บทที่ 1
รู้จักกับการเขียนโปรแกรม

การเขียนโปรแกรมคอมพิวเตอร์ คือการเขียนชุดคำสั่งเพื่อให้คอมพิวเตอร์ทำงานตามต้องการ ซึ่งชุดคำสั่งสำหรับให้คอมพิวเตอร์ทำงานมีมากมายหลายแบบ แต่ชุดคำสั่งที่คอมพิวเตอร์เข้าใจได้คือภาษาเครื่อง (machine code) แต่เนื่องจากว่าการเขียนชุดคำสั่งโดยใช้ภาษาเครื่องนั้นมีความยุ่งยาก จึงได้เกิดชุดคำสั่งใหม่ๆขึ้นอีกมากมาย เช่น ภาษาแอสเซมบลี ภาษาซี ภาษาปาสคาล ภาษาเบสิก ภาษาซีพลัสพลัส ภาษาจาวา ฯลฯ โดยที่แต่ละภาษาก็จะมีความซับซ้อนและรูปแบบการเขียนที่แตกต่างกันไป ซึ่งภาษาเหล่านี้จะต้องมีตัวแปรภาษา หรือเรียกว่าคอมไพเลอร์ หรือ อินเตอพรีเตอร์ (compiler or interpreter) ต่างๆกันไป ที่คอยทำหน้าที่แปลชุดคำสั่งที่เขียนเพื่อให้เป็นภาษาเครื่องให้คอมพิวเตอร์สามารถเข้าใจได้และทำงานได้ตามความต้องการ

นอกจากภาษาสำหรับเขียนโปรแกรม ตัวแปรภาษาแล้ว ยังมีเครื่องมือสำหรับใช้เขียนโปรแกรม หรือเรียกว่า Editor ปัจจุบันมีหลายๆบริษัทสร้าง editor ที่รองรับการเขียนชุดคำสั่งของภาษาต่างๆมากมายโดยที่ editor เหล่านี้ก็จะรวมเอาตัวแปรภาษาของภาษานั้นๆเข้าไปด้วย ทำให้การเขียนโปรแกรมเกิดความสะดวกยิ่งขึ้น เพียงมี editor ที่รองรับตัวแปรภาษาที่ต้องการเขียนเพียงอย่างเดียว ก็สามารถทำให้เขียนโปรแกรมได้แล้ว

จากข้อมูลข้างต้นสามารถสรุปตัวอย่างเพื่อความเข้าใจได้ดังตาราง

ชื่อภาษา	ตัวแปรภาษา	ตัวอย่างโปรแกรม editor
ภาษาซี	ภาษาซี	Turbo C ,Borland C++, Visual C++, Visual Studio.net 2005
ภาษาซีพลัสพลัส	ภาษาซีพลัสพลัส	Turbo C++,Borland C++,Visual C++ , Visual Studio.net 2005
ภาษาจาวา	ภาษาจาวา	JCreator,Netbeans,Eclipse
ภาษาปาสคาล	ภาษาปาสคาล	Turbo Pascal

รูปแบบการเขียนโปรแกรม (Programming Paradigm)

รูปแบบการเขียนโปรแกรมนั้นมีหลากหลายรูปแบบ ซึ่งแต่ละรูปแบบมีคุณสมบัติและจุดเด่นแตกต่างกันไป ในที่นี้จะยกตัวอย่างเพียงบางส่วนเพื่อประกอบความเข้าใจในการเรียน

1. การเขียนโปรแกรมแบบมีขั้นตอน (procedural programming) มีคุณสมบัติพิเศษคือ
   - สามารถใช้โค้ดเดิม ในตำแหน่งอื่นๆของโปรแกรมได้ โดยไม่ต้องคัดลอกโค้ดนั้นซ้ำอีกครั้ง
   - สามารถแบ่งโค้ดเขียนแยกเป็นส่วนๆแล้วนำมารวมกันได้
   - มีวิธีการเขียนโค้ดสำหรับการเขียนชุดคำสั่งสำหรับให้โปรแกรมย้ายไปทำงานในจุดต่างๆได้ง่ายกว่าการใช้คำสั่ง GOTO หรือ JUMP

ตัวอย่างภาษาที่ใช้รูปแบบการเขียนแบบนี้ได้แก่ Ruby , PHP , Perl , Python ,C , C++ , JAVA

2. การเขียนโปรแกรมเชิงวัตถุ (object oriented programming) มีคุณสมบัติพิเศษคือ
- ออกแบบโปรแกรมในรูปแบบของวัตถุ
- สามารถมีการถ่ายทอดคุณสมบัติระหว่างวัตถุได้
- สามารถแบ่งโค้ดเขียนแยกเป็นส่วนๆแล้วนำมารวมกันได้
- สามารถเขียนโค้ดทับส่วนของการกระทำของวัตถุที่ได้รับการถ่ายทอดมาได้
- สามารถกำหนดให้บางคุณสมบัติของโปรแกรมไม่สามารถแก้ไขเปลี่ยนแปลงได้

ตัวอย่างภาษาที่ใช้รูปแบบการเขียนแบบนี้ได้แก่ Ruby , PHP , Perl , Python , C++ , JAVA

