

บทที่ 2 พื้นฐานภาษา C++

ภาษา C++ (อ่านว่า ซี-พลัส-พลัส) เป็นลูกผสมระหว่างภาษา Simula และภาษา C โดยรับเอาแนวคิดของภาษา C มากกว่า 95% ประยุกต์เข้ากับแนวคิดเชิงวัตถุของ Simula ทำให้ภาษา C++ เป็นลูกผสมระหว่าง Procedural Language และ Object Oriented Language เราไม่สามารถบอกได้ว่า C++ เป็น OOP100% โดยเราอาจเลือกเขียนแบบภาษา C ได้อีกแบบหนึ่ง

ภาษา C++ เป็นภาษาโปรแกรมภาษาหนึ่งที่ได้รับการพัฒนาขึ้นมาไม่นานนักและเป็นภาษาที่มีความสามารถสูง ดังนั้น ในบทนี้จะเสนอความเป็นมาของภาษา C++ รูปแบบการเขียนโปรแกรมขั้นต้นเพื่อเรียนรู้ถึงองค์ประกอบต่างๆ ที่จำเป็นในการเขียนโปรแกรมด้วยภาษา C++ เช่น การเขียนคอมเมนต์ การประกาศตัวแปร ชนิดข้อมูลที่ควรรู้ รวมทั้งข้อสังเกตที่น่าสนใจในภาษา C++ เช่น ตัวดำเนินการต่างๆ การจัดการกับการเกิดส่วนล้น (Overflow) เป็นต้น

ความเป็นมาของภาษา C++

C++ มีรากฐานมาจากภาษา C และเป็นภาษาที่คลุมภาษา C ไว้ C++ ยังคงรักษาความสามารถและความยืดหยุ่นของ C ในการเขียนโปรแกรมระบบต่ำ รวมทั้งโปรแกรมควบคุมฮาร์ดแวร์ ที่สำคัญกว่านั้น คือ C++ ให้การสนับสนุนการเขียนโปรแกรมแบบ Object-Oriented C++ จัดเป็นภาษาที่มีความสามารถมากกว่า ADA และ Modula-2 ขณะที่ยังคงความมีประสิทธิภาพและความกะทัดรัดของภาษา C ไว้ ดังนั้น จึงเป็นภาษาโปรแกรมภาษาหนึ่งที่ยอมให้โปรแกรมเมอร์เขียนโปรแกรมแบบมีโครงสร้าง และเขียนโปรแกรมเชิงวัตถุได้อย่างมีประสิทธิภาพ

C++ เป็นภาษาผสม (Hybrid Language) โดยอาจแก้ปัญหาหนึ่งด้วยวิธี Object-Oriented ล้วนๆ หรืออาจแก้ปัญหาด้วยการใช้ภาษาแบบเก่า ซึ่งมีโครงสร้างบางอย่างเพิ่มขึ้นจากภาษา C ในทางปฏิบัติในการแก้ปัญหามักจะสะท้อนให้เห็นวิธีการทั้ง 2 แบบ

C++ ถูกพัฒนาโดย Bjarne Stroustrup ที่ Bell Labs ในช่วงทศวรรษ 1980 Dr.Stroustrup พัฒนาภาษานี้ขึ้นเพื่อเขียนซอฟต์แวร์จำลองเหตุการณ์ (Event-Driven Simulation) ที่มีความซับซ้อน ซึ่งมี Rick Mascitti เป็นผู้ตั้งชื่อของภาษานี้ให้กับเขา

C++ ถูกออกแบบให้ส่งเสริมการพัฒนาซอฟต์แวร์ขนาดใหญ่ โดยเพิ่มการตรวจสอบ Type เข้าไป เมื่อเปรียบเทียบกับ C แล้วจะลดข้อผิดพลาดลงได้มาก เพราะภาษา C ยอมให้โปรแกรมเมอร์ควบคุมระบบในระดับต่ำได้โดยตรง โปรแกรมเมอร์จำนวนมากจึงทำงานโดยเริ่มจากโครงสร้างระดับต่ำ แล้วนำส่วนต่างๆ เหล่านี้มาประกอบกันเป็นโครงสร้างใหญ่ แต่ในภาษา C++ จะทำในทางตรงกันข้าม คือ กำหนดโครงสร้างใหญ่ก่อนนำมาสัมพันธ์กัน แล้วจึงกำหนดโครงสร้างย่อยๆ ต่อไป

รูปแบบการเขียนโปรแกรม C++

ภาษาโปรแกรม C++ เป็นภาษาโปรแกรมที่ไม่มีรูปแบบการเขียนตายตัว กล่าวคือ ไม่ต้องกำหนดว่า องค์ประกอบของโปรแกรมจะต้องเขียนอยู่ในบรรทัดหรือบนหน้ากระดาษส่วนไหน ดังนั้น โปรแกรมเมอร์จึงมีอิสระที่จะวางรูปแบบของโปรแกรม แต่โปรแกรมเมอร์ที่มีประสบการณ์ย่อมทราบดีว่าการเขียนโปรแกรมรูปแบบที่ได้นั้นจะต้องอ่านง่าย สะดวกต่อการแก้ไขข้อผิดพลาดของโปรแกรม และง่ายต่อการดูแลรักษาโปรแกรม แต่อย่างไรก็ตาม เราสามารถเขียนตามระเบียบแบบแผนมาตรฐานของภาษา C++ ซึ่งมีข้อปฏิบัติต่างๆ ดังต่อไปนี้

1. การเขียนประโยคเตรียมประมวลผล #include ไว้ที่ตำแหน่งเริ่มต้นของโปรแกรม
2. เขียนบรรทัดละหนึ่งคำสั่ง
3. เขียนกลุ่มคำสั่งที่อยู่ภายในบล็อกแบบย่อหน้า
4. ให้มีการเว้นวรรคตรงเครื่องหมายตัวดำเนินการทั้งก่อนและหลังเครื่องหมาย เช่น $n = 4$

ระเบียบแบบแผนอีกลักษณะหนึ่งที่พึงปฏิบัติ คือ การเขียนชื่อตัวแปร ถ้าเขียนด้วยชื่อสั้นๆ จะลดโอกาสที่จะพิมพ์ผิด แต่ในขณะเดียวกันก็ควรจะเป็นชื่อที่สื่อความหมายว่าตัวแปรนั้นแทนอะไร การเขียนรูปแบบนี้ เรียกว่า รหัสคำสั่งเอกสารในตัวเอง (Self-Documenting Code) โปรแกรมเมอร์ C++ เกือบทั้งหมดนิยมเขียนชื่อตัวแปรด้วยพิมพ์เล็ก ยกเว้นในกรณีที่ชื่อตัวแปรประกอบด้วยคำหลายๆ คำจะเขียนตัวอักษรตัวแรกของคำที่มาต่อท้ายด้วยตัวพิมพ์ใหญ่ เช่น

```
Char Middle Initial;
```

```
Unsigned Max Unsigned Int;
```

เหตุผลที่เขียนแบบนี้ เพราะจะทำให้อ่านง่ายกว่าเขียนด้วยตัวพิมพ์เล็กเพียงอย่างเดียว เช่น Middleinitial และ Maxunsignedint หรือมีอีกวิธีหนึ่งที่นิยมให้เช่นกัน คือ การใช้เครื่องหมายสัญลักษณ์ประกาศ (underscore '_') เป็นตัวแยกคำแทนช่องว่าง เช่น

```
Char middle_initial;
```

```
Unsigned Max Unsigned Int;
```

โปรแกรมภาษา C++ อย่างง่าย

ตัวอย่างแรกจะแสดงส่วนประกอบหลักของโปรแกรม C++

ตัวอย่างที่ 2-1 โปรแกรม Hello World

```

1 #include <stdio.h>
2 #include <iostream.h>
3 int main()
4 {
5     cout<<"Hello,World,\n";
6     return 0;
7 }
8

```

บรรทัดแรกของโปรแกรมเป็นการกำหนดตัวเตรียมประมวลผล (Preprocessor Directive) ด้วยคำว่า #include เพื่อแสดงว่าโปรแกรมนี้นี้มีการนำข้อมูลออกหรือมีการแสดงผลออกทางอุปกรณ์ตัวใดตัวหนึ่ง โดยการอ้างถึงไฟล์ชื่อ iostream.h ซึ่งเป็นโปรแกรมที่ทำหน้าที่จัดการกับกลุ่มสารสนเทศที่ต้องการส่งให้หน่วยควบคุมการนำข้อมูลออก คือ ตัววัตถุ cout สำหรับเครื่องหมายวงเล็บมุม "<" และ ">" ไม่นับเป็นส่วนหนึ่งของชื่อไฟล์ แต่ใช้เพื่อแสดงถึงมาตรฐานของการกล่าวถึงคลังโปรแกรม หรือไลบรารีไฟล์ (Library File) เท่านั้น ดังนั้น จึงสรุปได้ว่าเมื่อใดที่โปรแกรมต้องการใช้วัตถุ cout เพื่อส่งข้อมูลออกทางอุปกรณ์แสดงผลจะต้องมีการกำหนดตัวเตรียมประมวลผลและไฟล์ istream.h ไว้ด้วย

บรรทัดที่ 2 คือ หมายเหตุโปรแกรมหรือที่เรียกกันโดยทั่วไปว่าคอมเมนต์ (Comment) เขียนนำด้วยสัญลักษณ์ // และตามด้วยข้อความ หมายเหตุโปรแกรม คือ ข้อความที่ใช้อธิบายการทำงานของโปรแกรมนอกเหนือจากคำสั่งของโปรแกรม จุดประสงค์ของหมายเหตุโปรแกรมมีไว้สำหรับให้อ่านเท่านั้น ไม่มีผลกระทบใดๆ ต่อการทำงานของโปรแกรม

