยังคงถูกต้อง คือ `i` มีค่าไม่เกิน `100` เราเรียกการทำงานของโปรแกรมในลักษณะ นี้ว่า Looping หรือการวนซ้ำ

สำหรับการบวกเลขจำนวนนับแบบนี้ ถ้าเรามานั่งนับทีละตัวและบวกมันเข้าด้วยกันก็เป็นงานที่น่าเบื่อสำหรับเราและบางทีก็อาจจะมีการบวกเลขผิดก็ได้ งานแบบนี้เหมาะสำหรับคอมพิวเตอร์หรือคนที่ว่างงานหรือนอนไม่หลับเท่านั้น ทางเลือกอีกทางหนึ่งหรืออีกวิธีหนึ่งที่ง่ายคิดในใจก็ได้ ก็คือการใช้สูตร

```
sum = 1 + 2 + 3 + ... + (N-1) + N  =  N*(N + 1)/2
```



รูปภาพประกอบ 1.1

ในบางกรณี เราไม่อาจจะคิดสูตรลัดในการแก้ปัญหาเพื่อประหยัดเวลา และเพิ่มประสิทธิภาพในการคำนวณได้ แต่เป็นเพราะคอมพิวเตอร์ทำงานได้อย่างรวดเร็วเราสามารถคำนวณได้เร็วขึ้น ในขณะที่งานชนิดเดียวกันจะกลายเป็นงานที่ใช้เวลามากและน่าเบื่อเมื่อเราต้องมาคำนวณโดยใช้ดินสอ และกระดาษทด หรือแม้กระทั่งใช้เครื่องคิดเลขแล้วก็ตาม เช่น นักสถิติต้องการจะหาค่าเฉลี่ยของข้อมูลที่เก็บจากการวัดปริมาณของก๊าซโอโซนในแต่ละวัน โดยวัดวันละสองครั้งในระหว่างช่วงหกเดือนที่ผ่านมา และต้องการจะหาค่าเบี่ยงเบนของข้อมูลด้วย ก็คงเป็นงานที่ต้องใช้เวลามาก ถ้าใช้เครื่องคิดเลขแบบธรรมดาในการคำนวณ แต่ถ้าใช้คอมพิวเตอร์ก็ใช้เวลาน้อยลง ในกรณีนี้ โปรแกรมที่เราเขียนขึ้นก็มีขั้นตอนการหาค่าผลรวมของข้อมูลตัวเลขที่ไม่

แตกต่างกับการใช้เครื่องคิดเลขมากนัก ซึ่งก็คือการบวกข้อมูลตัวเลขทีละตัวเข้าด้วยกัน เพียงแต่คอมพิวเตอร์สามารถประมวลผลได้เร็วกว่า สมมุติว่า เราไม่ทราบว่าจำนวนของข้อมูลมีทั้งหมดกี่จำนวน หลังจากที่หาผลรวมแล้วก็ต้องมานับเองอีกว่ามีข้อมูลทั้งหมดเท่าไหร เพื่อที่จะใช้ในการหาค่าเฉลี่ย ถ้าเครื่องคิดเลขไม่มีหน่วยความจำมากพอสำหรับข้อมูลที่เราได้ป้อนเข้าไป เพื่อเก็บเอาไว้ใช้ต่อไปในการคำนวณค่าทางสถิติตัวอื่นๆ ดังนั้นในการหาค่าเบี่ยงเบนของข้อมูลเราก็ต้องมากดตัวเลขใหม่อีกครั้ง เพื่อที่จะให้เครื่องคิดเลขคำนวณหาค่าเบี่ยงเบนของข้อมูลที่เราต้องการ

ถ้าเราเขียนโปรแกรมคอมพิวเตอร์ เราก็อาจจะเขียนขั้นตอนที่มีหน้าที่นับตัวเลขหรือข้อมูลที่เราป้อนให้คอมพิวเตอร์ในครั้งแรก และเก็บข้อมูลนี้ไว้ในหน่วยความจำเพื่อใช้จะใช้ในการคำนวณในครั้งต่อไป ทำให้เราไม่ต้องมานั่งป้อนข้อมูลหลายๆครั้ง นี้ก็เป็นเหตุผลง่ายๆเหตุผลหนึ่ง ทำไมเราจึงหันมาใช้คอมพิวเตอร์ และหัดเขียนโปรแกรมเพื่อใช้ในการประมวลผลข้อมูลต่างๆ

ขั้นตอนการแก้ปัญหาที่เราได้เขียนเป็นข้อๆตามลำดับ เราอาจจะเขียนมันให้อยู่ในรูปของแผนผังการทำงานแบบกราฟิกซึ่งเรียกว่า Control Flow หรือ Flowchart ทำให้เรามองภาพรวมของแผนงานได้ง่ายขึ้นดังในรูปภาพประกอบ 1.1

เมื่อเราได้คิดวิธีการแก้ปัญหาเสร็จแล้ว เราก็จะต้องเขียนวิธีการแก้ปัญหาของเราให้อยู่ในรูปของโปรแกรมคอมพิวเตอร์โดยใช้ภาษาใดภาษาหนึ่ง เช่น Pascal, FORTRAN, C เป็นต้น การเขียนโปรแกรมคอมพิวเตอร์ก็ไม่แตกต่างจากการเขียนอธิบายวิธีการแก้ปัญหาที่เป็นขั้นตอนดังที่ได้ลองยกตัวอย่างไปแล้ว โดยเรารู้ว่าในแต่ละขั้นตอนจะต้องทำอะไรบ้างและผลที่เกิดขึ้นในแต่ละขั้นตอนคืออะไร ซึ่งเราสามารถตรวจสอบได้ทุกขั้นตอน เพราะเมื่อมีขั้นตอนใดที่ทำงานไม่ถูกต้องเราสามารถตรวจสอบได้จากผลที่เกิดขึ้นในขั้นตอนนั้น

1.2 ภาษาคอมพิวเตอร์ (Computer Language)

ถ้าเราจะใช้คอมพิวเตอร์ เราก็มีความจำเป็นต้องเขียนโปรแกรมคอมพิวเตอร์สำหรับใช้ในงานต่างๆขึ้นมาใช้ (หรืออาจจะมีผู้อื่นเขียนขึ้นและขายผลงานของเขาให้เราใช้ ซึ่งคนพวกนี้เราเรียกว่า โปรแกรมเมอร์ หรือนักเขียนโปรแกรมคอมพิวเตอร์อาชีพ) แล้วก็ต้องเผชิญกับคำถามที่ว่า เราจะใช้ภาษาใดลักษณะใดในการเขียนคำสั่งต่างๆที่คอมพิวเตอร์สามารถเข้าใจได้

ในยุคแรกของคอมพิวเตอร์ คำสั่งสำหรับคอมพิวเตอร์จะเขียนโดยใช้ภาษาเครื่อง (Machine Code) ซึ่งก็คือการป้อนข้อมูลหรือคำสั่งที่อยู่ในรูปของรหัสตัวเลข เช่น 0110110110... ข้อมูลในรูปแบบนี้เหมาะสมกับการทำงานของคอมพิวเตอร์ เพราะใช้ในการสลับสัญญาณไฟฟ้าในส่วนต่างๆ เพื่อควบคุมการทำงานของคอมพิวเตอร์ หรือใช้เป็นสัญลักษณ์แสดงข้อมูลต่างๆในหน่วยความจำ แต่มันก็ไม่เหมาะสมกับมนุษย์ เพราะยากต่อการทำความเข้าใจ และยากต่อการ

ดัดแปลงแก้ไข โดยเฉพาะเมื่อโปรแกรมมีความซับซ้อนมากและมีขนาดใหญ่ขึ้น ภาษาเครื่องจัดเป็นภาษาระดับต่ำ ภาษาที่ใช้เขียนโปรแกรมคอมพิวเตอร์ในยุคปัจจุบันจัดเป็นภาษาระดับสูง (High-Level Programming Language) เช่น C/C++, FORTRAN, Pascal, BASIC, COBOL และอื่นๆอีกมากมาย ใช้ในงานแตกต่างกันไปภาษาระดับสูง มีลักษณะคล้ายภาษาของมนุษย์ คือ ภาษาอังกฤษใช้คำสั้นและกระทัดรัดซึ่งง่ายต่อการทำความเข้าใจ แต่อย่างไรก็ตามโปรแกรมที่เขียนโดยใช้ภาษาระดับสูง จะต้องถูกแปลงให้เป็นภาษาเครื่องก่อน โดยสิ่งที่เรียกว่า คอมไพล์เลอร์ (Compiler) หรือ ตัวแปลชุดคำสั่ง โปรแกรมก่อนที่จะถูกแปลงเป็นภาษาเครื่อง เรามักเรียกว่า ชุดคำสั่งต้นฉบับ (Source Code หรือ Program Code)

คอมไพล์เลอร์ ก็คือโปรแกรมคอมพิวเตอร์ชนิดหนึ่งมีหน้าที่แปลงคำสั่งหรือโปรแกรมโค้ดที่เขียนด้วยภาษาระดับสูงภาษาใดภาษาหนึ่ง เช่น ภาษาซี ให้เป็นภาษาเครื่อง โปรแกรมที่ถูกแปลงเป็นภาษาเครื่องแล้วเราเรียกว่า Object Code หรือ ชุดคำสั่งภาษาเครื่อง ทำให้โปรแกรมที่เราสร้างขึ้นสามารถทำงานบนเครื่องคอมพิวเตอร์ที่เราต้องการแบบใดแบบหนึ่งได้ ภายใต้ระบบปฏิบัติการตัวใดตัวหนึ่ง เพราะโครงสร้างของภาษาเครื่องนั้นขึ้นอยู่กับชนิดของคอมพิวเตอร์และฮาร์ดแวร์