ซึ่งภาษาโปรแกรมที่พัฒนาขึ้นในปัจจุบันนี้ ส่วนมากมีรูปแบบภาษาที่รองรับการเขียนโปรแกรมเชิงวัตถุ แต่ทว่าการเขียนโปรแกรมเชิงวัตถุนั้น วิธีการเขียนส่วนหนึ่งมีพื้นฐานมาจากรูปแบบการเขียนโปรแกรมแบบมีขั้นตอนรวมอยู่ด้วย ดังนั้นในการเขียนโปรแกรมเบื้องต้น ผู้เรียนควรทำความเข้าใจรูปแบบการเขียนโปรแกรมแบบมีขั้นตอนอย่างละเอียดก่อนการเรียนการเขียนโปรแกรมเชิงวัตถุต่อไป

สำหรับเนื้อหาต่อไปนี้จะมุ่งเน้นการทำความเข้าใจในรูปแบบการเขียนโปรแกรมแบบมีขั้นตอนและการออกแบบโปรแกรมมากกว่าที่จะมุ่งเน้นภาษาใดเป็นสำคัญ เนื่องจากการใช้ภาษาในการเขียนโปรแกรมนั้นเป็นเพียงการเลือกเครื่องมือสำหรับใช้งานเท่านั้น เมื่อผู้เรียนสามารถเข้าใจวิธีการเขียนโปรแกรม และหลักการในการออกแบบโปรแกรมแล้ว ไม่ว่าจะเลือกใช้ภาษาใดในการเขียนโปรแกรมก็จะไม่เป็นเรื่องยากอีกต่อไป เพราะจะแตกต่างกันตรงที่รูปแบบของชุดคำสั่งเท่านั้น แต่วิธีการหลักๆยังคงอยู่บนพื้นฐานเดียวกัน แต่เนื่องจากต้องมีการยกตัวอย่างประกอบความเข้าใจและเพื่อฝึกฝนการเรียนรู้ เนื้อหาในเอกสารเล่มนี้จึงอิงกับภาษาซีพลัสพลัส เนื่องจากภาษาซีพลัสพลัสเป็นภาษาที่ได้รับความนิยม อีกทั้งยังรองรับรูปแบบการเขียนทั้งสองรูปแบบ ลักษณะชุดคำสั่งไม่ซับซ้อนเกินไปนัก เหมาะแก่การเรียนรู้สำหรับผู้เริ่มต้นเป็นอย่างยิ่ง

บทที่ 2
รู้จักกับภาษาซีพลัสพลัส

ภาษาซีพลัสพลัส (C++ programming language) คือภาษาเขียนโปรแกรมที่สามารถจัดหมวดหมู่ให้อยู่ในภาษาระดับสูงและระดับต่ำได้ และมีรูปแบบการเขียนโปรแกรมรวมอยู่หลากหลายรูปแบบ ทั้งรูปแบบการเขียนโปรแกรมแบบมีขั้นตอน การเขียนโปรแกรมเชิงวัตถุ และการเขียนโปรแกรมโดยทั่วไป (Generic programming) ดังนั้นภาษาซีพลัสพลัสจึงจัดให้เป็นหมวดหมู่ภาษาระดับกลาง ซึ่งภาษาซีพลัสพลัสนั้นถูกสร้างโดย Dr. Bjarne Stroustrup ที่ห้องแลปเบลล์ (Bell Labs) ในปี ค.ศ. 1979 โดยอ้างอิงพื้นฐานมาจากภาษาซี และเพิ่มคุณสมบัติใหม่ๆให้ภาษาซีพลัสพลัสมีความแตกต่างและมีความสามารถมากยิ่งขึ้น

คุณลักษณะสำคัญของภาษาซีพลัสพลัสที่ควรรู้

1. เฉพาะ ฟังก์ชันหลัก(main function) เท่านั้นที่ถูกทำงานเป็นอันดับแรกเสมอในโปรแกรม
2. ลักษณะการทำงาน โปรแกรมเริ่มทำงานตั้งแต่ชุดคำสั่งบรรทัดแรกในฟังก์ชันหลักและเรียงลำดับตามชุดคำสั่งไปจนจบ
3. เมื่อเจอชื่อฟังก์ชันใดๆในฟังก์ชันหลัก คอมไพเลอร์ก็จะกระโดดไปทำงานที่ฟังก์ชันชื่อนั้นๆ และเมื่อทำงานเสร็จ ก็จะกลับมายังตำแหน่งก่อนหน้าที่จะไป
4. ไฟล์โค้ดของโปรแกรม นามสกุล .cpp
5. เมื่อทำการแปลชุดคำสั่งแล้วจะได้ ไฟล์นามสกุล .obj
6. เมื่อทำการรันโปรแกรมแล้วจะได้ไฟล์นามสกุล .exe ซึ่งไฟล์นามสกุล .exe คือผลลัพธ์ของโปรแกรมสำหรับนำไปใช้งานต่อไป

ตัวอย่างแสดงคุณสมบัติข้างต้น

```
code
void functionA()
{
cout<<" A";
}

int main()
{
cout<<"1 "<<endl;
cout<<"2 "<<endl;
return 0;
}
```

```
output
1
2
```

```
code
void functionA()
{
cout<<" A";
}

int main()
{
cout<<"1 "<<endl;
A();
cout<<"2 "<<endl;
return 0;
}
```

```
output
1
A2
```