บรรทัดที่ 3 คือ การกำหนดฟังก์ชัน main() ซึ่งเป็นฟังก์ชันหลักสำหรับทุกโปรแกรมที่เขียนด้วยภาษา C++ เป็นสิ่งที่บ่งบอกจุดเริ่มต้นการปฏิบัติงานของโปรแกรม ส่วนของวงเล็บ "()" ที่อยู่หลังคำ Main เป็นสัญลักษณ์ข้อกำหนดของภาษาที่ต้องเขียนรวมอยู่ด้วย

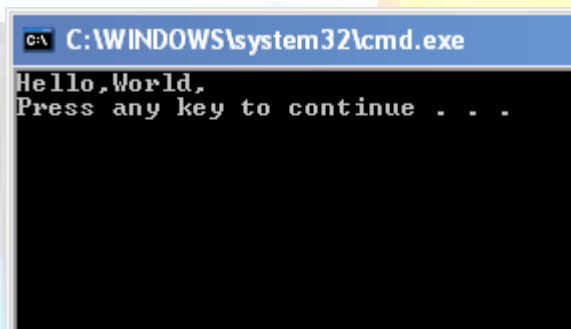
บรรทัดที่ 4 และ 7 มีเพียงเครื่องหมายวงเล็บปีกกาเปิด "{" และวงเล็บปีกกาปิด "}" ตามลำดับ ซึ่งเป็นเครื่องหมายแสดงถึงรายการคำสั่งต่างๆ ของฟังก์ชัน main () และเป็นส่วนที่ต้องมีในทุกโปรแกรมที่เขียนขึ้นด้วยภาษา C++

บรรทัดที่ 5 มีประโยคคำสั่ง ดังนี้

Cout<< "hello,world.\n";

ประโยคนี้นอกจากให้ระบบคอมพิวเตอร์ส่งข้อความ “hello world.\n” ไปที่ส่วนควบคุมการนำข้อมูลออก cout (ซี-เอาต์) วัตถุตัวนี้ คือ กระแสส่งออกมาตรฐาน โดยทั่วไปจะหมายถึงจอภาพ วัตถุ cout มาจากคำเต็มว่า Console Output คือ จอเฝ้าคุมแสดงผล

ผลลัพธ์ที่ได้จากการสั่ง Run โปรแกรมในตัวอย่างที่ 2-1 คือ



สัญลักษณ์ \n หมายถึงการขึ้นบรรทัดใหม่ ประกอบด้วยตัวอักขระสองตัว ได้แก่ เครื่องหมาย ‘\’ และตัวอักษร ‘n’ การใส่สัญลักษณ์นี้ต่อท้ายที่ข้อความภายในเครื่องหมายอัญประกาศเป็นการบอกให้ระบบคอมพิวเตอร์ขึ้นบรรทัดใหม่ หลังจากพิมพ์ตัวอักขระหรือข้อความที่อยู่หน้าเครื่องหมายนี้ หรืออีกนัยหนึ่งก็คือ เป็นเครื่องหมายแสดงจุดสิ้นสุดรายการข้อมูลของบรรทัดนั้นนั่นเอง

บรรทัดที่ 6 ประกอบด้วยคำสั่ง return 0 หมายถึง สิ้นสุดปฏิบัติการคำสั่งของโปรแกรมและสั่งการควบคุมการทำงานกลับไปให้ระบบปฏิบัติการ ส่วนเลข 0 ใช้เป็นสัญญาณแสดงการจบโปรแกรมเมื่อไม่มีข้อผิดพลาดใดๆ เกิดขึ้น

ประโยคคำสั่งแสดงผลลัพธ์ที่บรรทัดที่ 5 มีการใช้สัญลักษณ์หลายตัว หนึ่งในจำนวนนั้น คือ สัญลักษณ์ ‘<<’ เราเรียกสัญลักษณ์นี้ว่า ตัวดำเนินการส่งออก (Output Operator) หรือตัวดำเนินการแทรก (Insertion Operation) กล่าวคือ เป็นการแทรกข้อมูลเข้าไปที่กระแสส่งออก ส่วนสัญลักษณ์ \n ที่อยู่ปิดท้ายข้อความ คือ ตัวอักขระควบคุมการทำงานให้ระบบคอมพิวเตอร์ขึ้นบรรทัดใหม่ เมื่อไรก็ตามที่มีเครื่องหมายนี้ปรากฏอยู่ที่ข้อความส่งออกจะส่งผลให้บรรทัดของคำสั่งแสดงผลลัพธ์ในขณะนั้นสิ้นสุดลง และไปเริ่มต้นที่บรรทัดใหม่ ข้อสังเกตของสัญลักษณ์ทั้งสอง คือ การใช้ตัวอักขระสองตัวติดกันโดยไม่มีช่องว่างระหว่างตัวอักขระนั้น

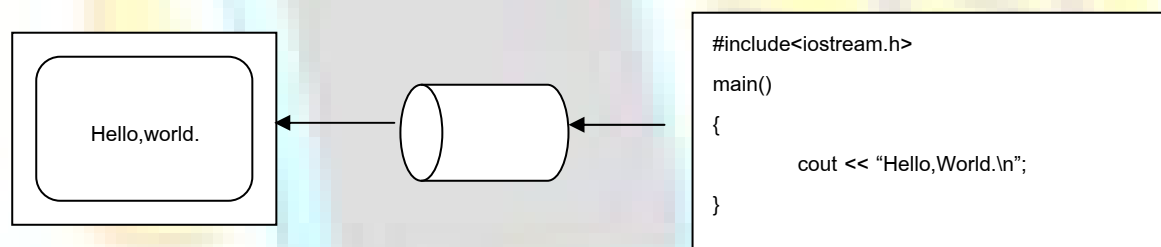
ประโยคคำสั่งในการโปรแกรมภาษา C++ ทุกคำสั่งจะต้องปิดท้ายด้วยเครื่องหมาย ‘;’ เรียกเครื่องหมายนี้ว่า เครื่องหมายอัมภาคหรือเครื่องหมายเซมิโคลอน และในหนึ่งบรรทัดอาจมีได้หลายคำสั่ง หรือในทางตรงกันข้ามอาจมีบางคำสั่งที่ต้องเขียนมากกว่าหนึ่งบรรทัด แต่ไม่ว่ากรณีใดก็ตามทุกคำสั่งจะต้องจบลงด้วยเครื่องหมายอัมภาค (;)

การส่งข้อมูลออกด้วย cout

ถึงแม้ว่าจะยังไม่ได้อธิบายรายละเอียดเกี่ยวกับวัตถุ cout ก็ตาม แต่เราก็ได้ทดลองใช้ cout ดูแล้ว ซึ่งก็แสดงให้เห็นว่าวัตถุมีจุดเด่นที่สามารถใช้งานวัตถุได้โดยไม่ต้องรู้รายละเอียดภายในของวัตถุเลย เพียงรู้วิธีการเชื่อมต่อ (Interface) ก็พอ วัตถุ cout มีวิธีการเชื่อมต่อแบบง่ายๆ

Cout<< ส่วนที่ต้องการส่งออก

ส่วนที่อยู่ทางด้านขวามือ จะถูกใส่เข้าไปในสายกระแส (Stream) cout ด้วยปฏิบัติการของเครื่องหมาย Insertion Operator (<<) ของวัตถุ cout



รูปที่ 2-1 แสดงความสัมพันธ์ของวัตถุ cout กับ โปรแกรมและจอภาพ

วัตถุ cout ทำหน้าที่เหมือนกระแสน้ำหรือสายนำส่งข้อมูลจากโปรแกรมไปปรากฏที่จอภาพ (เครื่องพิมพ์หรืออุปกรณ์แสดงผลอื่นๆ) ทีละตัวอักษรตามลำดับ

Cout สามารถทำงานได้ทั้งกับข้อความ(String) และจำนวนเต็ม ซึ่งเป็นความฉลาดของวัตถุ cout อันเป็นผลมาจากคุณลักษณะของ OOP ในภาษา C++ สำคัญก็คือ ตัวดำเนินการใส่ (Insertion Operator, <<) สามารถปรับเปลี่ยนพฤติกรรมได้ตามสภาพแวดล้อม สภาพแวดล้อมในที่นี้ หมายถึง ชนิดข้อมูลของตัวถูกดำเนินการ

ตัวอย่างที่ 2-1 ไม่ใช่โปรแกรมที่มีขนาดเล็กที่สุด การเขียนโปรแกรมภาษา C++ อาจไม่ต้องมีคำสั่งใดๆ ในโปรแกรมเลยก็สามารถเป็นโปรแกรมได้ เพียงแต่ขอให้มีส่วนประกอบหลักของโปรแกรมเท่านั้น โปรแกรมที่มีลักษณะดังกล่าว เรียกว่า โปรแกรมว่าง (Empty Program) ซึ่งไม่ได้มีวัตถุประสงค์ให้โปรแกรมทำอะไร

การใช้ cin

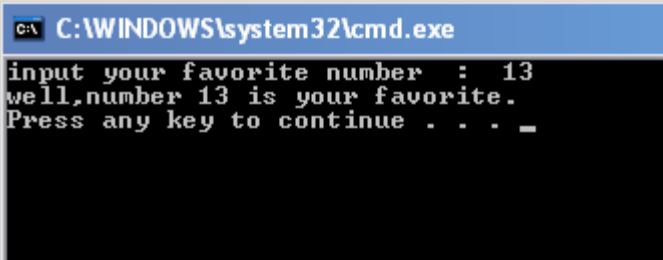
ลองมาดูตัวอย่างเพิ่มเติม โดยโปรแกรมจะถามผู้ใช้ให้ป้อนค่าเข้าเครื่องในขณะที่ Run โปรแกรมการทำงานนี้ทำได้โดยใช้ cin (อ่านว่า ซี-อิน) อ่านข้อมูลเข้า