แต่ในทางกลับกันโครงสร้างของภาษาระดับสูงจะต้องขึ้นอยู่กับฮาร์ดแวร์ของคอมพิวเตอร์น้อยที่สุด เพราะในบางครั้ง เราเขียนโปรแกรมสำหรับคอมพิวเตอร์ชนิดหนึ่งขึ้นใช้ เราก็ต้องการที่จะให้โปรแกรมที่เราเขียนขึ้นนี้ สามารถใช้กับคอมพิวเตอร์แบบอื่นได้ด้วย โดยเพียงแค่คอมไพล์โปรแกรมใหม่อีกครั้งให้เป็นออบเจคโค้ด หรืออาจจะมีการแก้ไขเพียงเล็กน้อยเท่านั้น การเขียนโปรแกรมโดยใช้ภาษาสูงทำให้เราไม่ต้องกังวลถึงเรื่องฮาร์ดแวร์ และปล่อยให้เป็นหน้าที่ของคอมไพล์เลอร์ในการสร้างส่วนของโปรแกรมที่ทำงานกับคอมพิวเตอร์ที่เราต้องการได้

1.3 ความผิดพลาดในการเขียนโปรแกรมคอมพิวเตอร์

ความผิดพลาดหรือข้อบกพร่องในโปรแกรมที่เราเขียนขึ้นสามารถแบ่งออกได้เป็นสามจำพวก

ความผิดพลาดในการเขียนโปรแกรมคอมพิวเตอร์

- ความผิดพลาดที่เกิดจากการใช้ไวยกรณ์ที่ไม่ถูกต้อง (Syntax Errors)
- ความผิดพลาดที่เกิดขึ้นในระหว่างการรันโปรแกรม (Run-time Errors)
- ความผิดพลาดเนื่องจากวิธีการทำงานของโปรแกรมที่ไม่ถูกต้อง (Logical Errors)

1) **ความผิดพลาดที่เกิดจากการใช้ไวยกรณ์ที่ไม่ถูกต้อง** (Syntax Errors)

เมื่อเราเขียนโปรแกรมไม่ว่าจะในภาษาใดก็ตาม บางครั้งเราเขียนหรือใช้คำสั่งไม่ถูกต้องตามหลักไวยากรณ์ของภาษา ความผิดพลาดแบบนี้มักเกิดจากความไม่รอบคอบของโปรแกรมเมอร์ เช่น พิมพ์ตัวอักษรผิด ใช้คำสั่งหรือสัญลักษณ์ที่คอมไพล์เลอร์ไม่ได้จำกัดความเอาไว้ ลืมใส่วงเล็บปิดหลังจากที่ใช้วงเล็บเปิด ทำให้ไม่ครบคู่ เหล่านี้เป็นต้น เมื่อทำการคอมไพล์โปรแกรม คอมไพล์เลอร์เมื่อตรวจพบความผิดพลาดใดๆ ก็จะรายงานให้เราทราบว่า มีส่วนใดในโปรแกรมที่ผิดอยู่ ซึ่งจะแจ้งออกมาในรูปของชนิดของความผิดพลาด และอยู่ในบรรทัดที่เท่าไรของโปรแกรมโค้ด รวมทั้งสาเหตุของความผิดพลาดที่น่าจะเป็นไปได้ ซึ่งทั้งหมดนี้ รวมเรียกว่า Error Messages บางครั้งเป็นแค่ความผิดพลาดเล็กน้อยซึ่งไม่มีผลต่อส่วนอื่นๆของโปรแกรมมากนัก คอมไพล์เลอร์ก็สามารถค้นหาความผิดพลาดตรงจุดอื่นในโปรแกรมต่อไปได้ แต่ถ้ามีความผิดพลาดที่มีผลต่อส่วนอื่นของโปรแกรมอย่างมาก เพราะความผิดพลาดนี้ย่อมทำให้ส่วนอื่นของโปรแกรมผิดพลาดตามไปด้วย คอมไพล์เลอร์ก็จะหยุดการทำงานต่อไปโดยแจ้งข้อผิดพลาดให้เราทราบ

เมื่อตรวจพบจุดใดที่ยังผิดอยู่ เราก็ต้องมาแก้ไขที่จุดนั้น บางคนมีประสบการณ์ในการเขียนโปรแกรมมากสามารถแก้ไขได้ทันที บางคนก็อาจจะใช้เวลามากหน่อยเพราะยังไม่เชี่ยวชาญ ยังไม่แม่นในหลักไวยากรณ์ของภาษา แต่สิ่งที่สำคัญสำหรับผู้ที่เริ่มหัดเขียนโปรแกรมก็คือ ทุกครั้งที่มีปัญหาเมื่อได้แก้ไขปัญหาหรือความผิดพลาดในโปรแกรมแล้ว ก็ควรจะจดบันทึกไว้ว่ามีความผิดพลาดในลักษณะใดเกิดขึ้น และแก้ไขได้อย่างไร เพราะถ้ามีความผิดพลาดในลักษณะเช่นนี้เกิดขึ้นอีก เราก็สามารถเปิดดูสิ่งที่เราได้จดบันทึกไว้ ซึ่งบางทีเราอาจจะลืมไปแล้ว ดีกว่ามานั่งลองผิดลองถูกอีกครั้ง และจะทำให้เสียเวลามาก

2) **ความผิดพลาดที่เกิดขึ้นในระหว่างการรันโปรแกรม** (Run-time Errors)

หลังจากที่เราได้คอมไพล์โปรแกรมเสร็จแล้ว โดยไม่มีความผิดพลาดใดๆ เมื่อเรารันโปรแกรมเพื่อที่จะให้มันทำงานตามที่เราต้องการ ในขณะที่โปรแกรมกำลังทำงานบางครั้งก็เกิดปัญหาขึ้นจนโปรแกรมไม่สามารถทำงานต่อไปได้ ความผิดพลาดในลักษณะนี้เกิดจากหลายสาเหตุ เช่น การหารเมื่อตัวส่วนมีค่าเป็นศูนย์ (Zero Division) หน่วยความจำที่โปรแกรมต้องการใช้มีไม่เพียงพอ การทำงานของโปรแกรมที่ละเมิดกฎของระบบปฏิบัติการ เช่น โปรแกรมพยายามเขียนทับหน่วยความจำที่อ่านได้อย่างเดียวเท่านั้น หรืออาจจะเกิดจากความผิดพลาดในการทำงานของชิ้นส่วนแต่ละชิ้นของคอมพิวเตอร์หรือสาเหตุอื่นๆจนทำให้โปรแกรมไม่สามารถทำงานต่อไปได้

3) **ความผิดพลาดที่เกิดขึ้นเพราะวิธีการทำงานของโปรแกรมที่ไม่ถูกต้อง** (Logical Errors)

ความผิดพลาดชนิดนี้มักตรวจพบได้ยากและมักจะแสดงตัวให้เห็นก็ต่อเมื่อได้มีการใช้ซอฟต์แวร์ไปแล้วหลายครั้ง แต่ไม่มีผลต่อการทำงานของโปรแกรมโดยตรง เพียงแต่ในบางกรณีโปรแกรมไม่ได้ให้ผลตามที่เราต้องการ เช่น ถ้าเราป้อนข้อมูลชุดหนึ่งให้โปรแกรม และเราก็รู้ว่าข้อมูลหรือผลของการทำงานจะออกมามีรูปร่างหน้าตาเป็นอย่างไร แต่โปรแกรมกลับให้ข้อมูลที่แตกต่างจากที่เราคาดหวังเอาไว้ ก็ให้เราสงสัยไว้ก่อนว่าต้องมีส่วนใดส่วนหนึ่งในโปรแกรมที่ทำงานไม่ถูกต้อง ซึ่งก็มีได้หลายสาเหตุ เช่น วางขั้นตอนการทำงานที่ไม่ถูกต้อง หรือเกิดจากความไม่รอบคอบของโปรแกรมเมอร์เอง

ซอฟต์แวร์ทุกตัวก่อนที่จะนำไปใช้งานจริงจะต้องมีการตรวจสอบค้นหาความผิดพลาดที่อาจจะแอบแฝงอยู่ในโปรแกรม ขั้นตอนนี้เราก็เรียกว่า Debugging และความผิดพลาดของโปรแกรมที่เราตรวจพบก็ เรียกว่า Bugs ปัจจุบันก็มีเครื่องมือที่ทำหน้าที่เป็น Debugger ซึ่งจำหน่ายมาพร้อมกับคอมไพล์เลอร์

อย่างไรก็ตาม แม้ว่าโปรแกรมจะได้รับการตรวจสอบอย่างดี ก็ใช่ว่าปราศจากข้อบกพร่องเสมอไป

1.4 ประวัติความเป็นมาของภาษาซี

ในประมาณปี ค.ศ. 1967 ภาษาที่มีชื่อว่า BCPL ได้ถูกประดิษฐ์ขึ้น เพื่อใช้ในการสร้างระบบปฏิ-บัติการ (Operating System) และการออกแบบคอมไพล์เลอร์ (Compiler) สำหรับคอมพิวเตอร์ในยุคนั้น ต่อมา Ken Thompson ได้พัฒนาภาษาชื่อ B ซึ่งมีรากฐาน มาจากภาษาบีซีพีแอลโดยตรง และในปี ค.ศ. 1970 ก็ได้มีการใช้ภาษานี้ในการสร้างระบบปฏิบัติการแบบ UNIX รุ่นแรกที่ AT&T Bell Laboratories