จะเห็นว่าตัวอย่างแรก ฟังก์ชัน functionA ไม่ได้ถูกทำงาน เพราะที่ฟังก์ชันหลักไม่ได้มีชุดคำสั่งให้ฟังก์ชัน functionA ทำงาน แต่ในตัวอย่างที่สอง ฟังก์ชัน functionA จะทำงาน เพราะฟังก์ชันหลักมีการสั่งให้ฟังก์ชัน funcitonA ทำงานด้วยการเขียนชื่อฟังก์ชันไว้ที่ฟังก์ชันหลัก และการทำงานของคอมไพเลอร์จะเรียงลำดับตั้งแต่บรรทัดแรกในฟังก์ชันหลักทีละบรรทัด เมื่อพบว่ามีการเรียกใช้ฟังก์ชันอื่นๆให้ทำงาน ก็จะทำงานที่ฟังก์ชันนั้นๆ และกลับมาทำงานต่อยังบรรทัดต่อมาจนกระทั่งจบชุดคำสั่งทั้งหมดในฟังก์ชันหลัก

บทที่ 3
เริ่มต้นเรียนรู้ภาษาซีพลัสพลัส

1. สิ่งที่ควรรู้

การเขียนโปรแกรมในภาษาซีพลัสพลัส เมื่อจบหนึ่งชุดคำสั่งจะต้องลงท้ายด้วยเครื่องหมาย ; เสมอ

โค้ดโปรแกรมจะถูกคอมไพเลอร์แปลคำสั่งเพื่อไปทำงานเสมอ แต่โค้ดส่วนที่อยู่ภายในเครื่องหมาย คอมเมนท์(comment)จะไม่ถูกนำไปคอมไพล์ ดังนั้นนักเขียนโปรแกรมนิยมเขียนคอมเมนท์ไว้เพื่อเตือนความจำ ฯลฯ ซึ่งวิธีการเขียนคอมเมนท์มี 2 รูปแบบ ดังนี้

1.2.1 ใช้เครื่องหมาย // ตามด้วยข้อความ หนึ่งบรรทัด รูปแบบนี้สามารถคอมเมนท์ได้เพียงบรรทัดเดียว

1.2.2 ใช้เครื่องหมาย /* ตามด้วยข้อความที่ต้องการคอมเมนท์ แล้วตามด้วยเครื่องหมาย */ รูปแบบนี้สามารถคอมเมนท์ได้หลายบรรทัด ซึ่งข้อความที่อยู่ภายในเครื่องหมาย /* */ จะไม่ถูกแปล

การเขียนโปรแกรมในภาษาซีพลัสพลัส จะต้องมีการบอกขอบเขตของชุดคำสั่งเสมอ ซึ่งขอบเขตของแต่ละชุดคำสั่ง จะอยู่ในเครื่องหมาย { }

2. ชนิดข้อมูล

ภาษาซีพลัสพลัสมีชนิดข้อมูลพื้นฐานหลายชนิดดังนี้

ชื่อเรียก	คุณสมบัติ	ขนาด	ขนาดความจุในการเก็บข้อมูล
char	ตัวอักษรหนึ่งตัว หรือ เลขจำนวนเต็มขนาดน้อยๆ	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	เลขจำนวนเต็มขนาดเล็ก	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	เลขจำนวนเต็ม	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	เลขจำนวนเต็มขนาดยาวๆ	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	ค่าตรรกศาสตร์เป็นได้แค่สองอย่าง คือ true หรือ false	1byte	true or false
float	เลขทศนิยม	4bytes	3.4e +/- 38 (7 digits)
double	เลขทศนิยมขนาดมากกว่า float	8bytes	1.7e +/- 308 (15 digits)
long double	เลขทศนิยมขนาดความละเอียดมากกว่า	8bytes	1.7e +/- 308 (15 digits)
wchar_t	ตัวอักษรขนาดใหญ่	2bytes	1 ตัวอักษรขนาดใหญ่

3. ตัวแปร

การประกาศตัวแปรแต่ละครั้ง จะมีการจองเนื้อที่หน่วยความจำ เพื่อใช้สำหรับบันทึกค่าต่างๆ ซึ่งจะจองด้วยขนาดเท่าไหร่นั้นขึ้นอยู่กับชนิดข้อมูลของตัวแปรนั้นๆ

3.1 การตั้งชื่อตัวแปร

ต้องขึ้นต้นด้วยตัวอักษร หรือเครื่องหมาย _ เท่านั้น แล้วตามด้วย ตัวอักษร หรือ ตัวเลข หรือเครื่องหมาย _ โดยไม่สามารถเว้นช่องว่างระหว่างตัวแปรหนึ่งชื่อได้ และไม่สามารถมีเครื่องหมาย _ มากกว่า 1 ตัวได้ (ทั้งนี้ขึ้นอยู่กับแต่ละคอมไพเลอร์)

3.2 วิธีการประกาศตัวแปร

```
        ชนิดข้อมูล  ชื่อตัวแปร
เช่น      int              x    ;
```