ตัวอย่างที่ 2-2 ให้มีการป้อนเลขที่ชอบและโปรแกรมจะแสดงข้อความตอบกลับมา

```

1 #include <stdio.h>
2 #include <iostream.h>
3 main()
4 {
5     int num;
6     cout<<"input your favorite number : ";
7     cin>>num;
8     cout<<"well,number "<<num <<" is your favorite.\n";
9     return 0;
10 }
11

```



จากโปรแกรมจะเห็นว่า ค่าที่พิมพ์เข้าไปทางแป้นพิมพ์ (13) จะถูกให้ค่าแก่ตัวแปร num คำสั่งที่อยู่เบื้องหลังปฏิบัติการนี้ คือ

Cin >> num;

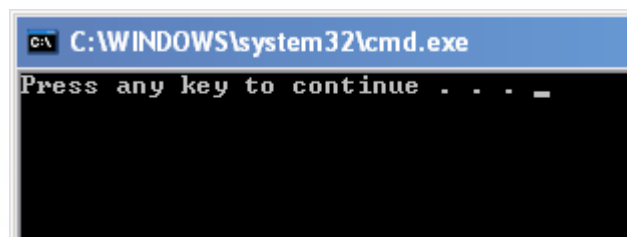
สำหรับข้อมูลนำเข้า cin ใช้เครื่องหมายดำเนินการ >> ทำหน้าที่ดึง (Extract) ข้อมูลจากสายกระแสนำเข้า โดยปกติจะวางตัวแปรไว้ทางขวาของเครื่องหมาย >> เพื่อรอรับข้อมูลที่ดึงออกมา

ตัวอย่างที่ 2-3 โปรแกรม C++ ที่สั้นที่สุด

```

1 #include <stdio.h>
2 #include <iostream.h>
3 int main()
4 {
5
6     return 0;
7 }
8

```



ในโปรแกรมนี้เรียกได้ว่าเป็นโปรแกรมว่าง หมายถึง โปรแกรมที่ไม่ได้ทำการประมวลผลคำสั่งใดๆ ทั้งสิ้น เป็นเพียงแสดงให้เห็นโครงสร้างของโปรแกรม C++ ที่ทุกโปรแกรมจะต้องมีเท่านั้น

คำสั่ง Return 0; อาจไม่จำเป็นต้องใช้กับคอมพิวเตอร์ส่วนใหญ่ คอมพิวเตอร์บางตัวจะแจ้งข้อความเตือนถ้าไม่มีการระบุไว้ แต่ในบทนี้จะมีการเขียนไว้ทุกโปรแกรม อย่างไรก็ตามควรมีการเขียนหมายเหตุโปรแกรมประกอบทุกโปรแกรม เพื่ออธิบายให้ผู้อ่านทราบพอสังเขปว่าโปรแกรมนั้นทำอะไร

การเขียนคอมเมนต์ (Comment)

คอมเมนต์หรือหมายเหตุโปรแกรม หมายถึง ข้อความที่เขียนอธิบายไว้ร่วมกับตัวโปรแกรม แต่คอมไพเลอร์จะไม่นำข้อความนั้นมาปฏิบัติตาม เนื่องจากมีไว้เพื่อให้ผู้ใช้โปรแกรมได้อ่าน ซึ่งจะช่วยให้สามารถทำความเข้าใจการทำงานของโปรแกรมได้ง่ายขึ้น และยังเป็นประโยชน์ต่อการบำรุงรักษาโปรแกรมต่อไป เราเรียกหมายเหตุโปรแกรมว่า คอมเมนต์

การเขียนหมายเหตุโปรแกรมในภาษา C++ เขียนได้ 2 รูปแบบ คือ การเขียนแบบมาตรฐานภาษา C จะเริ่มต้นส่วนของคอมเมนต์ด้วยเครื่องหมาย /* และจบด้วยเครื่องหมาย */ ข้อความที่เขียนอยู่ระหว่างเครื่องหมายทั้งสอง คือ หมายเหตุโปรแกรม ซึ่งคอมไพเลอร์จะไม่สนใจแปลข้อความดังกล่าว ดังนั้น จะไม่มีการตรวจสอบไวยากรณ์ของภาษาที่จุดนี้

ตัวอย่างการเขียนหมายเหตุโปรแกรมแบบมาตรฐานภาษา C

```
/* this is a C style comment */
```

การเขียนหมายเหตุโปรแกรมรูปแบบที่สอง คือ การเขียนแบบมาตรฐานภาษา C++ ซึ่งเริ่มต้นด้วยเครื่องหมาย // และตามด้วยข้อความที่จะใช้เป็นเครื่องหมายเหตุโปรแกรม

ตัวอย่างการเขียนโปรแกรมหมายเหตุโปรแกรมแบบมาตรฐานภาษา C

```
// this is a C++ style comment
```


โปรแกรมเมอร์ส่วนใหญ่นิยมใช้รูปแบบที่สองเพราะเขียนง่ายและสังเกตเห็นได้ง่ายในส่วน
ของโปรแกรม แต่อย่างไรก็ดี การเขียนหมายเหตุโปรแกรมแบบมาตรฐานภาษา C ก็มีความจำเป็น ถ้า
โปรแกรมเมอร์ต้องการใส่ส่วนของหมายเหตุไว้ในประโยคคำสั่ง แต่ไม่แนะนำให้ถือปฏิบัติเช่นนี้

เครื่องหมายจบประโยคคำสั่ง

ภาษา C++ ใช้เครื่องหมายอัฒภาคหรือเซมิโคลอน (;) แสดงจุดสิ้นสุดของประโยคคำสั่งนั่นคือ
ประโยคคำสั่งทุกประโยคต้องจบลงด้วยเครื่องหมายอัฒภาค (;) ซึ่งจะแตกต่างจากภาษาโปรแกรมภาษา
อื่น เช่น ภาษาปาสคาล โดยที่ภาษาปาสคาลใช้เครื่องหมายอัฒภาคเป็นตัวคั่นระหว่างคำสั่ง แต่สำหรับ
ประโยคที่ขึ้นต้นด้วยสัญลักษณ์จำนวนหรือแฮช (#) เช่น

```
#include<iostream.h>
```

จะไม่จบด้วยเครื่องหมายอัฒภาค เพราะประโยคนี้นี้ไม่ใช่ประโยคคำสั่งแต่เป็นประโยคที่กำหนด
ตัวเตรียมประมวลผล จากตัวอย่างที่ผ่านมาจะเห็นว่าประโยค C++ สามารถทำการแปลเป็นนิพจน์ หรือ
ในทางกลับกันนิพจน์ก็สามารถเป็นประโยคในภาษา C++ ได้ด้วย นิพจน์สามารถกำหนดให้เป็น
ประโยคเดี่ยวๆ เช่น ตัวอย่างสองประโยคต่อไปนี้ถือว่าเป็นประโยค C++ ที่ถูกต้อง

```
x + y;
```

```
22;
```

ประโยคทั้งสองนี้ไม่มีผลอะไรต่อโปรแกรม ดังนั้น จึงถือว่าเป็นประโยคที่สูญเปล่า แต่
อย่างไรก็ดี ถือว่าเป็นประโยคที่ถูกต้องตามหลักภาษา C++ เราจะดูนิพจน์ที่เป็นประโยคต่อไป
ภายหลัง

เครื่องหมายอัฒภาคเปรียบเหมือนตัวดำเนินการบนนิพจน์ ซึ่งเปลี่ยนนิพจน์ไปเป็นประโยค
คำสั่ง เครื่องหมายนี้ไม่ใช่ตัวดำเนินการที่ถูกต้องเพราะผลลัพธ์ คือ ประโยคคำสั่ง ไม่ใช่ค่าข้อมูล แต่ก็
เป็นจุดที่ช่วยอธิบายให้เห็นความแตกต่างระหว่างนิพจน์และประโยคคำสั่ง

คำสงวนและการกำหนดชื่อ (Reserved Words & Identifier)

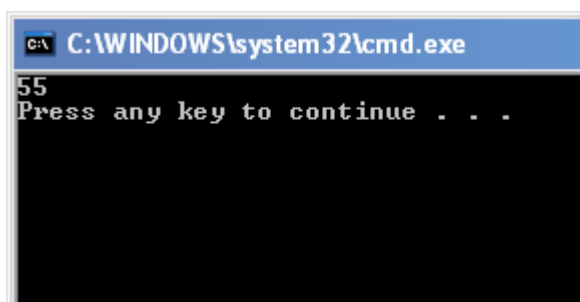
ในภาษาโปรแกรมต่างๆ ตัวโปรแกรมมักจะมีย่อประกอบจากการสร้างประโยคตาม
กฎไวยากรณ์ของภาษาสำหรับสมาชิกตัวหนึ่ง โดยจะเรียกสมาชิกเหล่านี้ว่า โทเค็น (Tokens) โทเค็น
เหล่านี้รวมถึงชื่อตัวแปร ค่าคงที่ คำหลัก ตัวดำเนินการ และเครื่องหมายวรรคตอน

ตัวอย่างที่ 2-4 โปรแกรมการแสดงสมาชิกโทเค็น


```

1  #include <stdio.h>
2  #include <iostream.h>
3  int main()
4  {
5      int n = 55;
6      cout << n << endl;
7      return 0;
8  }
9