ในปีค.ศ. 1972 นักคอมพิวเตอร์ของ AT&T Bell Lab. ชื่อ Dennis Ritchie ได้วางแผนออกแบบ ภาษา C เพื่อที่จะใช้ในการปรับปรุงระบบปฏิบัติการแบบ UNIX ใหม่อีกครั้งเพราะภาษาบีซีพีแอล และภาษาบียังไม่มีประสิทธิภาพเท่าที่ควร ในอีกประมาณสองปีต่อมา UNIX ก็ได้ถูกเขียนขึ้นใหม่โดยใช้ภาษาซี และเราจะเห็นได้ว่าเกือบจะทำหมดของระบบปฏิบัติการแบบ UNIX ที่ใช้กันอยู่ในปัจจุบันนั้นถูกเขียนขึ้นโดยใช้ภาษาซี และสามารถกล่าวได้ว่า ภาษาซีนั้นเติบโตมากับระบบปฏิบัติการแบบ UNIX ภาษาซีได้รับความนิยมขึ้นมาตามลำดับโดยเฉพาะในส่วนของมหาวิทยาลัย แต่ในยุคเริ่มต้นนั้นภาษาซีก็ยังแพร่หลายอยู่ในหมู่ของผู้เชี่ยวชาญทางการเขียนโปรแกรมคอมพิวเตอร์เท่านั้น

Kernighan และ Ritchie ได้นำเสนอผลงานของเขาคือ หลักการใช้ภาษาซี โดยเขาให้ชื่อว่า K&R C เป็นการวางโครงสร้างของภาษาซี ซึ่งในไม่ช้าก็ได้กลายเป็นมาตราฐานการเขียนภาษาซีในยุคนั้น การพัฒนาทางด้านฮาร์ดแวร์ของคอมพิวเตอร์อย่างรวดเร็วทำให้คอมพิวเตอร์สามารถทำงานได้รวดเร็วและมีประสิทธิภาพมากขึ้นหลายเท่า ส่งผลให้ผู้ผลิตซอฟต์แวร์ได้หันมาให้ความสนใจที่จะทำซอฟต์แวร์และระบบปฏิบัติการสำหรับคอมพิวเตอร์ขนาดเล็กและได้มีการสร้างคอมไพล์เลอร์ที่ทำงานกับเครื่องคอมพิวเตอร์แบบพีซี (Personal Computer) ได้โดยเฉพาะสำหรับภาษาซี

แต่อย่างไรก็ตาม ในบางจุดของ K&R C ก็ยังกำหนดไม่ชัดเจน ทำให้ผู้ผลิตคอมไพล์เลอร์ทั้งหลาย ตีความแตกต่างกันไปในการใช้ออกแบบสร้างคอมไพล์เลอร์ของตน ในที่สุด ANSI C (ANSI ย่อมาจาก American National Standards Institute) ก็ได้กำเนิดขึ้น เพื่อแก้ไขปัญหานี้จนกลายเป็นมาตราฐานของภาษาซีจนถึงยุคปัจจุบัน ซึ่งผู้ผลิตคอมไพล์เลอร์ทั้งหลายจะต้องทำตามมาตราฐานนี้

แม้ว่าภาษาซีจะได้รับการปรับปรุงแก้ไขอย่างมากในช่วงที่ผ่านมา แต่ภาษาซียังคงมิใช่ภาษาที่สมบรูณ์แบบและดีที่สุด แต่แทนที่ภาษาซีจะได้รับการแก้ไขปรับปรุงต่อไปโดยตรง ก็ได้มีการคิดค้นภาษาใหม่ขึ้นมา ซึ่งภาษาใหม่นี้ก็คือ C++ (อ่านว่า ซีพลัซพลัซ) โดย Bjarne Stroustrup

ภาษาซีพลัซพลัซ นี้ก็ยังคงใช้โครงสร้างของภาษาซีเดิมเป็นหลัก แต่ได้มีการเพิ่มเติมคุณสมบัติหลายๆประการเข้าไป และคุณสมบัติที่สำคัญที่สุดก็คือ การสนับสนุนการเขียนโปรแกรมโดยเน้นหลักการของออบเจค (Object-Oriented Programming หรือใช้คำย่อว่า OOP) เพราะช่วยให้นักเขียนโปรแกรมคอมพิวเตอร์สามารถจัดการกับซอฟต์แวร์ที่มีขนาดใหญ่และซับซ้อนได้ง่ายและมีประสิทธิภาพมากขึ้น

การมองปัญหาให้อยู่ในรูปของออบเจค หรือเราอาจจะพิจารณาให้อยู่ในรูปของระบบที่มีขอบเขตและคุณสมบัติในตัวมันเอง โดยเราจำแนกออบเจคหนึ่งออกจากออบเจคตัวอื่นๆที่มาประกอบเข้ากันเป็นโครงสร้างที่ซับซ้อนขึ้น แต่ละออบเจคจะมีคุณสมบัติที่แตกต่างกันไป ถ้าออบเจคหลายตัวที่มีลักษณะคุณสมบัติเหมือนกัน เราก็จัดให้มันอยู่ในประเภทเดียวกันหรือเรียกว่า คลาส (Class) เป็นเสมือนกับต้นแบบหรือโครงสร้างของออบเจค และนอกจากนี้ก็จะมีการกำหนดวิธีการจัดการกับข้อมูลเหล่านั้นด้วยเพื่อใช้กับข้อมูลของออบเจค ยกตัวอย่างเช่น ถ้าเราต้องการเขียนโปรแกรมสำหรับวาดรูปวงกลม เราก็มองการวาดรูปวงกลมเสมือนเป็นออบเจคหนึ่ง (หรือเราจะเรียกว่าคลาสได้ถ้าเรามองตามโครงสร้าง) ประกอบด้วยข้อมูลสมาชิก (Member Data) อาทิเช่น จุดศูนย์กลาง รัศมี ความหนาของเส้น สีของเส้นวงกลม เหล่านี้เป็นต้น เมื่อเรามีคุณสมบัติของ ออบเจคแล้ว เราก็ต้องกำหนดและสร้างฟังก์ชันสมาชิก (Member Method) ขึ้นมาจัดการข้อมูลภาย

ในออบเจค เพราะตามหลักของ OOP ไม่อนุญาตให้ฟังก์ชันอื่นที่ไม่ใช่สมาชิกของ ออบเจคเข้ามาเกี่ยวข้องกับข้อมูลดังกล่าว เช่น การวาดรูปวงกลมโดยใช้ข้อมูลที่อยู่ในออบเจค การเปลี่ยนรัศมีของวงกลม การย้ายพิกัดของจุดศูนย์กลางของวงกลม เป็นต้น ถ้าต้องการอ่านหรือเปลี่ยนแปลงแก้ไขข้อมูลของออบเจคก็ต้องเรียกใช้ฟังก์ชันสมาชิกที่มีหน้าที่เหล่านี้เท่านั้น ฟังก์ชันสมาชิกทำหน้าที่เป็นตัวกลางระหว่างข้อมูลภายในออบเจค และสิ่งที่อยู่ภายนอกของออบเจค

เหล่านี้เป็นเพียงคุณสมบัติหนึ่งของการเขียนโปรแกรมคอมพิวเตอร์ซึ่งใช้ภาษาซีพลัซพลัซเน้นการมองปัญหาแบบออบเจค เราสามารถกล่าวได้ว่าภาษาที่ใช้ในการเขียนโปรแกรมคอมพิวเตอร์ในยุคปัจจุบันจะต้องมีคุณสมบัติข้อนี้

ภาษาซีพลัซพลัซเป็นน้องใหม่ กำลังได้รับความนิยมอย่างมากเมื่อเทียบกับภาษาซีซึ่งเป็นรุ่นพี่ ไม่ว่าจะเป็นนักเขียนโปรแกรมคอมพิวเตอร์ระดับมืออาชีพหรือมือสมัครเล่นก็ตาม ตอนนี้ใครๆก็อยากเริ่มต้นเรียนภาษาซีพลัซพลัซมากกว่าภาษาซี และยิ่งไปกว่านั้นภาษาซีพลัซพลัซก็เป็นรากฐานของภาษาอื่นๆเช่น จาวา (Java) ที่กำลังก้าวขึ้นมามีบทบาทอย่างมากในปัจจุบันและอนาคต แต่อย่างไรก็ตามภาษาซีพลัซพลัซจะมีประโยชน์มากกว่าภาษาซีก็ต่อเมื่อเราได้ใช้คุณสมบัติข้อสำคัญๆของภาษาซีพลัซพลัซอย่างเต็มที่ ถ้าปราศจากการใช้คุณสมบัติเหล่านี้ ภาษาซีพลัซพลัซก็อาจจะลดลงไปอยู่ระดับเดียวกับภาษาซี

1.5 โครงสร้างและองค์ประกอบของโปรแกรมภาษาซี

ถ้าใครเป็นนักเล่นเครื่องเสียงก็จะทราบดีว่า เครื่องเสียงหนึ่งชุด ก็จะประกอบไปด้วยส่วนต่างๆที่นำมาต่อเข้าด้วยกัน เช่น เครื่องขยายสัญญาณเสียง เครื่องเล่นแผ่นซีดี เครื่องเล่นเทปเพลง เครื่องรับสัญญาณวิทยุ เครื่องปรับเปลี่ยนสัญญาณ และ ลำโพงกระจายเสียงเป็นต้น แต่ละส่วนมีหน้าที่แตกต่างกันทุกๆส่วน เป็นอิสระต่อกันและอาจจะมาจากผู้ผลิตเดียวกันหรือต่างยี่ห้อกันก็ได้ โดยเลือกตามความพอใจของผู้เล่น แต่ทุกส่วนต้องทำงานร่วมกันได้ เช่นเครื่องขยายสัญญาณเสียงจะต้องรับสัญญาณไฟฟ้าจากเครื่องเล่นเทปเพลงได้ และส่งต่อไปยังลำโพงโดยคุณภาพของเสียงไม่แตกต่างจากที่บันทึกไว้ในเทปเพลง