ถ้าตัวแปรหลายๆตัวมีชนิดข้อมูลเดียวกันสามารถประกาศในชุดคำสั่งเดียวกันได้ โดยใช้เครื่องหมาย , คั่นระหว่างแต่ละตัวแปร เช่น int x , y ;

นอกจากจะประกาศตัวแปรแล้วยังกำหนดค่าเริ่มต้นให้กับตัวแปรด้วยก็ได้ ด้วยการเขียนเครื่องหมาย = แล้วตามด้วยค่าที่กำหนดให้ เช่น int x = 10; หรือ char y = ‘A’ , z = ‘B’ ;

3.3 รูปแบบตัวแปร มี2 รูปแบบคือ

3.3.1 ตัวแปรแบบสาธารณะ (global variable) ตัวแปรชนิดนี้จะเป็นที่รู้จักไปทั่วทั้งไฟล์ โค้ดโปรแกรม วิธีการประกาศตัวแปรชนิดนี้จะต้องประกาศไว้ที่ต้นไฟล์โค้ดโปรแกรม ก่อนที่จะถูกเอามาเรียกใช้เท่านั้น

3.3.2 ตัวแปรแบบท้องถิ่น (local variable) ตัวแปรชนิดนี้จะเป็นที่รู้จักเฉพาะในขอบเขตของ { } ที่ตัวแปรถูกประกาศไว้เท่านั้น วิธีการประกาศตัวแปรชนิดนี้ ประกาศไว้ในฟังก์ชันที่ใช้งานเท่านั้น

ตัวอย่าง การประกาศตัวแปรแบบ global variable

```
int x ;
void functionA()
{
cout<<” A”;
}

int main()
{
cout<<” x= “<<x<<endl;
return 0;
}
```

ตัวอย่าง การประกาศตัวแปรแบบ local variable

```
void functionA(int x)
{
cout<<" x = "<<x<<endl;
}

int main()
{
คอมไพเลอร์ไม่รู้จัก x เขียนแบบนี้ไม่ได้
cout<<" x= "<<x<<endl;
return 0;
}
```

```
void functionA(int x)
{
cout<<" x = "<<x<<endl;
}

int main()
{
int  x=10;
cout<<" x= "<<x<<endl;
return 0;
}
```

x ในฟังก์ชัน main เป็นคนละตัวกับใน functionA จะไม่รู้จักกัน เพราะฉะนั้นค่าก็ไม่เท่ากันด้วย

4. ฟังก์ชัน

ในภาษาซีพลัสพลัส ฟังก์ชันที่ถูกทำงานคือ ฟังก์ชันหลัก ซึ่งจะต้องตั้งชื่อฟังก์ชันนี้ว่า main เท่านั้น เพราะฉะนั้นการสร้างฟังก์ชันอื่นๆมีข้อกำหนดดังนี้

โครงสร้างการเขียนฟังก์ชัน มีดังนี้

```
ชนิดข้อมูลของ output   ชื่อฟังก์ชัน  (  ชนิดข้อมูลของ input )
{


}
```

หมายเหตุ ชื่อของฟังก์ชันต้องติดกันเป็นคำๆเดียวเท่านั้น

ฟังก์ชันมี สองชนิดหลักๆคือ

ฟังก์ชันที่มีการคืนค่า

ฟังก์ชันที่ไม่มีการคืนค่า

ถ้าเขียนแบบที่ 1.1 จะได้

```
int  functionTest ( int x )
{
return  x;
}
```

จะพบว่าถ้าเขียนแบบนี้ มีการคืนค่า เพราะฉะนั้นจะต้องมีคำว่า return ซึ่งจะตามด้วยค่าที่ต้องการคืนให้เพื่อเป็นคำตอบของฟังก์ชันนั้นๆ

ถ้าเขียนแบบที่ 1.2 จะได้

```
void  functionTest ()
{
}
```

จะพบว่าถ้าเขียนแบบนี้ ไม่มีการคืนค่า เพราะฉะนั้นไม่ต้องมีคำว่า return และชนิดข้อมูลของ output จะต้องเป็นคำว่า void เท่านั้น

5. การแสดงผลทางหน้าจอกับการรับค่าทางคีย์บอร์ด

ชุดคำสั่งที่ใช้ในการแสดงผลทางหน้าจอคือ ชื่อ object ชื่อ cout ส่วนคำสั่งที่ทำการแสดงค่า คือ << เพราะฉะนั้น จะเขียนได้ว่า

```
cout<< " ข้อความ"  ;
```

การ ใช้ << หนึ่งครั้งคือการสั่งให้แสดงผลทางหน้าจอหนึ่งข้อมูล เพราะฉะนั้นถ้าข้อมูลคนละชนิดกันต้องใช้คนละ << เช่น cout<<" ขัอความ " << 1.0<<endl;

และคำสั่งที่ใช้ขึ้นบรรทัดใหม่คือ endl หรือ "\n"

ชุดคำสั่งที่ใช้ในการรับค่าข้อมูลจากทางคีย์บอร์ดคือ ชื่อ object ชื่อ cin ส่วนคำสั่งที่ทำการแสดงค่า คือ >> เพราะฉะนั้นจะเขียนได้ว่า

```
cin>> ชื่อตัวแปรที่จะเก็บค่าข้อมูล
```