```



ตัวอย่างโปรแกรมนี้แสดงให้เห็นจำนวนโทเค็น 15 ตัว ได้แก่

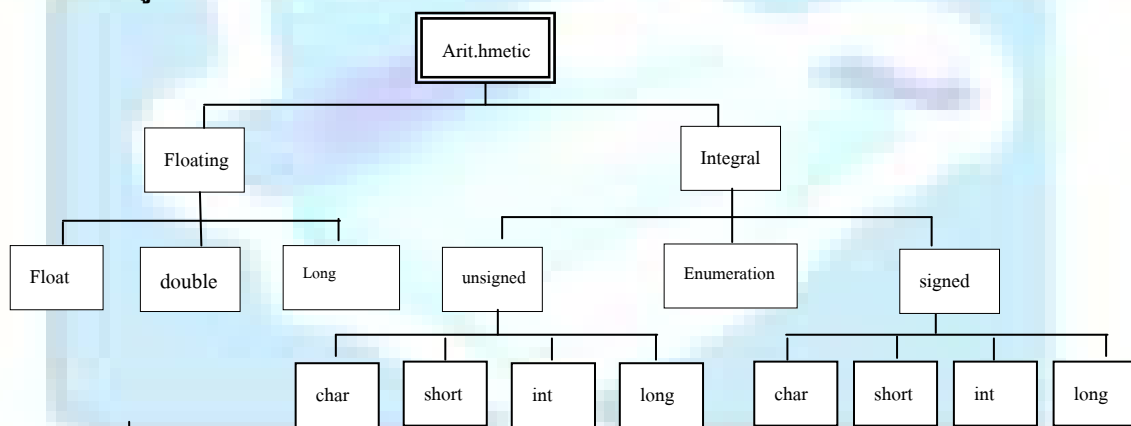
main,(,),{,int,n,55,;,cout,<<,endl,return,0 และ } โดยที่โทเค็น n คือ ตัวแปร, โทเค็น 55,0,และ endl คือ ค่าคงที่สำหรับโทเค็น int และ return คือ คำหลัก โทเค็น = และ << คือ ตัวดำเนินการ ส่วนโทเค็น (,),{,; และ } คือ เครื่องหมายวรรคตอน สำหรับสองบรรทัดแรกเป็นการกำหนดตัวชี้ทาง (Directive) และหมายเหตุโปรแกรม ซึ่งไม่ถือว่าเป็นส่วนประกอบที่แท้จริงของโปรแกรม

คำสงวน (Reserved Words) เป็นคำสงวนไว้เฉพาะตัวภาษาโปรแกรม C++ เท่านั้น ไม่สามารถนำไปตั้งชื่อให้กับตัวแปร หรือนำไปใช้เพื่อวัตถุประสงค์อื่นได้

การกำหนดชื่อ (Identifier) คือ สายอักขระที่ประกอบด้วยตัวอักษรเลข (Alphanumeric) ที่มีตัวอักขระตัวแรกเป็นตัวอักษร ถ้าจัดกลุ่มของตัวอักขระจะได้ว่า มีจำนวนตัวอักขระที่เป็นตัวอักษรทั้งหมด 53 ตัว แบ่งเป็นตัวอักษรภาษาอังกฤษจำนวน 52 ตัว และเครื่องหมายสัญลักษณ์ประกาศ (_) รวมเป็น 53 ตัว กลุ่มที่สอง คือ ตัวอักษรเลขมีจำนวน 63 ตัว ได้แก่กลุ่มของตัวอักษร 53 ตัว รวมกับกลุ่มของตัวเลขอีก 10 ตัว (0,1,2,3,...,9) ดังนั้น main(),int,n,cout และ endl จึงมีลักษณะเป็นตัวยอมรับได้ เช่นเดียวกับคำต่อไปนี้อย่าง x1 y4 LastName และ the_day_after_tomorrow เป็นต้น ภาษา C++ จะตรวจสอบตัวอักษรที่เป็นตัวพิมพ์ใหญ่กับตัวพิมพ์เล็ก ดังนั้นตัวยอมรับที่ตั้งชื่อเหมือนกันแต่พิมพ์ด้วยตัวอักษรที่ต่างกัน โปรแกรม C++ จะมองตัวยอมรับทั้งสองเป็นคนละตัวกัน เช่น ตัวยอมรับ stack กับ stack จะหมายถึง

ตัวระบุให้สำหรับการตั้งชื่อให้กับสิ่งต่างๆ ในโปรแกรม เช่น ชื่อตัวแปร และชื่อฟังก์ชันเป็นต้น จากตัวอย่างข้างต้น main เป็นชื่อของฟังก์ชัน int เป็นชื่อของแบบชนิดข้อมูล n และ cout เป็นชื่อของตัวแปร ส่วน endl เป็นชื่อค่าคงที่ ตัวระบุบางตัว เช่น int ถือเป็นคำสงวนเพราะเป็นส่วนของภาษาโปรแกรม ส่วนตัวระบุอื่นๆ เช่น n ถูกกำหนดโดยตัวโปรแกรม

ชนิดของข้อมูล



รูปที่ 2-2 แสดงชนิดของข้อมูลในภาษา C++ บางส่วน

จำนวนเต็ม คือ ตัวเลขทั้งหมดได้แก่ 0,1,-1,2,-2,3,-3 เป็นต้น สำหรับจำนวนเต็มชนิด unsigned integer คือ จำนวนเต็มที่ไม่ใช่ลบ เช่น 0,1,2,3, ฯลฯ ภาษา C++ จำแนกจำนวนเต็มออกเป็น 9 ชนิดได้แก่

Char	short int	unsigned short int
Signed char	int	unsigned int
Unsigned char	long int	unsigned long int

ความแตกต่างของจำนวนเต็มทั้ง 9 ชนิดนี้ คือ พิสัยของจำนวนเต็มซึ่งขึ้นกับระบบเครื่องคอมพิวเตอร์ที่นำมาใช้งาน เช่น บนระบบคอมพิวเตอร์พีซีที่ใช้ระบบปฏิบัติการ DOS พิสัยของจำนวนเต็มชนิด int มีค่าอยู่ระหว่าง -32768 และ 32768 ในขณะที่บนระบบปฏิบัติการ UNIX ซึ่งทำงานบนเครื่องเวิร์กสเตชันจะมีพิสัยอยู่ระหว่าง -2147483648 และ 2147483647 คำว่า int สามารถใช้ได้ตอนที่กำหนดแบบชนิดข้อมูลดังต่อไปนี้ short,int,long,unsigned short int,unsigned int และ unsigned long int

ตัวอย่างโปรแกรมที่ 2-5 นี้ จะแสดงพิสัยของจำนวนเต็มชนิดต่างๆ ตามข้อกำหนดของระบบคอมพิวเตอร์ที่ใช้อยู่ สำหรับค่าคงที่ SCHAR_MIN, LONG_MAX ฯลฯ เป็นค่าคงที่ของขีดจำกัดต่ำสุดและสูงสุดซึ่งได้เก็บอยู่ในตัวชี้ทางไฟล์ที่ชื่อ <limits.h> ดังนั้น ตอนเริ่มต้นโปรแกรมจึงต้องมีการกำหนดประโยคตัวเตรียมประมวลผลดังนี้

```
#include<limits.h>
```

ตัวอย่างที่ 2-5 พิสัยของเลขจำนวนเต็ม

โปรแกรมนี้จะพิมพ์ขอบเขตข้อมูลจำนวนเต็มแต่ละชนิด

```
1 #include <stdio.h>
2 #include <iostream.h>
3 #include <limits.h>
4 int main()
5 {
6     cout<<"minimum char ="<<CHAR_MIN<<endl;
7     cout<<"maximum char ="<<CHAR_MAX<<endl;
8     cout<<"minimum short ="<<SHRT_MIN<<endl;
9     cout<<"maximum short ="<<SHRT_MAX<<endl;
10    cout<<"minimum int ="<<INT_MIN<<endl;
11    cout<<"maximum int ="<<INT_MAX<<endl;
12    cout<<"minimum long ="<<LONG_MIN<<endl;
13    cout<<"maximum long ="<<LONG_MAX<<endl;
14    cout<<"minimum signed char ="<<SCHAR_MIN<<endl;
15    cout<<"maximum signed char ="<<SCHAR_MAX<<endl;
16    cout<<"maximum unsigned char ="<<UCHAR_MAX<<endl;
17    cout<<"maximum unsigned short ="<<USHRT_MAX<<endl;
18    cout<<"maximum unsigned ="<<UINT_MAX<<endl;
19    cout<<"maximum unsigned long ="<<ULONG_MAX<<endl;
20    return 0;
21 }
22
```

```

C:\WINDOWS\system32\cmd.exe
minimum char =-128
maximum char =127
minimum short =-32768
maximum short =32767
minimum int =-32768
maximum int =32767
minimum long =-2147483648
maximum long =2147483647
minimum signed char=-128
maximum signed char =127
maximum unsigned char=255
maximum unsigned short=65535
maximum unsigned =65535
maximum unsigned long=4294967295
Press any key to continue . . .

```

ผลลัพธ์ที่เกิดขึ้นนี้ได้จากระบบ UNIX บนเครื่องเวิร์กสเตชัน ซึ่งถ้าพิจารณาให้ดีจะเห็นว่า มีข้อมูลที่แตกต่างกันเพียง 6 ชนิดเท่านั้น ดังนี้

Char	ช่วงพิสัย	-128 ถึง 127 (1 ไบต์)
short	ช่วงพิสัย	-32768 ถึง 32767 (2 ไบต์)
int	ช่วงพิสัย	-2147483648 ถึง 2147483647 (4 ไบต์)
unsigned char	ช่วงพิสัย	0 ถึง 255 (1 ไบต์)
unsigned short	ช่วงพิสัย	0 ถึง 65535 (2 ไบต์)
unsigned	ช่วงพิสัย	0 ถึง 4294967295 (4 ไบต์)

จำนวนเต็มชนิด short ใช้เนื้อที่ 2 ไบต์ (จำนวน 16 บิต) เนื่องจากมีข้อมูลอยู่ในช่วงพิสัย -32768 ถึง 32767 ซึ่งมีจำนวนข้อมูลเท่ากับ $65536 = 2^{16}$ ตัว (1 ไบต์มีขนาด 8 บิต)

บนระบบโปรแกรม Borland C++ ให้พิสัยค่าเท่ากัน ยกเว้นชนิด int และ unsigned ซึ่งมีค่าดังนี้

Int	ช่วงพิสัย	-32768 ถึง 32767 (2 ไบต์)
Unsigned	ช่วงพิสัย	0 ถึง 65535 (2 ไบต์)