ชุดเครื่องเสียงก็เป็นตัวอย่างหนึ่งของระบบ (System) ที่เราสามารถแบ่งออกเป็นส่วนย่อยลงไปอย่างคร่าวๆตามหน้าที่ของมัน โปรแกรมในภาษาซีก็มีลักษณะเช่นเดียวกัน คือมีการแบ่งโปรแกรมออก เป็นกลุ่มของขั้นตอนการทำงานโดยอยู่ในรูปของฟังก์ชัน (Function) ซึ่งไม่ใช่ฟังก์ชันทางคณิตศาสตร์ตามที่ผู้อ่านบางคนเข้าใจ ในภาษาปาสคาล (Pascal) ก็มีฟังก์ชันในโครงสร้างของภาษา โดยใช้ชื่อว่าโพรซีเจอร์ (Procedure)

ฟังก์ชันทางคณิตศาสตร์อาจเป็นฟังก์ชันหลายตัวแปร เช่น x_1, x_2 ,..., x_n และให้ค่าของฟังก์ชัน เพียงค่าเดียวซึ่งเราเขียนให้อยู่ในรูป ของ $y = f(x_1, x_2, ..., x_n)$ โดยมีตัวแปร x_1, x_2 ,..., x_n ซึ่งก็คือข้อมูล เราที่ป้อนเข้าไป (Input) ให้ฟังก์ชันหรือเรียกว่า อาร์กิวเมนท์ของฟังก์ชัน และ y ก็คือข้อมูลที่ฟังก์ชันสร้าง ออกมา (Output) แต่ในภาษาซี ฟังก์ชันไม่จำเป็นที่จะต้องรับข้อมูลใดๆ คือไม่มีอาร์กิวเมนท์ของฟังก์ชันหรือคืนข้อมูลใดๆออกมา และเวลาเราผ่านข้อมูลหลายๆตัวให้ฟังก์ชันข้อมูลเหล่านี้ก็อาจจะเป็นข้อมูลต่างชนิดกันก็ได้ ในบทที่ 4 เราจะได้เรียนรู้เกี่ยวกับการสร้างฟังก์ชันในภาษาซีอย่างละเอียด แต่ตอนนี้เราลองมาพูดถึง เรื่องโครงสร้างของโปรแกรมที่เขียนโดยใช้ภาษาซี เราสามารถแบ่งโปรแกรมออกเป็นองค์ประกอบที่สำคัญสามส่วน ดังนี้

องค์ประกอบที่สำคัญของโปรแกรม

1) ฟังก์ชันหลัก หรือ Main Function
2) การแจ้งหรือตกลงใช้ตัวแปรภายในฟังก์ชัน หรือ Variable Declaration
3) ประโยคคำสั่งหรือขั้นตอนการทำงานในฟังก์ชันหลัก Program Statement

1) **ฟังก์ชันหลัก หรือ** Main Function

ฟังก์ชันหลัก เป็นจุดเริ่มต้นของโปรแกรมนั้นเอง โดยระบบปฏิบัติการที่เราใช้ เช่น MS-DOS หรือ UNIX จะเริ่มต้นรันโปรแกรมที่จุดนี้ของโปรแกรม ทุกๆโปรแกรมที่เขียนด้วยภาษาซีจะต้องมีฟังก์ชันหลัก และมีเพียงหนึ่งฟังก์ชันเท่านั้นที่ทำหน้าที่เป็นฟังก์ชันหลัก ซึ่งมีรูปลักษณ์อย่างง่ายๆ ดังต่อไปนี้

```
main()
{
      variable declarations;
      program statements;
}
```

โดยคำว่า `main` เป็นชื่อที่บอกให้ทราบว่า ฟังก์ชันนี้ทำหน้าที่เป็นฟังก์ชันหลักของโปรแกรม ตามปรกติแล้วโปรแกรมสามารถมีฟังก์ชันได้หลายฟังก์ชันและการแบ่งโปรแกรมออกเป็นหลายๆฟังก์ชันก็เหมือนกับการจัดแบ่งกลุ่มในการทำงาน โดยแต่ละกลุ่มจะได้รับมอบหมายงานที่มีหน้าที่แตกต่างกันไป ทำให้ง่ายต่อการวางแผนงาน และในแต่ละกลุ่มก็อาจจะมีการแบ่งออกเป็นกลุ่มย่อยๆอีกโดยมีหัวหน้ากลุ่มเป็นผู้คอยประสานงานและเป็นตัวแทนของกลุ่ม ลักษณะการแบ่งงานเช่นนี้ ก็พบเห็นได้อยู่ทั่วไปในชีวิตประจำวัน โดยเฉพาะเมื่องานที่เราจะต้องทำมีขนาดใหญ่และซับซ้อน

2) **Variable Declaration**

Variable Declaration หมายถึง การแจ้งให้คอมไพล์เลอร์ทราบว่าเราต้องการใช้ตัวแปรใดในการ แทนค่าของข้อมูลชนิดใด (Data Types) และใช้ในฟังก์ชันใด โดยเขียนให้อยู่ในรูปดังต่อไปนี้

```
data_type   variable_name;
```

ตัวอย่างเช่น

```
int    i;
char   alphabet;
```

จากตัวอย่างข้างบน เราใช้ตัวแปรชื่อ `i` โดยกำหนดให้เก็บข้อมูลแบบ `int` ซึ่งก็คือเลขจำนวนเต็มที่มีค่าอยู่ระหว่าง -32768 และ 32767 สำหรับคอมพิวเตอร์ที่มีขนาดความยาวของเวิรด์เท่ากับ 16 บิต เช่น เครื่องพีซี 80386/ 486 เป็นต้น หรือมีค่าอยู่ระหว่าง -2^{31} และ $2^{31} - 1$ สำหรับขนาดความยาวของเวิรด์เท่ากับ 32 บิต ส่วนตัวแปร `alphabet` นั้นใช้แทนข้อมูลแบบ `char` ซึ่งหมายถึง ตัวอักษรในรหัสแอสกี้ (ASCII) ยกตัวอย่างเช่น `'A' 'b' '#'` เป็นต้น ที่มีค่าอยู่ระหว่าง -128 และ 127

3) **Program Statement**

Program Statements หมายถึง ประโยคคำสั่งต่างๆ หรือข้อความที่เป็นส่วนหนึ่งของภาษาซี ทำหน้าที่แจ้งให้คอมไพล์เลอร์ทราบว่า ในโปรแกรมมีขั้นตอนการทำงานใดและอย่างไรบ้าง โปรดสังเกตว่า ไม่ว่าจะเป็นการกำหนดใช้ตัวแปรหรือใช้คำสั่งต่างๆ ก็ตาม จะมีเครื่องหมายอัฒภาค (Semicolon) อยู่ข้างท้าย ซึ่งเป็นการแบ่งขั้นตอนของชุดคำสั่งนั้นเอง เราอาจจะเปรียบเทียบการเขียนโปรแกรมในภาษาซีได้กับการเขียนเรื่องราวในภาษาอังกฤษ เมื่อเวลาจบประโยคหลักแต่ละประโยคก็ต้องมีจุดหรือ Full Stop นั้นเอง

1.5.1 คำอธิบายในโปรแกรม (Program Comment)

ในบางครั้งเราต้องการเขียนคำอธิบายกำกับขั้นตอนการทำงานของโปรแกรมในแต่ละขั้น โดยเฉพาะเมื่อมีใครมาอ่านหรือตรวจสอบโปรแกรมที่เราได้เขียนขึ้น เขา ก็สามารถเข้าใจโปรแกรมของเราได้ง่ายขึ้นหรือบางครั้งคนที่เป็นผู้เขียนโปรแกรมเองก็อาจจะลืมได้ ซึ่งทำให้ไม่เข้าใจในสิ่งที่ตนได้เขียนเอาไว้เมื่อต้องมาอ่านมันอีกครั้ง

คำอธิบายในโปรแกรมภาษาซี จะเริ่มต้นด้วย /* และจบด้วย */ ซึ่งเป็นการบอกให้คอมไพล์เลอร์ ทราบว่า ข้อความที่อยู่ระหว่างสัญลักษณ์ทั้งสองนี้คือคำอธิบายและไม่มีผลต่อการทำงานของโปรแกรม

คำเตือน ถ้าเราเขียนคำอธิบายในโปรแกรม เมื่อเริ่มต้นด้วย /* จะต้องตามด้วยข้อความที่ไม่มี /* เพราะทำให้คอมไพล์เลอร์ไม่เข้าใจ เราจะต้องจบด้วย */ ก่อนที่เริ่มใช้ /* ใหม่อีกครั้ง

ตัวอย่างที่ผิดก็เช่น

```
/* program comment
 *     /* ... nesting comments .... */
 */
```