การใช้ >> หนึ่งครั้งคือการเก็บค่าข้อมูลลงในตัวแปรหนึ่งครั้ง ถ้ามีการเก็บหลายครั้งก็ใช้คนละ >> เช่น cin>> x>>y;

ข้อควรจำ คำสั่ง cin , cout อยู่ใน library (ที่เก็บรวบรวมคำสั่งที่มีอยู่แล้วในภาษาซีพลัสพลัส) ที่ชื่อว่า iostream ดังนั้นจำเป็นต้องระบุชื่อ library ลงไปในโค้ดโปรแกรมทุกครั้งที่มีการใช้ cin,cout

การระบุชื่อ library ใช้คำว่า #include< ชื่อ library > เช่น #include<iostream>

บทที่ 4
เริ่มต้นออกแบบโปรแกรม

ในการออกแบบโปรแกรม เปรียบได้เหมือนกันกับการแก้โจทย์คณิตศาสตร์ เพราะฉะนั้นต้องรู้ก่อนว่า

1. โจทย์ต้องการให้หาอะไร
2. ต้องใส่อะไรเข้าไปในโจทย์
3. อะไรคือคำตอบของโจทย์ข้อนี้

การเขียนโปรแกรมสำหรับผู้ที่เริ่มต้น ควรจะมีลำดับการคิดที่มีประสิทธิภาพ การเขียนโปรแกรมจึงจะเป็นได้อย่างง่ายดาย ดังนั้นควรมีลำดับการคิดดังนี้

การคิดขั้นตอนที่ 1 คิดแบบกว้างที่สุดตามโจทย์กำหนด

สมมติว่า โจทย์ให้เขียนโปรแกรมเปลี่ยนอุณหภูมิในหน่วยองศาเซลเซียสไปเป็นหน่วยองศาเคลวิน

1. จุดประสงค์ (purpose) : เปลี่ยนอุณหภูมิในหน่วยองศาเซลเซียสไปเป็นหน่วยองศาเคลวิน
2. ข้อมูลที่ต้องใช้ในโปรแกรม(input) : อุณหภูมิในหน่วยองศาเซลเซียส
3. คำตอบของโปรแกรม(output) : อุณหภูมิในหน่วยองศาเคลวิน

การคิดขั้นตอนที่ 2 ระบุความต้องการของโจทย์ให้อยู่ในขอบเขตของภาษาที่เขียน

จาก สามสิ่งที่ได้วิเคราะห์ไว้ ก็ต้องมาคิดตามภาษาที่ใช้เขียนว่า ถ้าเขียนด้วยภาษานั้นๆ จะต้องมีข้อกำหนดกฎเกณฑ์อะไรบ้าง เพราะฉะนั้นจากหลักการเขียนฟังก์ชันในภาษาซีพลัสพลัส

```
ชนิดข้อมูลของ output   ชื่อฟังก์ชัน  (  ชนิดข้อมูลของ input )
{


}
```

สามารถระบุลงไปแต่ละส่วนได้ว่า

ชื่อฟังก์ชัน มาจาก จุดประสงค์ (purpose) จะได้ C_to_K

ชนิดข้อมูลของ input มาจาก ข้อมูลที่ต้องใช้ในโปรแกรม(input) : อุณหภูมิในหน่วยองศาเซลเซียส

อุณหภูมิก็สามารถเป็นทศนิยมได้ ก็จะได้ double

ชนิดข้อมูลของ output มาจาก คำตอบของโปรแกรม(output) : อุณหภูมิในหน่วยองศาเคลวิน

อุณหภูมิก็สามารถเป็นทศนิยมได้ ก็จะได้ double

เพราะฉะนั้นจากขั้นตอนที่ 2 ก็จะได้

4. โครงสร้างฟังก์ชัน (contract) : C_to_K double->double

การคิดขั้นตอนที่ 3 สมการที่ใช้ในโปรแกรม

ต้องคิดว่าสมการที่จะใช้คืออะไร และถ้าใส่ input เท่านี้จะได้ output ออกมาเท่าไหร่

จากความรู้เดิมการเปลี่ยนจาก องศาเซลเซียสไปเป็นเคลวิน ก็คือ องศาเคลวิน = องศาเซลเซียส + 273

ดังนั้นถ้ามี input คือองศาเซลเซียสเท่านี้จะได้ องศาเคลวินเท่าไหร่ ก็จะได้

5. ตัวอย่างโปรแกรม (example) :

C_to_K(0) = 273.0

C_to_K(50.0) = 323.0

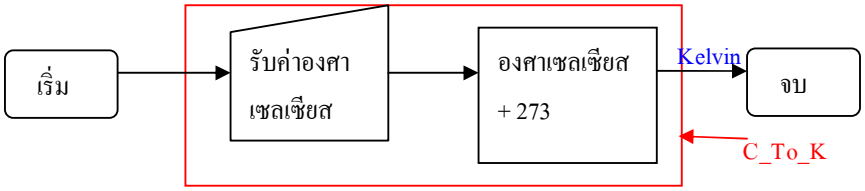
C_to_K(100.0) = 373.0

ทดลองทำสัก 3-4 ตัวอย่างเพื่อที่จะได้แน่ใจในคำตอบของโปรแกรมเมื่อเขียนเสร็จทดลองว่าโปรแกรมให้คำตอบถูกหรือไม่ด้วยการใส่ input เท่ากับตัวอย่างที่เราคิดด้วยตนเอง แล้วดูคำตอบว่าตรงกันหรือไม่