Char ในภาษา C++ ข้อมูลชนิด char เป็นข้อมูลจำนวนเต็มชนิดหนึ่ง หมายความว่า ตัวแปรใด ๆ ก็ตามที่ถูกกำหนดให้เป็นชนิด char สามารถนำมาใช้ในประโยคนิพจน์จำนวนเต็มได้เช่นเดียวกับข้อมูลจำนวนเต็มชนิดอื่น ดังตัวอย่างการใช้งานต่อไปนี้

```
Char C = 54;
```

```
Char d = 2*C-7;
```

```
C+ = d%3;
```

คำว่า char เป็นคำย่อของคำว่า character การใช้ char เมื่อตัวแปรชนิดนี้ถูกใช้เป็นข้อมูลรับเข้า หรือข้อมูลส่งออกจะทำการแปลงเป็นตัวอักษร เมื่อใดก็ตามที่ตัวอักษรถูกใช้เป็นข้อมูลรับเข้า ระบบคอมพิวเตอร์จะจัดเก็บรหัสแอสกี (ASCII) ไว้เป็นจำนวนเต็ม และเมื่อใดก็ตามที่ ตัวแปรชนิด char ถูกส่งเป็นข้อมูลส่งออก ระบบจะส่งตัวอักษรที่มีรหัสแอสกีตามนั้น ไปยังส่วนของกระแสส่งออก ดังแสดงในตัวอย่างข้างล่างนี้

ภาษา C++ ได้นิยามข้อมูลจำนวนเต็มขนาด 8 บิต ไว้ 3 ชนิด คือ char, signed char และ unsigned char แต่มีเพียง 2 ชนิดเท่านั้นที่แตกต่างกัน คือ ชนิด char จะเป็น signed char หรือ unsigned char ขึ้นอยู่กับคอมพิวเตอร์จะใช้ชนิด char สำหรับตัวอักษรธรรมดา และใช้ unsigned char สำหรับข้อมูลสายอักขระที่มีขนาดสั้นมากๆ ส่วนชนิด signed char ไม่มีปรากฏให้เห็นบ่อยนัก ข้อมูลชนิดที่กล่าวมานี้เหมาะสำหรับจัดเก็บข้อมูลที่มีค่าน้อยๆ แต่มีเป็นจำนวนมาก และไม่ต้องการส่งผลลัพธ์โดยผ่านทางตัวดำเนินการส่งออก <<

ตัวอย่างที่ 2-6 แสดงผลลัพธ์แบบตัวอักษร

ตัวอย่างนี้แสดงให้เห็นว่าตัวแปร char ให้ผลลัพธ์อย่างไร

```

1  #include <stdio.h>
2  #include <iostream.h>
3  int main()
4  {
5      char c=64;
6      cout<<c++<<" "; // ให้แสดง @ c= 64 ตรงกับเลข แอสกี และจะตรงกับ ตัว@ นั่นเอง
7      cout<<c++<<" ";
8      cout<<c++<<" ";
9      cout<<c++<<" "<<endl;
10     c=96;
11     cout<<c++<<" ";
12     cout<<c++<<" ";
13     cout<<c++<<" ";
14     cout<<c++<<endl;
15     return 0;
16 }
```

```
C:\WINDOWS\system32\cmd.exe
a b c
Press any key to continue . . .
```

คำสั่งแสดงผลคำสั่งแรกจะส่งตัวแปร c ไปยังกระแสส่งออก เนื่องจากข้อมูลดังกล่าว มีค่าเท่ากับ 64 ผลลัพธ์ที่เกิดขึ้น คือ ตัวอักษร “@” (ตัวอักษร “@” มีรหัสแอสกีเท่ากับ 64) ต่อจากนั้น c จะได้รับการเพิ่มค่าเป็น 65 ซึ่งส่งผลให้ผลลัพธ์ตัวถัดไปคือ ตัวอักษร “A” (ตัวอักษร “A” มีรหัสแอสกีเท่ากับ 65) ส่วนที่เหลือของโปรแกรมจะดำเนินการต่อไปในลักษณะเดิม (ถ้าระบบคอมพิวเตอร์ที่ใช้งานอยู่เป็นรหัส EBCDIC ผลลัพธ์ที่ได้จะแตกต่างออกไป)

ตัวอย่างที่ 2-7 แสดงรหัสแอสกี

```
1 #include <stdio.h>
2 #include <iostream.h>
3 main()
4 {
5     char c='A';
6     cout<<c<<" "<<int(c++)<<endl;
7     cout<<c<<" "<<int(c++)<<endl;
8     cout<<c<<" "<<int(c++)<<endl;
9     return 0;
10 }
```

```
C:\WINDOWS\system32\cmd.exe
A 65
B 66
C 67
Press any key to continue . . .
```

เมื่อโปรแกรมนี้ได้รับการประมวลผล จะทำให้ตัวแปร c มีค่าเป็น 65, 66 และ 67 ตามลำดับ แต่เนื่องจากตัวแปรอักษรได้พิมพ์ออกมาเป็นตัวอักษร ดังนั้น ผลลัพธ์ที่เกิดขึ้นในแต่ละบรรทัด คือ ตัว

นิพจน์ `int (c)` ซึ่งเรียกว่า (Cast) มีหน้าที่แปลงข้อมูลตัวอักษรเป็นจำนวนเต็ม ซึ่งทำให้สามารถพิมพ์รหัสแอสกีของตัวอักษรได้

สัญลักษณ์ “Hello,” ซึ่งเรียกว่าสายอักขระ (String) ประกอบด้วยลำดับของตัวอักษรที่อยู่ในเครื่องหมายอัญประกาศ (“”)

สายอักขระ (String) หรือเรียกกันโดยทั่วไปว่า สตริง คือ ลำดับของตัวอักษรที่อยู่ต่อเนื่องกันในหน่วยความจำและสิ้นสุดลงที่ตัวอักษร Null คือ ‘\0’ ข้อมูลสายอักขระถูกเข้าถึงด้วยตัวแปรชนิด `char*` (ตัวชี้ไปที่ `char`) เช่น ถ้า `s` เป็นชนิด `char*` แล้วประโยคคำสั่ง `cout<< s<<endl;` จะพิมพ์ตัวอักษรทั้งหมดที่เก็บอยู่ในหน่วยความจำโดยเริ่มต้นที่เลขที่อยู่ `s` และจบลงทันทีที่พบกับตัวอักษร Null

ตัวอักษร (Character) หมายถึง สัญลักษณ์ที่ใช้ในภาษาต่างๆ ซึ่งมีทั้งสัญลักษณ์ในรูปตัวอักษร ตัวเลข และสัญลักษณ์อื่นๆ ได้แก่ A-Z, 0-9 และเครื่องหมายต่างๆ ที่คอมพิวเตอร์สามารถอ่านเข้าใจความหมายและนำไปเก็บไว้ในหน่วยความจำได้ (ช่องว่างก็นับเป็น “ตัวอักษร” ด้วย) เครื่องคอมพิวเตอร์ส่วนใหญ่ที่ใช้กันอยู่จะใช้ชุดตัวอักษรรหัสแอสกี (ASCII มาจากคำเต็มว่า American Standard Code for Information Interchange) ชุดอักขระนี้รวมทั้งตัวอักษรตัวเล็กและตัวใหญ่จำนวน 52 ตัว ตัวเลขจำนวน 10 ตัว เครื่องหมายวรรคตอนที่พบเห็นบนแป้นพิมพ์และตัวอักษรที่ไม่ใช้ในงานพิมพ์ แต่ใช้สำหรับควบคุมการทำงานของโปรแกรม

ตัวอย่างตัวอักษรที่ไม่นำมาใช้ในงานพิมพ์ เช่น ‘\n’ กำหนดขึ้นด้วยเครื่องหมายตัวอักษรทับหลัง (\) และตัวอักษร n นอกจากนี้ ยังมีตัวอักษรในลักษณะเช่นนี้อีกมาก เช่น ‘\t’ หมายถึง การตั้งระยะ และตัวอักษรเตรียมพร้อม คือ ‘\a’ หมายถึง เมื่อมีการพิมพ์จะมีเสียงบีป (Beep) เกิดขึ้น นอกจากนี้ ถ้าต้องการพิมพ์ตัวอักษรที่มีสัญลักษณ์ตรงกับตัวที่ใช้ในการควบคุมการทำงานของโปรแกรม เช่น ต้องการพิมพ์เครื่องหมายอัญประกาศก็สามารถทำได้โดยการพิมพ์ ‘\’ หรือถ้าต้องการพิมพ์เครื่องหมาย \ ก็สามารถพิมพ์ได้โดยการพิมพ์ ‘\\’

ตัวอักษรสามารถนำมาใช้เป็นส่วนหนึ่งของสายอักขระในคำสั่งโปรแกรม หรือเป็นข้อมูลตัวเดียวโดดๆ ก็ได้ ถ้าเมื่ออยู่ในลักษณะของข้อมูลตัวเดียวโดดๆ จะหมายถึง ค่าคงที่อักขระ และ จะต้องอยู่ในเครื่องหมายอัญประกาศเดี่ยว (‘’)

โปรแกรมที่ 2-8 ตัวอย่างโปรแกรม Hello World อีกรูปแบบหนึ่ง


```

1 #include <stdio.h>
2 #include <iostream.h>
3 int main()
4 {
5     cout << "Hello," << 'W' << 'o' << 'r' << "ld" << '.' << '\n';
6     return 0;
7 }
8

```

```

C:\WINDOWS\system32\cmd.exe
Hello,World.
Press any key to continue . . .