1.5.2 หลักการตั้งชื่อตัวแปรในภาษาซี

เราสามารถเลือกตัวอักษรที่ประกอบกันขึ้นเป็นชื่อของตัวแปรได้ บางทีเราก็เรียกชื่อต่างๆ ของตัวแปรที่เราต้องการใช้ว่า Identifier หรือ ตัวระบุชื่อ ซึ่งเป็นตัวบอกความแตกต่างของตัวแปรแต่ละตัว ในภาษาซี ชื่อของตัวแปรประกอบด้วยตัวอักษรมากกว่าหนึ่งตัวขึ้นไป โดยเป็นตัวอักษรตั้งแต่ A ถึง Z ไม่ว่า จะเป็นตัวพิมพ์ใหญ่ หรือตัวพิมพ์เล็ก หรือตัวเลขระหว่าง 0 ถึง 9 รวมทั้งขีดล่าง (Undertscore) ตัวอักษร ที่นอกเหนือจากนี้ไปห้ามนำมาใช้ และที่สำคัญ ชื่อของตัวแปรห้ามขึ้นต้นด้วยตัวเลข เราจะตั้งชื่อตัวแปรให้ยาวเท่าไหรก็ได้แต่ไม่เกิน 31 ตัวอักษร ตามปรกติแล้ว เราควรจะใช้ชื่อตัวแปรที่ไม่ยาวจนเกินไป และควรจะให้มีความหมายในตัวของมันเองด้วย เช่น เราต้องการจะหาค่าพื้นที่ของวงกลม เราก็สามารถใช้ชื่อ ตัวแปรดังต่อไปนี้

```
float  PI;     /* PI = 3.141592....      */
float  radius; /* the radius of a circle */
float  area;   /* the area of a circle   */
```

ตัวอย่างของชื่อตัวแปรที่ถูกต้อง เช่น

```
int     variable_name;
float   x;
char    ch1;
char    Ch1;
```

คำเตือน ในภาษาซีมีการจำแนกความแตกต่างระหว่างตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก

ในกรณีที่ตัวแปรหลายๆตัวเก็บค่าของข้อมูลที่เป็นชนิดเดียวกันเราก็สามารถเขียนให้อยู่ในรูปต่อไปนี้ (แต่ไม่จำเป็นเสมอไป) เพื่อประหยัดเนื้อที่

```
data_type   variable1, variable2,..., variableN;
```

ตัวอย่างเช่น

```
float   PI, radius, area;
```

1.5.3 ตัวอักษรในภาษาซี (Character)

การศึกษาโครงสร้างของภาษาซีก็คล้ายกับการทำความเข้าใจโครงสร้างของภาษาของมนุษย์ เช่น ภาษาอังกฤษ ในแต่ละภาษาย่อมมีตัวอักษรซึ่งเป็นหน่วยย่อยหรือส่วนที่เล็กที่สุดของภาษา ในภาษาซีก็เช่นเดียวกัน เราใช้ตัวอักษรที่เรียกว่า ตัวอักษรแบบ ASCII ซึ่งเลขรหัสของตัวอักขระเหล่านี้มีค่าอยู่ระหว่าง 0 ถึง 255 แต่เราจะใช้เพียงส่วนหนึ่งเท่านั้นในการนำมาเขียนเป็นชุดคำสั่งในภาษาซี

ในภาษาอังกฤษ ถ้าเราเขียนคำต่างๆ ติดกันโดยไม่เว้นช่องว่างระหว่างคำ ก็จะเป็นการยากที่จะอ่านข้อความเหล่านั้นให้เข้าใจได้อย่างรวดเร็วว่า มันมีความหมายอย่างไร ในภาษาซีคอมไพล์เลอร์ก็ต้องการการแยกคำแยกประโยคเพื่อที่จะสามารถเข้าใจความหมายของโปรแกรมที่เราเขียนขึ้นได้อย่างถูกต้อง การแยกคำในภาษาซีจะใช้สัญลักษณ์หลายๆตัว ซึ่งเรารวมเรียกว่า White Spaces ประกอบไปด้วยสัญลักษณ์ ที่บ่งบอกถึงการเว้นช่องว่างระหว่างตัวอักษร ซึ่งก็คือ Blank และ Space การขึ้นบรรทัดใหม่ (Linefeed และ Carriage Return) ก็จัดเป็นการแยกคำในภาษาซี

เมื่อเรานำตัวอักษรซึ่งอยู่ในกลุ่มของตัวอักษรตั้งแต่ A ถึง Z ทั้งตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก ตัว เลขโดดตั้งแต่ 0 ถึง 9 หรือ ตัวอักษรอื่นๆที่เราสามารถกดได้โดยตรงจากแป้นพิมพ์ เช่น

```
!  #  ^  &  *  (  )  _  \  -  |  +  =  {  }  [  ]
:  ;  "  '  ~  <  >  .  /
```

เข้ามาประกอบกันขึ้น เรียกว่า โทเคน (Token) จะมีความหมายหรือไม่มีก็ตาม ตัวอย่างของโทเคนที่สำคัญ ก็คือ ตัวระบุชื่อ (Identifier) ไม่ว่าจะเป็นชื่อของตัวแปรที่เราใช้ในโปรแกรม ยกตัวอย่างเช่น `x, y, radius` หรืออาจจะเป็นชื่อของฟังก์ชัน เช่น `printf, main, add` เหล่านี้เป็นต้น หลายๆโทเคนที่มีความหมายเมื่อนำมาประกอบเข้าด้วยกันก็จะเป็นนิพจน์ (Expression) หรือประโยค (Statement) และ ฟังก์ชัน จนเป็นโปรแกรมที่สมบูรณ์ในที่สุด

1.5.4 ตัวระบุชื่อ (Identifier)

ตัวบอกชื่อที่เราใช้เรียกชื่อตัวแปรหรือฟังก์ชันจะต้องไม่มีสัญลักษณ์ที่จัดว่าเป็นโอเปอร์เรเตอร์หรือ ตัวดำเนินการในภาษาซี ตัวอย่างเช่น ถ้าเราต้องการตั้งชื่อตัวแปรว่า ***name** ในที่นี้คอมไพล์เลอร์จะเข้าใจว่า ตัวแปรมีชื่อว่า **name** และมิใช่ ***name** ตามที่เราต้องการจะตั้งชื่อให้ตัวแปร และเครื่องหมายดอกจันทร์ทำให้คอมไพล์เลอร์ในภาษาซีจะเข้าใจว่า เราต้องการใช้ตัวแปรชื่อ **name** ซึ่งทำหน้าที่เป็นพอยน์เตอร์ (Pointer) หรือ ตัวชี้

พอยน์เตอร์ คืออะไรเราจะได้เรียนต่อไปอย่างละเอียด เพราะการใช้พอยน์เตอร์ในภาษาซีถือว่าเป็นเรื่องที่สำคัญมาก โดยเฉพาะในบางสถานการณ์ เราสามารถใช้พอยน์เตอร์ในการเพิ่มประสิทธิภาพการทำงานของโปรแกรมได้ หรือสำหรับโครงสร้างของข้อมูล (Data Structure) บางชนิดเราจำเป็นต้องใช้พอยน์เตอร์เท่านั้นในการกำหนดขึ้นมาใช้งาน

นอกจากนี้ ยังมีคำที่เราห้ามนำมาใช้ตั้งชื่อ คำพวกนี้เรียกว่า C Keywords เป็นคำที่สงวนไว้ตาม มาตราฐานของ ANSI เพราะสำหรับแต่ละตัว ได้มีการกำหนดความหมายและหน้าที่ของมันไว้แล้วอย่าง ชัดเจนและไม่สามารถเปลี่ยนแปลงได้

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

1.5.5 นิพจน์ (Expression)

ในภาษาที่เราใช้สื่อสารกัน คำคือหน่วยที่เล็กที่สุดของประโยคที่มีความหมาย ในภาษาซีก็เช่นเดียวกัน มีองค์ประกอบของประโยคที่เรียกว่า นิพจน์ หรือ Expression แบ่งออกเป็นสี่จำพวก

นิพจน์ ในภาษาซี

1) นิพจน์ที่เป็นค่าคงที่ (Constant Expressions)
2) นิพจน์ที่เป็นตัวแปร (Variable Expressions)
3) นิพจน์ที่ประกอบขึ้นจากนิพจน์สองแบบแรก (Complex Expressions) โดยอาศัยตัวดำเนินการ หรือโอเปอร์เรเตอร์
4) นิพจน์ที่เกิดจากการเรียกใช้ฟังก์ชัน (Function Calls)

1) นิพจน์ที่เป็นค่าคงที่ (Constant Expressions) เช่น

```
12
143.12
'A'
"Hello World"
```

2) นิพจน์ที่เป็นตัวแปร (Variable Expressions) เช่น

```
x, y, sum, radius
```

3) นิพจน์ที่ประกอบขึ้นจากนิพจน์สองแบบแรก เรียกว่า Complex Expression โดยมีสิ่งที่เรียกว่าโอเปอร์เรเตอร์ (Operator) เช่น เครื่องหมายบวกลบคูณหาร เครื่องหมายมากกว่าน้อยกว่า หรือเท่ากับ วงเล็บเปิดและวงเล็บปิด และอื่นๆอีกมาก ซึ่งเราจะได้ทำความรู้จักในบทต่อๆไป ตัวอย่างของประพจน์ชนิดนี้ เช่น

```
sum = x + y
 x  = (y - 5 ) * 2
```

โปรดสังเกตว่า เครื่องหมายเท่ากับในภาษาซีไม่ได้มีความหมายตามที่เราเข้าใจในวิชาคณิตศาสตร์ เช่น สมมุติว่า เราเขียนสมการทางคณิตศาสตร์ดังต่อไปนี้

```
y = x + 2
```