การคิดขั้นตอนที่ 4 ขั้นตอนของโค้ดของโปรแกรมอย่างละเอียด

เราจะเขียนลำดับของการทำงานของโปรแกรมอย่างละเอียดโดยเขียนโดยคำนึงถึงโครงสร้างภาษาและข้อจำกัดของภาษามากขึ้น เขียนออกมาในรูปแบบของแผนผังอย่างละเอียด จะได้

6. แผนผังการทำงานของโปรแกรม (flowchart)



และขั้นตอนสุดท้ายคือการเขียนโค้ดโปรแกรมจริงๆ ซึ่งก็มีสองส่วนคือ

ส่วนของฟังก์ชันที่เราออกแบบไว้

และส่วนของฟังก์ชันเพื่อที่จะให้โปรแกรมทำงานได้

7. โค้ดของฟังก์ชัน (function code)

```
double C_to_K (double degree )
{
return 273.0+degree;
}
```

8. โค้ดของฟังก์ชันหลัก (main function code)

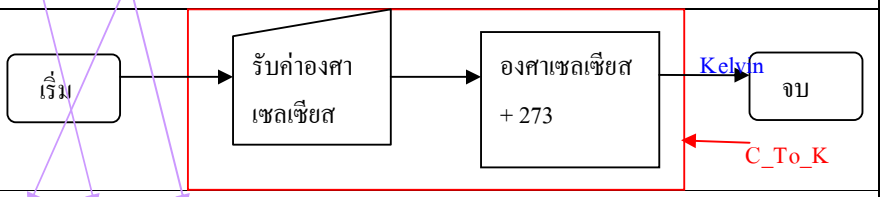
```
int main() // ต้องตั้งชื่อฟังก์ชันว่า main เท่านั้น
{
int degree;
cout<<" type number of celcius degree ";
cin>>degree;
cout<<" Kelvin degree = "<<C_to_K(degree)<<endl;
return 0; // คืนค่า 0 เพื่อบอกว่าจบโปรแกรมสมบูรณ์
}
```

นอกจากcout เพื่อโชว์คำตอบอาจเก็บค่าที่ได้จากฟังก์ชันลงในตัวแปรก่อนได้ดังนี้ double result ;

result = C_to_K(degree) ; แล้ว cout<<result;

การเรียกใช้ฟังก์ชันจะใช้ชื่อฟังก์ชัน และถ้าฟังก์ชันนั้นๆมี input ก็ต้องใส่ค่า input ลงไป จะเป็นค่าข้อมูลโดยตรงหรือชื่อตัวแปรที่มีชนิดข้อมูลตรงกับที่ฟังก์ชันกำหนด โดยการใส่ input ค่าจะเรียงลำดับตาม input ในฟังก์ชันที่เขียนไว้ กรณีฟังก์ชันที่ไม่มีการคืนค่า จะทำได้เรียกชื่อฟังก์ชันเพื่อให้ทำงานเท่านั้นเพราะไม่มีการคืนค่าใดๆกลับ

สรุปทั้ง 8 ขั้นตอนจะได้

Purpose	เปลี่ยนจากองศาเซลเซียสไปเป็นองศาเคลวิน
Input	องศาเซลเซียส
Output	องศาเคลวิน
Contract	C_to_K : double->double
Example	C_to_K (0) = 273.0 C_to_K (50) = 323.0 C_to_K (100) = 373.0
Flowchart	

Function code	double C_to_K (double degree)
	{
	return 273.0+degree;
	}

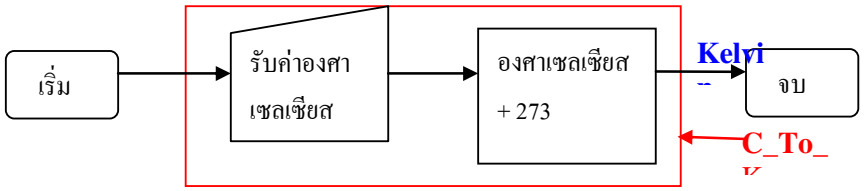
Main function code	int main()
	{
	int degree;
	cout<<" type number of celcius degree ";
	cin>>degree;
	cout<<" Kelvin degree = "<<C_to_K(degree)<<endl;
	return 0;
	}

ซึ่งในขั้นตอนทั้งหมดจะเป็นเพียงแนวความคิดของเราเท่านั้น ดังนั้นหากพิมพ์ลงไปในโปรแกรมก็จะต้องทำการใส่คอมเมนท์ไว้ด้วย เพื่อที่จะได้ไม่ถูกคอมไพล์ แต่ส่วนที่จะเป็นโค้ดที่จะคอมไพล์จริงๆก็คือขั้นตอน 2 ขั้นตอนสุดท้าย ดังนั้นใน โค้ดของสองขั้นตอนนี้จะไม่ต้องใส่คอมเมนท์