```

ประโยคคำสั่งในโปรแกรมนี้อาจจะดูสับสนว่าทำไมถึงใช้ cout จำนวน 7 ชุด ได้แก่ สายอักขระ 2 ชุด คือ “Hello,” และ “ld” กับค่าคงที่อักขระ 5 ตัว คือ ‘W’, ‘o’, ‘r’, ‘.’ และ ‘\n’

อย่างไรก็ตาม ตัวอักษรใดๆ สามารถนำมาประกอบกันเป็นสายอักขระได้ ดังนั้น จากประโยคคำสั่งข้างต้นอาจเขียนแทนด้วย

```
cout << "Hello," << 'W' << 'o' << 'r' << "ld" << '.' << '\n';
```

ประโยคนี้อาจจะส่งสายอักขระจำนวน 7 ชุด ไปยัง cout ซึ่งทำให้ได้ผลลัพธ์เหมือนกัน แต่การใช้เป็นค่าคงที่อักขระจำดีกว่า เนื่องจากการจัดเก็บข้อมูลสายอักขระจะสิ้นเปลืองมากกว่า

นอกจากนี้ยังมีสายอักขระแบบพิเศษ กล่าวคือ ไม่มีตัวอักษรใดๆ ปรากฏอยู่ในเครื่องหมายอัญประกาศ และเขียนแสดงด้วย "" เรียกว่าสายอักขระว่าง (Empty String) ซึ่งสามารถพิมพ์ข้อความด้วยการนำสายอักขระว่างมาใช้ประกอบได้ด้วยได้ เช่น

```
cout << "Hello, Wo" << "" << "r!" << "" << "" << "d.\n";
```

ความยาวของสายอักขระ คือ จำนวนของตัวอักษรที่ประกอบกันเป็นข้อความนั้น เช่น “ABCDE” จะมีความยาวเท่ากับ 5

C++ ได้จัดเตรียมฟังก์ชัน `strlen()` ไว้สำหรับหาค่าความยาวของสายอักขระ ดังตัวอย่างการใช้งานต่อไปนี้

ตัวอย่างที่ 2-9 ตัวอย่างนี้จะแสดงความยาวของสายอักขระจากข้อความหลายๆ ชุด

```

1 #include<iostream.h>
2 #include<string.h>
3 int main()
4 {
5     cout<<strlen("Hello, World.\n")<<\n';
6     cout<<strlen("Hello, World.")<<\n';
7     cout<<strlen("Hello, ")<<\n';
8     cout<<strlen("H")<<\n';
9     cout<<strlen("")<<\n';
10    return 0;
11 }
12
C:\WINDOWS\system32\cmd.exe
14
13
7
1
0
Press any key to continue . . . _
  
```

ฟังก์ชัน `strlen()` จะทำหน้าที่นับจำนวนตัวอักขระในสายอักขระที่ทำการระบุอยู่ในฟังก์ชัน จากคำสั่งสองคำสั่งแรกนับจำนวนตัวอักขระได้ 14 และ 13 ตัวตามลำดับ เนื่องจากสัญลักษณ์ `\n` จะนับเป็นหนึ่งตัวอักขระเท่านั้น สำหรับสายอักขระ "Hello" มีความยาวเท่ากับ 7 สายอักขระ "H" มีความยาวเท่ากับ 1 และสายอักขระว่างมีความยาวเท่ากับ 0

ฟังก์ชัน `strlen()` อ่านว่า 'สเติร์-เลน (Str-Len)' ได้กำหนดแยกไว้ในไฟล์ `string.h` ซึ่งมีมาพร้อมกับระบบโปรแกรม C++ ดังนั้น ถ้าเมื่อใดที่มีการใช้ฟังก์ชัน `strlen()` จะต้องมีการประกาศตัวเตรียมประมวลผลด้วย ดังนี้

```
#include<string.h>
```

โดยกำหนดไว้ที่บรรทัดของ `include` เหนือส่วนของฟังก์ชัน `main()`

ตัวดำเนินการ (Operator)

ตัวดำเนินการ หรือโอเปอเรเตอร์ของ C++ คือ สัญลักษณ์ที่ใช้ทำหน้าที่คำนวณนิพจน์เมื่อได้ค่าผลลัพธ์ จะนำเอามาเก็บไว้ที่ตัวแปร เราได้พบตัวดำเนินการส่งออก << และตัวดำเนินการกำหนดค่า = มาแล้ว

ตัวดำเนินการพื้นฐานส่วนใหญ่จะเป็นตัวดำเนินการทางคณิตศาสตร์ ได้แก่ เครื่องหมาย +, -, *, / และ % เครื่องหมายเหล่านี้จะดำเนินการกับข้อมูลจำนวนเต็ม และได้ผลลัพธ์เป็นจำนวนเต็มแต่อาจเป็นลบก็ได้ เช่น $m + n$ จะให้ค่าเป็นผลรวมของค่า m กับ n และประโยค $m - n$ จะให้ค่าเป็นผลต่างระหว่าง m กับ n สำหรับ $-n$ หมายถึง ค่าลบของ n ประโยค $m * n$ คือ ผลคูณของ m กับ n ประโยค m / n คือ ผลหารที่เป็นส่วนจำนวนเต็มเมื่อทำการหาร m ด้วย n และประโยค $m \% n$ คือ เศษเหลือที่เป็นจำนวนเต็มเมื่อทำการหาร m ด้วย n ดังนั้น จึงสรุปการทำงานของตัวดำเนินการได้ดังตารางที่ 2-1

ตารางที่ 2-1 ตัวดำเนินการของ C++ (บางส่วน)

โอเปอเรเตอร์ทางเลขคณิต		หน้าที่
ตัวอย่าง		
+	บวก	$m + n$
-	ลบ	$m - n$
-	เครื่องหมายลบหน้า	$-n$
*	คูณ	$m * n$
/	หาร	m / n
%	เศษเหลือ	$m \% n$
++	เพิ่มค่าอีก 1	$++n, n++$
--	ลดค่าลง 1	$--n, n--$

โอเปอเรเตอร์ทางตรรกะ ตัวอย่าง	หน้าที่
&&	และ
	หรือ
!	นิเสธ
==	เท่ากับ
!=	ไม่เท่ากับ
>	มากกว่า
>=	มากกว่าหรือเท่ากับ
<	น้อยกว่า
<=	น้อยกว่าหรือเท่ากับ

โอเปอเรเตอร์แบบ bit ตัวอย่าง	หน้าที่
&	และ
	หรือ
^	เอกซ์คลูซีฟออร์
~	นิเสธ
>>	เลื่อนไปทางขวา
<<	เลื่อนไปทางซ้าย

ในส่วน of ตัวโอเปอเรเตอร์แบบ bit จะมีใช้เฉพาะสำหรับ type ชนิด จำนวนเต็มเท่านั้น

ตัวอย่างที่ 2-10 ตัวดำเนินการจำนวนเต็ม

โปรแกรมนี้แสดงการใช้ตัวดำเนินการคณิตศาสตร์ 6 ตัวที่แสดงไว้ในตารางที่ 2-1

```

1 #include <stdio.h>
2 #include <iostream.h>
3 int main()
4 {
5     int m=38,n=5;
6     cout<<m<<"+"<<n<<"="<<(m+n)<<endl;
7     cout<<m<<"-"<<n<<"="<<(m-n)<<endl;
8     cout<<"-"<<n<<"="<<(-n)<<endl;
9     cout<<m<<"*"<<n<<"="<<(m*n)<<endl;
10    cout<<m<<"/"<<n<<"="<<(m/n)<<endl;
11    cout<<m<<"%"<<n<<"="<<(m%n)<<endl;
12    return 0;
13 }
14

```

```

C:\WINDOWS\system32\cmd.exe
38+5=43
38-5=33
-5=-5
38*5=190
38/5=7
38%5=3
Press any key to continue . . . _

```

จากตัวอย่างข้างต้นเป็นที่น่าสังเกตว่า $38/5 = 7$ และ $38\%5 = 3$ ซึ่งการดำเนินการทั้งสองอย่างนี้ให้ผลลัพธ์ที่ได้จากการหาร 38 ด้วย 5 ได้เท่ากับ $38/5 = 7.6$ ส่วนผลหารที่เป็นจำนวนเต็มคือ $38/5 = 7$ และเศษที่เหลือที่เป็นจำนวนเต็ม คือ 3 ซึ่งสามารถรวมกับตัวตั้ง 38 และตัวหาร 5 ดังความสัมพันธ์ต่อไปนี้ $7*5+3 = 38$

การหารในลักษณะดังกล่าวจะยุ่งยากมากขึ้นถ้าเลขจำนวนเต็มนั้นไม่ใช่จำนวนมาก และที่แน่นอน คือ ตัวหารต้องไม่เป็นศูนย์ แต่ถ้าทั้ง m หรือ n เป็นจำนวนเต็มลบแล้ว m/n และ $m\%n$ อาจให้ผลลัพธ์ที่แตกต่างกันไปในแต่ละระบบคอมพิวเตอร์ มีเพียงจุดเดียวเท่านั้นที่เป็นไปได้คือ