ถ้าเรารู้ว่าตัวแปร x และตัวแปร y มีค่าเท่าไร เราก็เรียกว่าประพจน์ เพราะเรารู้ว่าค่าที่อยู่ทางซ้ายมือเท่ากับ ค่าที่อยู่ทางขวามือหรือไม่ แต่ในภาษาซีมันมีความหมายที่แตกต่างออกไป นิพจน์ข้างล่างนี้ที่มีเครื่องหมายอัฒภาคอยู่ข้างท้าย (เราจะไม่เรียกมันว่าสมการอีก เพราะมันไม่ได้มีความหมายของสมการดังที่กล่าวไป)

```
y = x + 2;
```

มีความหมายว่า ค่าของตัวแปร y จะมีค่าเท่ากับ ค่าของตัวแปร x ในขณะนั้น บวกด้วยสองโดยไม่คำนึงถึงว่า ค่าของตัวแปร y ก่อนหน้านี้จะเป็นเท่าไรและผลของการกระทำก็คือตัวแปร y จะมีค่า

ใหม่ และเก็บค่านี้ไว้ในตัวของมันจนกว่า เราจะให้ค่าใหม่ เหมือนกับว่าเรามีเมมโมรี่ที่ใช้เก็บข้อมูลอยู่ เมื่อต้องการจะเก็บข้อมูลใหม่ เราก็เขียนข้อมูลใหม่ทับข้อมูลเก่าในเมมโมรี่

ดังที่กล่าวไปแล้วเครื่องหมายเท่ากับ จัดเป็นโอเปอร์เรเตอร์ที่มี ชื่อว่า Assignment Operator คือเป็นการผ่านค่าให้ตัวแปร เพราะสำหรับตัวแปรแต่ละตัวที่เราใช้ในโปรแกรม เราก็ต้องการให้มันเก็บค่าใดๆที่เราต้องการและบางครั้งเราก็ต้องการให้มันเก็บค่าใหม่ได้ด้วย ในภาษาปาสคาลจะใช้สัญลักษณ์ := และสิ่งที่จะลืมเสียไม่ได้ก็คือ ตัวแปรเท่านั้นมีสามารถอยู่ทางซ้ายมือของเครื่องหมายเท่ากับได้ ตัวอย่างที่ไม่ถูกต้องเช่น

```
1 = 1;
x + 2 = 2;
```

ในตัวอย่างข้างบน เลขหนึ่งเป็นนิพจน์ที่มีค่าคงที่ และ x + 2 เป็นนิพจน์ที่ให้ค่าคงที่คือ ค่าของ x บวกด้วย 2 ดังนั้นเราไม่อาจจะเปลี่ยนแปลงค่าของนิพจน์ทั้งสองได้

4) นิพจน์ที่เกิดจากการเรียกใช้ฟังก์ชัน (Function Calls) เราได้ทำความรู้จักกับการทำงานของโปรแกรมในภาษาซีที่แบ่งออกเป็นฟังก์ชันแล้วในตอนต้นแล้ว สมมุติว่า เราได้กำหนดฟังก์ชันชื่อ add ขึ้นมาใช้ในตัวอย่างข้างล่างนี้ สำหรับการบวกเลขสองจำนวนและเรารู้ว่าเราจะเขียนฟังก์ชันขึ้นมาใช้เองได้อย่างไร

```
/* 1 */     int x;
/* 2 */     x = add(543, 1991);
/* 3 */     if (add(x, 101) > 2520) {
/* 4 */           x = 2520;
/* 5 */     }
```

บรรทัดแรกเป็นการแจ้งให้คอมไพล์เลอร์ทราบว่า เราจะใช้ตัวแปร x เก็บข้อมูลที่เป็นจำนวนเต็มแบบ int ในบรรทัดที่สองเป็นการกำหนด ค่าตัวแปร x โดยให้มีค่าเป็นผลรวมของเลขสองจำนวน คือ 543 และ 1991 ซึ่งเราใช้ฟังก์ชัน add() สำหรับการบวกเลข เรากำหนดตัวเลข 543 และ 1991 ให้เป็นข้อมูลที่ฟังก์ชันนำไปใช้ในการบวกเลข หรือบางครั้งเราก็เรียกว่า Function Arguments หรือ Function Parameters ในตัวอย่างนี้ add(543, 1991) และ add(x, 101) จัดเป็นนิพจน์ที่เกิดจากการเรียกใช้ฟังก์ชัน

1.5.6 ประโยคในภาษาซี (Statement)

ขั้นตอนการทำงานแต่ละขั้นในภาษาซีหรือเราอาจจะเรียกว่า ประโยคก็ได้ ในภาษาอังกฤษใช้คำ ว่า Statement โดยแบ่งออกเป็น สองจำพวกคือ ประโยคเดี่ยว (Simple Statement) และ ประโยคซับซ้อน (Compound Statement) สำหรับประโยคเดี่ยวเราสามารถสังเกตได้จากนิพจน์ที่มีเครื่อง หมายอัฒภาค (Semicolon) อยู่ข้างท้าย เช่น `x = add(18,1921);` ส่วนประโยคซับซ้อนก็คือกลุ่มของประโยคเดี่ยว หลายๆ ประโยคที่อยู่ระหว่างเครื่องหมายปีกกา โดยเริ่มต้นด้วย `{` และจบด้วย `}` และคอมไพล์เลอร์ก็จะมองประโยคซับซ้อนนี้ราวกับว่ามันเป็นประโยคเดี่ยว

1.5.7 ฟังก์ชันในภาษาซี (Function)

ภาษาซีเป็นภาษาที่มีขนาดเล็ก ซึ่งหมายถึงตัวภาษาซีเองประกอบด้วยหลักไวยกรณ์ที่จำเป็นเท่านั้น ไม่รวมฟังก์ชันต่างๆ ไม่ว่าจะเป็นฟังก์ชันมาตราฐาน (Standard Function) หรือฟังก์ชันที่เราเขียนเพิ่มเติมขึ้น ไม่เหมือนกับในภาษาระดับสูงอื่นๆที่มีฟังก์ชันมาตราฐานหรือที่เราเรียกว่า Built-in Function เป็นส่วนหนึ่งของตัวภาษาสำหรับทำหน้าที่พื้นฐานต่างๆ ซึ่งกำหนดมาอย่างแน่ชัด เช่น สำหรับการอ่านข้อมูลหรือพิมพ์ข้อมูลออกทางจอภาพ เป็นต้น

ในภาษาซี แม้ว่าตัวภาษาซีเองจะไม่รวมฟังก์ชันพื้นฐานต่างๆดังที่ยกตัวอย่างไป และเราจะต้องเขียนฟังก์ชันขึ้นมาใช้เองทั้งหมด แต่ถ้าฟังก์ชันใดเราใช้บ่อยเราก็สามารถกำหนดให้เป็นฟังก์ชันมาตราฐานได้ ฟังก์ชันมาตราฐานเหล่านี้จะถูกเก็บรวบรวมอยู่ในไฟล์ที่ถูกคอมไพล์เสร็จแล้ว เป็นเสมือนกับคลังของฟังก์ชัน (Function Libraries) เมื่อเราต้องการใช้ฟังก์ชันเหล่านี้เราก็เรียกมาใช้ได้เลยโดยไม่ต้องมากำหนดรูปแบบสร้างมันขึ้นมาใหม่ เช่น ฟังก์ชัน `printf()` เป็นฟังก์ชันที่เราใช้ในการพิมพ์ข้อ ความตัวเลข หรือข้อมูลต่างๆออกทางจอภาพ

แต่ก่อนที่เราจะใช้ฟังก์ชันมาตราฐานใดๆในโปรแกรม เราก็ต้องแจ้งรูปแบบของฟังก์ชันก่อนว่า มีรูปร่างหน้าตาอย่างไร เช่น ชื่อของฟังก์ชัน ค่าพารามิเตอร์ที่ฟังก์ชันต้องการ ฟังก์ชันให้ค่าใดกลับคืนออกมาหรือไม่หลังจากที่จบการทำงานของฟังก์ชัน เรารวมเรียกมันว่า ส่วนหัวของฟังก์ชัน ข้อมูลนี้จะเป็นประโยชน์ต่อคอมไพล์เลอร์ คล้ายๆเป็นการบอกให้คอมไพล์เลอร์ทราบว่าฟังก์ชันที่เราแจ้งนี้ได้ถูกสร้างขึ้นมาแล้ว และเราต้องการใช้ฟังก์ชันนี้ในโปรแกรม

ส่วนใหญ่แล้ว เรามักจะเรียกใช้ฟังก์ชันมาตราฐานหลายๆตัวในโปรแกรม ดังนั้นเพื่อเป็นการประหยัดการเขียนประโยคคำสั่งแจ้งการใช้ฟังก์ชันมาตราฐานหลายๆตัว เราสามารถแทรกไฟล์หรือแฟ้มข้อมูล ที่รวบรวมส่วนหัวของฟังก์ชันมาตราฐานเหล่านี้เอาไว้ในคราวเดียวกัน ไฟล์ชนิดนี้

เราเรียกว่า Header File มักจะมีส่วนขยายของชื่อไฟล์ที่ลงท้ายด้วย .h ทำให้เราสังเกตได้ง่าย เช่น stdio.h stdlib.h