ดังนั้นโค้ดทั้งหมดของโปรแกรมจะเป็นดังรูปต่อไปนี้

ไฟล์ C_to_K.cpp

```
#include<iostream>
/*
Purpose  เปลี่ยนจากองศาเซลเซียสไปเป็นองศาเคลวิน
Input องศาเซลเซียส
Output องศาเคลวิน
Example
C_to_K (0) = 273.0
C_to_K (50) = 323.0
C_to_K (100) = 373.0
Flowchart
```




```
*/
//Function code
    double C_to_K (double degree )
    {
    return 273.0+degree;
    }
//Main function code
    int main()
    {
    int degree;
    cout<<" type number of celcius degree ";
    cin>>degree;
    cout<<" Kelvin degree = "<<C_to_K(degree)<<endl;
    return 0;
    }
```

บทที่ 5

การเขียนโปรแกรมแบบซับซ้อน

ในโปรแกรมจริงๆที่เขียนขึ้นไม่ได้มีเพียงฟังก์ชันเดียว ในหนึ่งไฟล์ .cpp อาจจะมีกี่ฟังก์ชันก็ได้ แต่จะมีฟังก์ชันที่ทำงานเสมอ เพียงฟังก์ชันเดียวคือฟังก์ชันหลัก ดังนั้นในบทนี้เราจะเรียนรู้วิธีการเขียนโปรแกรมที่มีหลายๆฟังก์ชันถูกเรียกใช้งาน

หลักการคือให้จำไว้ว่า 1 ฟังก์ชัน คือหนึ่งวัตถุประสงค์ เพราะฉะนั้น การเขียนโปรแกรมออกแบบให้มีกี่ฟังก์ชัน หรืออาจจะไม่มีฟังก์ชันเลยก็ตาม ขึ้นอยู่กับการออกแบบของผู้เขียนโปรแกรม

ถ้าสมมติให้โปรแกรมเมอร์คือคนเขียนบทละคร แล้วแต่ละฟังก์ชันก็คือแต่ละตัวละคร ซึ่งมีหน้าที่ต่างๆกันไป ละครหนึ่งเรื่อง ก็อาจจะใช้ตัวละครไม่เท่ากันก็ได้ ถึงแม้ว่าตอนจบของละครจะเหมือนกัน เพราะคนเขียนบทอาจจะให้ตัวละครคนหนึ่งทำหลายหน้าที่ อีกคนอาจจะให้ตัวละครหนึ่งทำหน้าที่เดียว จำนวนตัวละครที่ใช้ก็อาจจะไม่เท่ากัน เหมือนกับโปรแกรม ในโปรแกรมที่มีคำตอบเดียวกัน ก็อาจจะมีวิธีการเขียนออกมาไม่เหมือนกันได้

ตัวอย่างสมมติให้ ฟังก์ชันหลักเปรียบเหมือน ผู้กำกับที่เล่นละครเรื่องนั้นด้วย
แต่ละฟังก์ชัน คือ ตัวละครแต่ละตัวหน้าที่ต่างกัน
และโปรแกรมเมอร์คือคนเขียนบท
เขียนโปรแกรมการเปลี่ยนจากองศาฟาเรนไฮน์ไปเป็นองศาเคลวิน
บทละครที่ 1 (ออกแบบแบบนี้เพราะรู้ว่าสมการเปลี่ยนองศาฟาเรนไฮน์ เป็นเคลวิน คืออะไร)
มีตัวละครหนึ่งตัวคือ ผู้กำกับ ทำหน้าที่ทุกอย่าง ตั้งแต่

1. รับค่า input องศาฟาเรนไฮน์
2. คำนวณเปลี่ยนองศาฟาเรนไฮน์ไปเป็นองศาเคลวิน
3. แสดงผลคำตอบ

บทละครที่ 2 (ออกแบบแบบนี้เพราะรู้ว่าสมการเปลี่ยนองศาฟาเรนไฮน์ เป็นเคลวิน คืออะไร)
มีตัวละครสองตัว
ตัวละคร 1 คือ ผู้กำกับ ทำหน้าที่

1. รับค่า input องศาฟาเรนไฮน์
2. สั่งให้ตัวละครสองเล่น
3. แสดงผลคำตอบ

ตัวละคร 2 ทำหน้าที่

1. คำนวณเปลี่ยนองศาฟาเรนไฮน์ไปเป็นองศาเคลวิน

บทละครที่ 3 (ออกแบบแบบนี้เพราะไม่รู้ว่าสมการเปลี่ยนองศาฟาเรนไฮน์ เป็นเคลวิน คืออะไร แต่รู้ว่าเปลี่ยนจากองศาฟาเรนไฮน์เป็นองศาเซลเซียสคืออะไร และ เปลี่ยนจากองศาเซลเซียสเป็นเคลวินคืออะไร)

มีตัวละครสามตัว

ตัวละคร 1 คือ ผู้กำกับ ทำหน้าที่

1. รับค่า input องศาฟาเรนไฮน์
2. สั่งให้ตัวละครสองเล่น
3. สั่งให้ตัวละครสามเล่น
4. แสดงผลคำตอบ