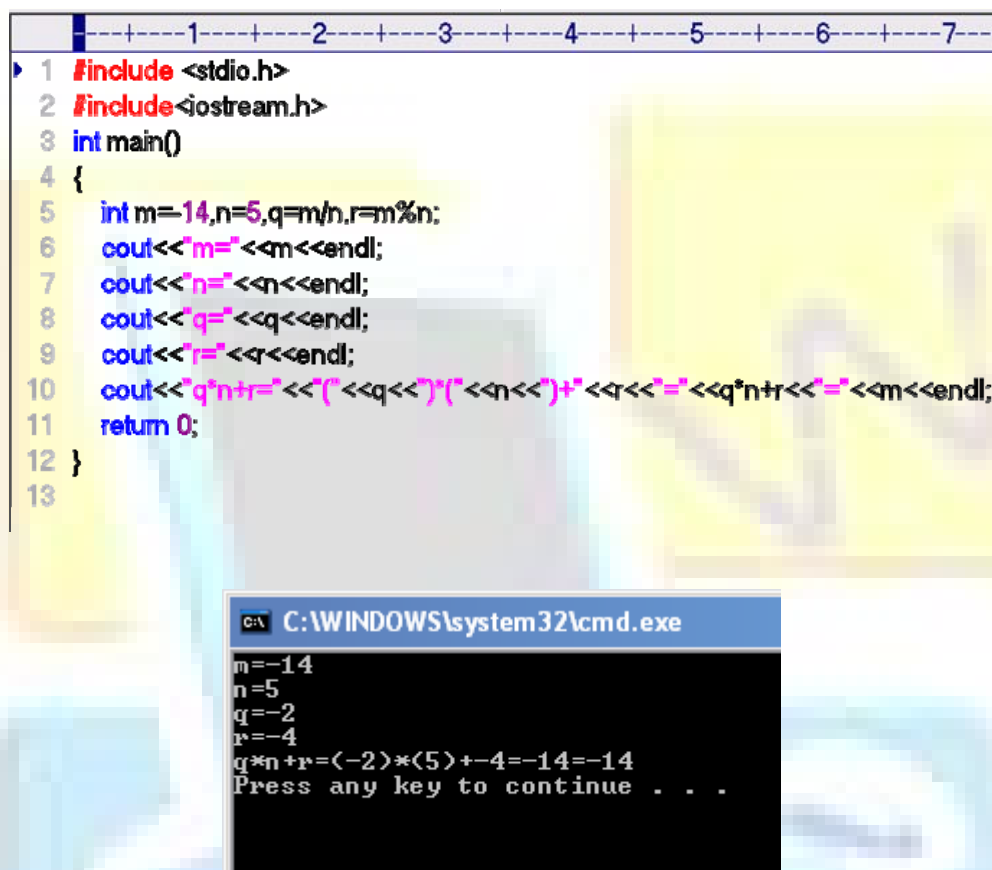
$$q * n + r = m$$

โดยที่ $q = m/n$ และ $r = m\%n$

ตัวอย่างการคำนวณ -14 หารด้วย 5 ผลลัพธ์ที่ได้ -2.8 สำหรับผลหารที่เป็นจำนวนเต็มจะถูกปัดเศษเป็น -3 หรือ -2 ถ้าระบบคอมพิวเตอร์ ปัดผลหาร q เป็น -3 แล้ว เศษที่เหลือ r จะเป็น 1 แต่ถ้าระบบคอมพิวเตอร์ปัดผลหาร q เป็น -2 แล้ว เศษที่เหลือ r จะเป็น -4

ตัวอย่างที่ 2-11 การหารจำนวนเต็มลบ

โปรแกรมนี้ใช้สำหรับพิจารณาว่าคอมพิวเตอร์จัดการอย่างไรกับการหารของเลขจำนวนเต็มลบ



```

1 #include <stdio.h>
2 #include <iostream.h>
3 int main()
4 {
5     int m=-14,n=5,q=m/n,r=m%n;
6     cout<<"m="<<m<<endl;
7     cout<<"n="<<n<<endl;
8     cout<<"q="<<q<<endl;
9     cout<<"r="<<r<<endl;
10    cout<<"q*n+r="<<"("<<q<<")("<<n<<")+<<r<<""<<q*n+r<<""<<m<<endl;
11    return 0;
12 }
13
C:\WINDOWS\system32\cmd.exe
m=-14
n=5
q=-2
r=-4
q*n+r=(-2)*(5)+-4=-14=-14
Press any key to continue . . .

```

ลำดับการทำการก่อนและการจัดกลุ่มตัวดำเนินการ

ภาษา C++ มีเครื่องหมายตัวดำเนินการอยู่หลายตัว เนื่องจากนิพจน์หนึ่งๆ อาจประกอบด้วยตัวดำเนินการจำนวนหลายตัว ดังนั้น จึงต้องพิจารณาถึงลำดับของการคำนวณด้วย มักจะคุ้นเคยกันดีอยู่แล้ว สำหรับลำดับการทำการก่อนของตัวดำเนินการทางคณิตศาสตร์ เช่น เครื่องหมาย *,/ และ % จะมีลำดับการทำการก่อนสูงกว่าเครื่องหมาย + และ - ซึ่งเครื่องหมายที่มีลำดับการทำการก่อนสูงกว่าจะได้รับการดำเนินการก่อน เช่น

นิพจน์ $42 - 3 * 5$

ลำดับการประมวลผล คือ

$$42 - (3 * 5) = 42 - 15 = 27$$

ตารางที่ 2-2 ลำดับความสำคัญของโอเปอเรเตอร์ในภาษา C++ (บางส่วน)

[]	กำหนดดัชนีของอาร์เรย์
()	กำหนดฟังก์ชัน
++,--	เพิ่ม,ลด (วางข้างหลังมีลำดับสูงกว่า)
sizeof	ขนาดของตัวแปร หรือ type
~	นิเสธแบบ bit
!	นิเสธทางตรรกะ
-	เครื่องหมายลบนำหน้า
*,/,%	คูณ,หาร,หารเก็บเศษ (ลำดับเดียวกัน)
+, -	บวก,ลบ (ลำดับเดียวกัน)
<<,>>	เลื่อนซ้าย,เลื่อนขวา
<,>,<=,>=	เปรียบเทียบต่างๆ
=,!=	เท่ากับ,ไม่เท่ากับ
&&	และ ทางตรรกะ
	หรือ ทางตรรกะ

กรณีที่มีตัวดำเนินการที่ต่างกัน แต่มีลำดับการมาก่อนเท่ากับปรากฏอยู่ในนิพจน์เดียวกัน เช่น
 ดำเนินการ + และ - ทั้งคู่อยู่ในระดับเดียวกันและเป็นการจัดกลุ่มซ้าย ดังนั้น ตัวดำเนินการจะทำงานจาก
 ซ้ายไปขวา ดังตัวอย่างต่อไปนี้

$$8 - 5 + 4$$

การทำงานลำดับแรก คือ $8 - 5$ ได้ผลลัพธ์เท่ากับ 3 จึงนำไปบวกกับ 4 ดังนี้

$$(8 - 5) + 4 = 3 + 4 = 7$$

ตัวดำเนินการเพิ่มและลด

ลักษณะที่สำคัญหลายๆ อย่างของภาษา C++ ได้มาจากภาษา C สิ่งหนึ่งที่เป็นประโยชน์อย่างมาก คือ ตัวดำเนินการเพิ่ม ++ และลด -- ตัวดำเนินการเหล่านี้จะช่วยลดรูปแบบการเขียนประโยคนิพจน์เมื่อต้องการเพิ่มหรือลดค่าตัวแปร โดยเขียนให้อยู่ในรูปแบบอย่างย่อ ดังตัวอย่างการใช้งานต่อไปนี้

ตัวอย่างที่ 2-12 ตัวดำเนินการเพิ่มและลด

โปรแกรมนี้แสดงการทำงานของตัวดำเนินการเพิ่มและลด


```

1 #include <stdio.h>
2 #include <iostream.h>
3 int main()
4 {
5     int m=44,n=66;
6     cout<<"m="<<m<<","n="<<n<<endl;
7     ++m;
8     --n;
9     cout<<"m="<<m<<","n="<<n<<endl;
10    m++;
11    n--;
12    cout<<"m="<<m<<","n="<<n<<endl;
13    return 0;
14 }
15

```

```

C:\WINDOWS\system32\cmd.exe
m=44,n=66
m=45,n=65
m=46,n=64
Press any key to continue . . .

```

จากตัวอย่างจะสังเกตเห็นได้ว่า ตัวดำเนินการเพิ่มทั้งสองลักษณะ คือ เพิ่มก่อนและหลังตัวแปร ได้แก่ ++m และ m++ จะให้ค่าเหมือนกันคือ การเพิ่มค่า 1 ให้กับตัวแปร m ในทำนองเดียวกัน สำหรับ ตัวดำเนินการลดทั้งก่อนและหลังได้แก่ --n และ n-- ก็จะทำให้ค่าเท่ากันซึ่งหมายถึง การลบค่า 1 ออกจากตัวแปร n

การเขียนนิพจน์ตามลำดับในลักษณะเช่นนี้ คือ ++m และ m++ จะมีความหมายเหมือนกันกับประโยคนิพจน์ต่อไปนี้

```
m=m+1;
```

หมายถึง การเพิ่มค่าของตัวแปร m อีก 1 ค่า ในลักษณะที่คล้ายกัน คือ --n และ n-- จึงเหมือนกับประโยคนิพจน์

```
n=n-1;
```

อย่างไรก็ตาม ถ้ามีการใช้ตัวดำเนินการเพิ่มและลดกับนิพจน์ย่อย (หมายถึง ส่วนนิพจน์ที่อยู่ในนิพจน์ประโยคใหญ่) ตัวดำเนินการเพิ่มก่อน เช่น ++m จะให้ความหมายที่ต่างจาก m++ โดยที่ตัว

เนื่องจากกระบวนการเพิ่มเทียบเท่ากับการใช้การกำหนดค่าแบบแยกตามลำดับ เพราะจริงๆ แล้วเสมือนมีสองคำสั่งที่จะต้องปฏิบัติงาน เมื่อการดำเนินการเพิ่มค่าได้ใช้เป็นส่วนของนิพจน์ย่อย การกำหนดค่าเพิ่มและนิพจน์ที่ใหญ่กว่าที่อยู่ในประโยคเดียวกัน ความแตกต่างระหว่างการเพิ่มค่าก่อนและการเพิ่มค่าหลัง มีความแตกต่างกันตรงที่ปฏิบัติตัวกำหนดค่าก่อนหรือหลังปฏิบัติการนิพจน์ที่อยู่ติดกัน ตัวอย่าง ที่ 2-13 ตัวดำเนินการเพิ่มก่อนและหลัง

โปรแกรมนี้แสดงความแตกต่างระหว่างตัวดำเนินการเพิ่มก่อนและหลัง

```

1  #include <stdio.h>
2  #include <iostream.h>
3  int main()
4  {
5      int m = 66,n;
6      n=++m;
7      cout<<"m="<<m<<","n"<<n<<endl;
8      n=m++;
9      cout<<"m="<<m<<","n"<<n<<endl;
10     cout<<"m="<<m++<<endl;
11     cout<<"m="<<m<<endl;
12     cout<<"m="<<++m<<endl;
13     return 0;
14 }

```

```

C:\WINDOWS\system32\cmd.exe
m=67,n67
m=68,n67
m=68
m=69
m=70
Press any key to continue . . .