วิธีการง่ายๆ ในการแทรกเนื้อหาข้อความของไฟล์อื่นเข้าในไฟล์ที่เป็นโปรแกรมโค้ดหลักของเรา ก็คือ การใช้คำสั่งที่เรียกว่า #include จัดเป็นคำสั่งจำพวกพรีโปรเซสเซอร์ไดเรคทีฟ (Preprocessor Directives) คือ จะมีผลในตอนที่เราเริ่มทำการคอมไพล์โปรแกรมโค้ด ขั้นตอนแรกสุดของการคอมไพล์โปรแกรมโค้ดก็คือ คอมไพล์เลอร์ที่เราใช้จะเรียกส่วนที่เรียกว่า พรีโปรเซสเซอร์ ขึ้นมาและซึ่งมีหน้าที่ตรวจดูว่า ในโปรแกรมโค้ดของเรามีคำสั่งของพรีโปรเซสเซอร์หรือไม่ ถ้ามีพรีโปรเซสเซอร์ไดเรคทีฟในบรรทัดใด พรีโปรเซสเซอร์ก็จะตีความคำสั่งเหล่านั้นว่า จะต้องทำอะไรบ้างก่อนที่จะผ่านโปรแกรมโค้ดที่ถูกดัดแปลงแก้ไขแล้วไปยังคอมไพล์เลอร์อีกที เราจะได้เรียกรู้คำสั่งที่สำคัญของ พรีโปรเซสเซอร์ ในบทต่อๆไป

ดังที่กล่าวไปแล้วในข้างตน ภาษาซีเป็นภาษาของโปรแกรมคอมพิวเตอร์ที่เน้นการเรียกใช้ฟังก์ชัน (Function Call) เป็นหลัก ไม่ว่าจะเป็นฟังก์ชันมาตราฐานหรือฟังก์ชันตัวอื่นๆ ที่เราสร้างขึ้นมาใหม่ คุณสมบัติในข้อนี้ทำให้ภาษาซีจัดอยู่ในกลุ่มของภาษาที่เรียกว่า Modular Programming Language ซึ่งหมายความว่า โปรแกรมโค้ดในภาษาซีจะต้องประกอบขึ้นมาจากหน่วยของฟังก์ชัน

1.5.8 ตัวอย่างการสร้างฟังก์ชัน

ในการเขียนโปรแกรมภาษาซี เรามักจะเกี่ยวข้องกับการเรียกใช้ฟังก์ชันมาตราฐาน เช่น printf() เป็นต้น นอกจากนี้ในบางกรณีเราก็ต้องนิยามและสร้างฟังก์ชันขึ้นมาใช้เองเพื่อจุดประสงค์ต่างๆ เราลองมาดูตัวอย่างการสร้างฟังก์ชันแบบง่าย คือ ฟังก์ชัน add() ที่เราใช้บวกเลขแบบ int สองจำนวน

`int`	ฟังก์ชันนี้ให้ค่าแบบ int
`add`	ชื่อของฟังก์ชันคือ add
`(int a, int b)`	แบบของพารามิเตอร์ที่ต้องผ่านให้ฟังก์ชันเมื่อเรียก
`{`	จุดเริ่มต้นการทำงานของฟังก์ชัน
`int sum;`	แจ้งการใช้ตัวแปร sum ซึ่งใช้ภายในฟังก์ชันเท่านั้น
`sum = a + b;`	หาผลรวมของค่าพารามิเตอร์ a และ b
`return (sum);`	ผ่านค่าผลลัพธ์ของฟังก์ชัน
`}`	จุดจบการทำงานของฟังก์ชัน

โปรแกรมข้างล่างนี้เป็นตัวอย่างการเรียกใช้ฟังก์ชัน add() ที่เราได้นิยามขึ้นตามตารางข้างบน

```
/* 1 */   #include <stdio.h>
/* 2 */
/* 3 */   int main ()
/* 4 */   {
/* 5 */      int  x, y;       /* Variable Declarations */
/* 6 */
/* 7 */      x = 1091;
/* 8 */      y = 1432;
/* 9 */      printf ("%d + %d = %d\n", x, y, add(x,y));
/*10 */      printf ("%d + %d = %d\n", 1091, 1432, add(1091,1432));
/*11 */      printf ("%d + %d = %d\n", x, 1432, add(x,1432));
/*12 */      return 0;
/*13 */   }
```

ในบรรทัดแรกเป็นการใช้พรีโปรเซสเซอร์ไดเรคทีฟ #include เพื่อแทรกไฟล์ชื่อ stdio.h ภายในไฟล์ นี้จะมีส่วนหัวของฟังก์ชันมาตราฐาน printf() ที่เราต้องการใช้ภายในโปรแกรม ไฟล์ stdio.h มีอยู่ ในไดเรคทรอรี่ใดนั้นขึ้นอยู่กับชนิดของคอมไพล์เลอร์ที่ใช้ เช่น ในระบบปฏิบัติการแบบ UNIX ไฟล์นี้จะ เก็บไว้ที่ /usr/include/stdio.h หรือ สมมุติว่าใช้คอมไพล์เลอร์ของ Microsoft Borland C++ เวอร์ชัน 4.0 สำหรับวินโดวส์ (เราสามารถใช้คอมไพล์เลอร์สำหรับซีพลัซพลัซในการคอมไพล์โปรแกรมภาษาซีได้ แต่อาจจะมีความแตกต่างจากภาษาซีที่เขียนตามมาตราฐานของ ANSI บ้างในบางจุด) ก็อาจจะเก็บไว้ที่ C:\BC4\INCLUDE\STDIO.H

เนื่องจากว่าเราใช้คอมพิวเตอร์และคอมไพล์เลอร์ที่แตกต่างกัน ไฟล์ stdio.h จึงเก็บไว้ในที่แตกต่างกัน เพื่อตัดปัญหานี้เราก็เขียนย่อๆว่า #include <stdio.h> โดยไม่รวมชื่อของไดเรคทรอรี่ เวลาทำการคอมไพล์ พรีโปรเซสเซอร์ซึ่งเป็นส่วนหนึ่งของคอมไพล์เลอร์ ก็จะค้นหาไฟล์ดังกล่าว ในไดเรคทรอรี่มาตราฐานของคอมไพล์เลอร์ เช่น /usr/include/ หรือสำหรับ Borland C++ ซอฟต์แวร์อาจจะติดตั้งไว้ที่ C:\BC4 ดังนั้นคอมไพล์เลอร์ของ Borland C++ ก็จะค้นหาไฟล์ stdio.h ที่ C:\BC4\INCLUDE\ ถ้าเราต้องการเจาะจงว่าไฟล์ stdio.h อยู่ที่ใดเราก็สามารถเติมชื่อไดเรคทรอรี่ข้างหน้าชื่อไฟล์ได้ เช่นแทนที่จะเป็น #include <stdio.h> เราก็เขียนใหม่เป็น

```
#include "/usr/inclide/stdio.h"
#include "C:\BC4\INCLUDE\STDIO.H"
```

สำหรับระบบ UNIX และ Borland C++ ตามลำดับ หรือบางคนใช้คอมไพล์เลอร์ Turbo C++ เวอร์ชัน 2.0 สำหรับดอส เช่น สมมุติว่าติดตั้งซอฟต์แวร์ไว้ที่ D:\TC\ ในลักษณะเดียวกันถ้าเราต้องการจะใส่ชื่อไดเรคทรอรี่ เราก็สามารถเขียนได้ดังต่อไปนี้

```
#include "D:\TC\INCLUDE\STDIO.H"
```

โปรแกรมนี้ยังเป็นโปรแกรมที่สมบรูณ์แบบตามหลักไวยกรณ์ของภาษาซี ผู้อ่านสามารถพิมพ์ลงในไฟล์ เช่น ให้ชื่อว่า `program1.c` และ ลองทำการคอมไพล์โปรแกรมโค้ดที่อยู่ในไฟล์นี้

ผลจากโปรแกรมคือ

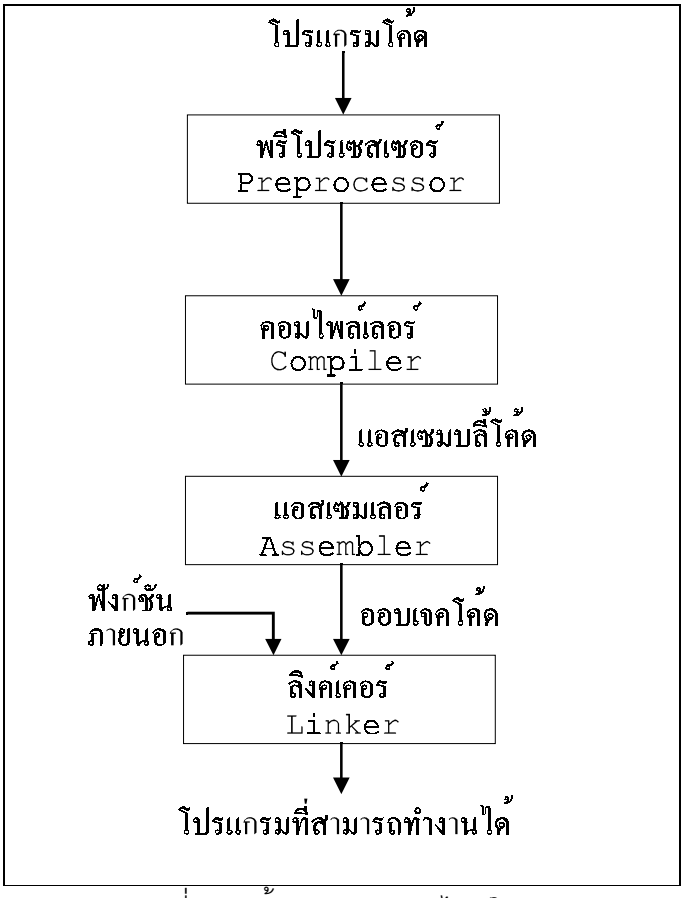
```
1091 + 1432 = 2523
1091 + 1432 = 2523
1091 + 1432 = 2523
```

จะเห็นได้ว่าขั้นตอนการทำงานในบรรทัดที่ 9 10 และ 11 ให้ผลออกทางจอภาพเหมือนกัน โปรดสังเกตว่า เวลาเราเรียกใช้ฟังก์ชัน `add()` เราสามารถผ่านค่าตัวแปรหรือค่าคงที่แบบ `int` ก็ได้ (ตัวเลขจำนวนเต็มใน โปรแกรมที่มีค่าอยู่ระหว่าง -32768 และ 32767 จะถือว่าเป็นค่าคงที่แบบ `int`) นอกจากนี้เรายังสามารถ ผ่านค่าของฟังก์ชันแบบ `int` ให้เป็นพารามิเตอร์ของฟังก์ชันอื่นก็ได้

ฟังก์ชัน `add()`จะให้ค่าที่ผิดเมื่อผลลัพธ์ของเลขจำนวนเต็มแบบ `int` สองตัวมีค่าอยู่นอกขอบเขต ของข้อมูลแบบ `int` ยกตัวอย่างเช่น `add(32767, 1)` หรือ `add(-20000,-20000)` แม้ว่าค่าของพารามิเตอร์ของฟังก์ชันทั้งหมดจะอยู่ในขอบเขตที่ถูกต้อง แต่ฟังก์ชันก็ให้ค่าที่ผิด เพราะฉะนั้นเวลาเราจะเรียกใช้ฟังก์ชันใดๆก็ควรจะรู้ข้อจำกัดหรือเงื่อนไขของฟังก์ชัน หรืออีกทางหนึ่งก็คือ ถ้าเราต้องสร้างฟังก์ชันขึ้นใช้เอง เราก็สามารถเพิ่มขั้นตอนการทำงานที่มีหน้าที่ตรวจสอบเงื่อนไขของฟังก์ชัน ถ้าข้อมูลที่ผ่านให้ฟังก์ชันไม่เป็นตามเงื่อนไขของฟังก์ชัน ก็อาจจะเตือนให้ผู้ใช้ทราบเวลาโปรแกรมทำงานโดยพิมพ์ข้อความแจ้งความผิดพลาดออกทางจอภาพ หรืออาจจะหยุดการทำงานของโปรแกรมก็ได้

1.5.9 ขั้นตอนการสร้างโปรแกรมที่สามารถทำงานได้

เมื่อเราเขียนโปรแกรมโค้ดเสร็จแล้วและเราต้องการทดลองดูว่า มันจะทำงานได้ตามที่กำหนดเอาไว้หรือไม่ เราก็ต้องทำการคอมไพล์โปรแกรมโค้ดก่อนโดยแปลงให้เป็นภาษาเครื่องที่เราสามารถรันโปรแกรมนี้ได้ ขั้นตอนของการคอมไพล์โปรแกรมโค้ดแบ่งออกเป็นหลายขั้นตอน โปรดดูรูปภาพที่ 1.2



รูปภาพที่ 1.2 ขั้นตอนการคอมไพล์โปรแกรม

สำหรับคอมพิวเตอร์ที่มีระบบปฏิบัติการแบบ UNIX เราก็มักจะใช้คำสั่งชื่อ cc หรือ gcc ซึ่งเป็นคำสั่งที่ ใช้คอมไพล์โปรแกรมโค้ด เมื่อรันคำสั่งแล้วส่วนที่เรียกว่า *พรีโปรเซสเซอร์* ก็จะอ่านโปรแกรมโค้ดและตรวจดูว่า มีบรรทัดไหนที่จะต้องแก้ไขก่อนหรือไม่ เช่น แทรกไฟล์อื่นเข้าในโปรแกรมโค้ดเพื่อใช้ประกอบในการคอมไพล์ เมื่อทำการแก้ไขแล้วพรีโปรเซสเซอร์ก็จะผ่านข้อมูลต่อไปยังตัวคอมไพล์เลอร์เพื่อแปลงเป็นโค้ดในภาษาระดับต่ำลงไป คือคล้ายๆภาษาแอสเซม บลี้ จนกลายเป็นโค้ดในภาษาเครื่องที่เรียก ว่า *ออบเจคโค้ด* (Object Code) โดยบันทึกลงในไฟล์ที่มีส่วนขยายของชื่อไฟล์ที่ลงท้าย ด้วย .o เมื่อสร้างออบเจคโค้ดแล้ว ขั้นต่อไปก็คือการค้นหาส่วนของฟังก์ชันมาตราฐานหรือฟังก์ชันภายนอกที่เราเรียกใช้ในโปรแกรมโค้ดของเรา โดยหาจากไฟล์ที่เป็นคลังรวบรวมฟังก์ชันมาตราฐานเหล่านั้น เมื่อค้นพบส่วนของฟังก์ชันเหล่านั้นแล้วก็จะนำมาประกอบเข้ากับออบเจคโค้ด และสร้างไฟล์ที่เก็บรวมคำสั่งขั้นตอนการทำงานทั้งหมดของโปรแกรม ซึ่งเราสามารถรันได้เหมือนกับไฟล์ของดอสที่ลงท้ายด้วย .EXE

1.5.10 สไตล์การเขียนโปรแกรมโค้ดในภาษาซี

ในหัวข้อสุดท้ายของบทนี้ เราลองมาพิจารณาสองตัวอย่างดังต่อไปนี้ ซึ่งเป็นตัวอย่างของโปรแกรมง่ายๆในภาษาซี

แบบที่หนึ่ง

```
void main()
{  int x; x=10;
   if(x < 5){ x = 0;} else { x = 1; }
}
```

แบบที่สอง

```
void main ()
{
      int x;

      x = 10;
      if (x < 5)
      {
            x = 0;
      }
      else
      {
            x = 1;
      }
}
```

เราจะเห็นได้ว่ารูปแบบในการเขียนโปรแกรมแบบที่สองดูดีกว่า และอ่านได้ง่ายดูเป็นระเบียบเรียบร้อยกว่า แบบแรก แม้ว่าทั้งสองแบบเขียนได้ถูกต้องตามหลักไวยกรณ์ในภาษาซี ส่วนแบบแรกก็มีข้อดีคือประหยัดเนื้อที่ อย่างไรก็ตามสิ่งนี้ก็เป็นเรื่องความชอบของแต่ละบุคคล และโปรแกรมเมอร์แต่ละคนก็มีสไตล์การเขียนที่แตกต่างกัน ก็ให้ผู้อ่านตัดสินใจด้วยตนเองว่าชอบแบบไหน และจะเลือกเขียนโปรแกรมของตนในสไตล์ใด

แบบฝึกหัดท้ายบท

1. จงตอบคำถามต่อไปนี้

1.1) อะไรคือโปรแกรมคอมพิวเตอร์

1.2) มีอะไรบ้างที่เป็นองค์ประกอบสำคัญของโปรแกรมในภาษาซี

1.3) จงอธิบายความแตกต่างระหว่างโปรแกรมโค้ด และ ออบเจ็คโค้ด

1.4) อะไรคือคอมไพล์เลอร์ และมีหน้าที่อย่างไรในการสร้างโปรแกรม

1.5) ฟังก์ชันในภาษาซีมีองค์ประกอบที่สำคัญใดบ้าง

2. คำใดต่อไปนี้ที่เราสามารถใช้เป็นชื่อของตัวแปรในโปรแกรมภาษาซีได้

```
distance        baby            main            int
_MS_DOS_        time-of-day     0x9b            x2y
do              ball%           long            DATA_0
```

3. โปรแกรมที่มีขนาดเล็กที่สุดในภาษาซีคือ

```
void main(void) {}
```

ลองพิมพ์โปรแกรมนี้ลงในไฟล์และทำการคอมไพล์โปรแกรม

4. จงอธิบายความแตกต่างระหว่างโปรแกรมในแบบฝึกหัดข้อที่ 3 และโปรแกรมโค้ดข้างล่างนี้

4.1)

```
int main ()
{
      return 0;
}
```

4.2)

```
void main (void)
{
      return ;
}
```

5. จงพิมพ์และคอมไพล์โปรแกรมต่อไปนี้ ซึ่งมีอยู่จุดหนึ่งที่ผิดอยู่ ลองดูว่าคอมไพล์เลอร์ที่ใช้จะแจ้งข้อผิดพลาดว่าอย่างไร

```
#include <stdio.h>

void main()
{
      int x, y;
      /* This program has a syntax error ....
      x = 10;
      y = x;
```

```
        printf ("%d \n", x + y) ;
        return;
}
```

6. ประโยคคำสั่งต่อไปนี้มีผลอย่างไรในโปรแกรม

```
10000;
1.-1.000;
"ABC";
'm';
(0);
```

7. จงให้เหตุผลว่า ทำไมประโยคคำสั่งต่อไปนี้เป็นประโยคคำสั่งที่ผิด โดยเรากำหนดให้ตัวแปร x เป็นตัวแปรแบบ int

```
(1 = 1);
x + 1 = 1000;
x*x   = 100;
```

8. จงหาค่าของตัวแปรต่างๆของฟังก์ชันหลักในแต่ละบรรทัด ตั้งแต่บรรทัดที่ 4 ถึงบรรทัดที่ 7

```
/* 1 */     int main()
/* 2 */     {
/* 3 */           int  x, y, z;
/* 4 */           x = y = 0;
/* 5 */           z = -1;
/* 6 */           x = y = y - z;
/* 7 */           x = 5 - (x+y-z);
/* 8 */           return 0;
/* 9 */     }
```

9. จงคำนวณหาค่าที่เกิดจากการเรียกใช้ฟังก์ชันต่อไปนี้

9.1)

```
add( add( add( add( add(1,2), 3), 4), 5), 6)
```

9.2)

```
add( add( add(1,2), add(3,4)), add(5,6))
```