```

การใช้ตัวดำเนินการเพิ่มและลดในนิพจน์ย่อนี้ อาจทำให้เกิดการสับสน ดังนั้น จึงควรใช้ด้วยความระมัดระวัง เช่น ลำดับของการทำงานนิพจน์ที่มีส่วนเกี่ยวข้องกับตัวดำเนินการเหล่านี้ไม่ได้ถูกกำหนดในหลักการของภาษาโปรแกรม เมื่อเป็นเช่นนี้จึงไม่อาจคาดการณ์ได้

ตัวอย่างที่ 2-14 แสดงผลลัพธ์ที่ไม่สามารถทราบค่าที่แน่นอนได้ของการคำนวณนิพจน์ย่อ

```

1  #include <stdio.h>
2  #include <iostream.h>
3  int main()
4  {
5      int n=5,x;
6      x=++n*-n;
7      cout<<"n="<<n<<" x="<<x<<endl;
8      cout<<++n<<" "<<++n<<" "<<++n<<endl;
9      return 0;
10 }
11
C:\WINDOWS\system32\cmd.exe
n=5,x=25
6 7 8
Press any key to continue . . . _

```

ในการกำหนดค่าให้กับตัวแปร x นั้น เริ่มต้นด้วย n ได้เพิ่มค่าก่อนเป็น 6 แล้วจึงถูกลดค่ากลับมาเป็น 5 ก่อนที่จะทำการคูณ ดังนั้น จึงได้กำหนดผลการคูณ $5*5$ ให้กับตัวแปร x ในบรรทัดสุดท้าย นิพจน์ย่อจะได้รับการประมวลผลจากขวาไปซ้าย การจัดกลุ่มซ้ายของตัวดำเนินการส่งออก << ไม่อยู่ในประเด็น เพราะไม่มีเครื่องหมายตัวดำเนินการอื่นที่มีลำดับการทำก่อนระดับเดียวกันปะปนอยู่ในนิพจน์นี้ด้วย

การกำหนดนิพจน์เชิงประกอบ

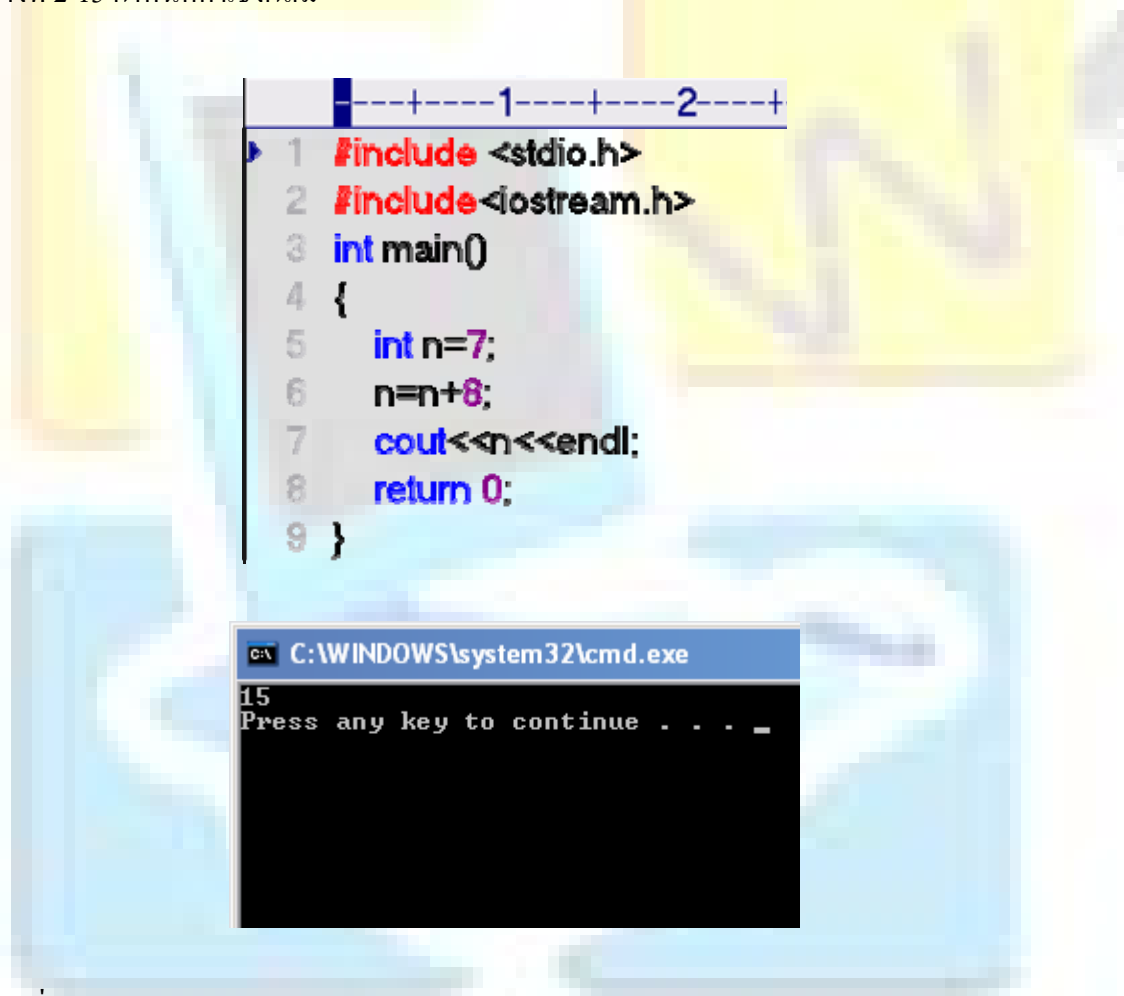
ตัวดำเนินการเพิ่มและลด คือ การกำหนดค่านิพจน์อย่างย่อ ภาษา C++ สามารถกำหนดค่าโดยใช้ร่วมกับตัวดำเนินการอื่นได้ ซึ่งมีรูปแบบการเขียนเชิงผสม ดังนี้

Variable op = expression

โดยที่ op คือ ตัวดำเนินการ ผลของการกำหนดค่าเชิงผสมมีผลเหมือนกับ

Variable = Variable op expression

ตัวอย่างที่ 2-15 กำหนดค่าเชิงผสม



ตัวอย่างที่ 2-16 ตัวดำเนินการกำหนดค่า

โปรแกรมนี้แสดงการใช้ตัวดำเนินการเชิงผสม

```

1 #include <stdio.h>
2 #include <iostream.h>
3 int main()
4 {
5     int n= 44;
6     n+=9;
7     cout<<n<<endl;
8     n-=5;
9     cout<<n<<endl;
10    n*=2;
11    cout<<n<<endl;
12    return 0;
13 }

```

```

C:\WINDOWS\system32\cmd.exe
53
48
96
Press any key to continue . . .

```

ประโยค `n += 9` หมายถึง การเพิ่มค่า 9 ให้กับตัวแปร `n` ประโยค `n -= 5` หมายถึง การลบค่า `n` ด้วย 5 ประโยค `n *= 2` หมายถึง การคูณ `n` ด้วย 2

ส่วนล้น (Overflow)

จำนวนเต็มในเครื่องคอมพิวเตอร์จะมีขอบเขตจำกัดตามที่กล่าวมาแล้ว ไม่เหมือนกับจำนวนเต็มทางคณิตศาสตร์ทั่วไป ดังนั้น ถ้าค่าของตัวแปรจำนวนเต็มเกินพิสัยจะเกิดสถานการณ์ที่เรียกว่า ส่วนล้น (Overflow)

ตัวอย่างที่ 2-17 การทดสอบ Overflow

โปรแกรมนี้จะแสดงให้เห็นว่าเกิดอะไรขึ้น เมื่อวัตถุชนิด `short` มีส่วนล้นเกินพิสัย

```
1 #include<stdio.h>
2 #include<iostream.h>
3 #include<limits.h>
4 main()
5 {
6     short n =SHRT_MAX-1;
7     cout<<n++<<endl;
8     cout<<n++<<endl;
9     cout<<n++<<endl;
10    cout<<n++<<endl;
11    return 0;
12 }
```

```
C:\WINDOWS\system32\cmd.exe
32766
32767
-32768
-32767
Press any key to continue . . . _
```

ค่าของจำนวนเต็มจะมีค่าใกล้เคียงค่าที่จุดพิสัยของ 32767 และ -32768 จากตัวอย่างจะพบว่าค่าของ 32767 +1 ได้เท่ากับ -32768 ซึ่งเป็นค่าที่ผิด

คอมพิวเตอร์ส่วนใหญ่จะมีลักษณะการจัดการกับส่วนสั้นเช่นนี้ ดังนั้น ค่าที่อยู่ถัดจากค่าสูงสุดจะกลายเป็นค่าต่ำสุดซึ่งเป็นจุดที่อันตรายมาก เพราะไม่มีสัญญาณบอกให้ทราบถึงจุดผิดพลาดตรงนี้ โปรแกรมยังสามารถดำเนินการไปได้ตามปกติ

การเกิดส่วนสั้นของจำนวนเต็มเช่นนี้ เป็นข้อผิดพลาดชนิดหนึ่งที่เกิดขึ้นตอน Run โปรแกรม ตัวอย่างข้อผิดพลาดอื่นๆ ที่เกิดกับข้อมูลจำนวนเต็ม คือ การหารด้วยศูนย์ แต่ข้อผิดพลาดนี้สามารถรู้ได้ เพราะโปรแกรมจะเกิดการขัดข้องและไม่ทำงานต่อ