ตัวละคร 2 ทำหน้าที่

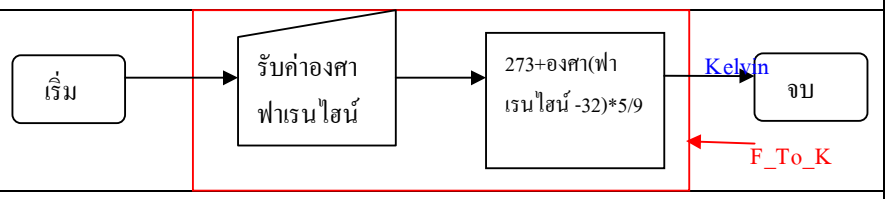
1. คำนวณเปลี่ยนองศาฟาเรนไฮน์ไปเป็นองศาเซลเซียส

ตัวละคร 3 ทำหน้าที่

1. คำนวณเปลี่ยนองศาเซลเซียสไปเป็นองศาเคลวิน

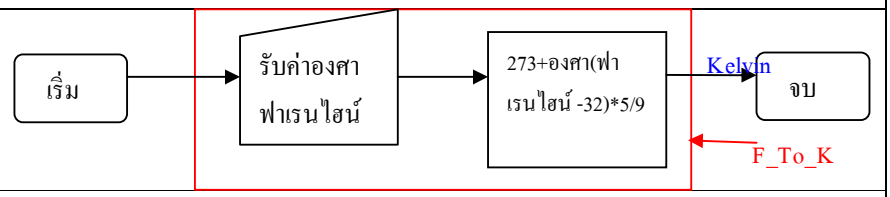
ถ้าเขียนโปรแกรมตามแบบบทละครทั้งสามแบบจะได้ดังนี้

บทละครที่ 1 มีตัวละครเดียวคือผู้กำกับ ดังนั้นก็จะมีฟังก์ชันหลักอย่างเดียว

Purpose	เปลี่ยนจากองศาองศาฟาเรนไฮน์ไปเป็นองศาเคลวิน
Input	องศาฟาเรนไฮน์
Output	องศาเคลวิน
Contract	F_to_K : double->double
Example	F_to_K (32.0) = 273.0 F_to_K (122.0) = 323.0 F_to_K (212.0) = 373.0
Flowchart	
Function code	

Main function code	int main()
	{
	int degree;
	cout<<" type number of farenhite degree ";
	cin>>degree;
	cout<<" Kelvin degree = "<< 273.0+((degree-32.0)*5/9)<<endl;
	return 0;
	}

บทละครที่ 2 มีตัวละคร 2 ตัว

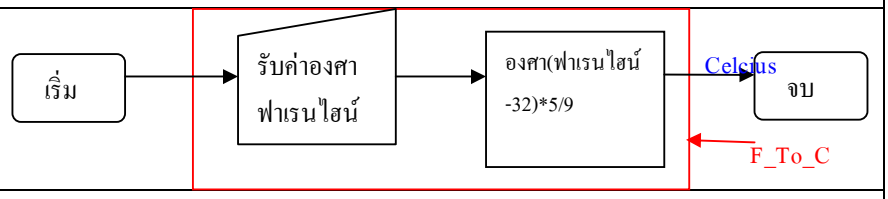
Purpose	เปลี่ยนจากองศาองศาฟาเรนไฮน์ไปเป็นองศาเคลวิน
Input	องศาฟาเรนไฮน์
Output	องศาเคลวิน
Contract	F_to_K : double->double
Example	F_to_K (32.0) = 273.0 F_to_K (122.0) = 323.0 F_to_K (212.0) = 373.0
Flowchart	

Function code	double F_to_K (double degree)
	{
	return 273.0+((degree-32.0)*5/9);
	}

Main function code	int main()
	{
	int degree;
	cout<<" type number of farenhite degree ";
	cin>>degree;
	cout<<" Kelvin degree = "<<F_to_K(degree)<<endl;
	return 0;
	}

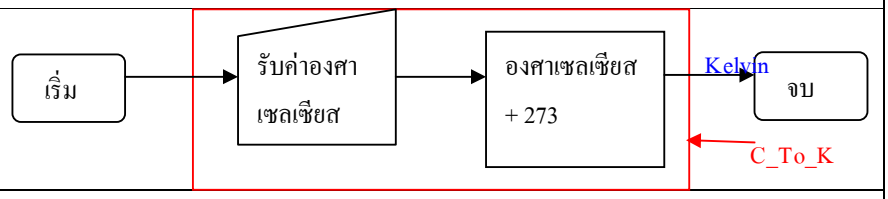
บทละครที่ 3 มีตัวละครสามตัว

ออกแบบให้กับตัวละครที่ 2

Purpose	เปลี่ยนจากองศาฟาเรนไฮน์ไปเป็นองศาเซลเซียส
Input	องศาฟาเรนไฮน์
Output	องศาเซลเซียส
Contract	F_to_C : double->double
Example	F_to_C (32.0) = 0.0 F_to_C (122.0) = 50.0 F_to_C (212.0) = 100.0
Flowchart	

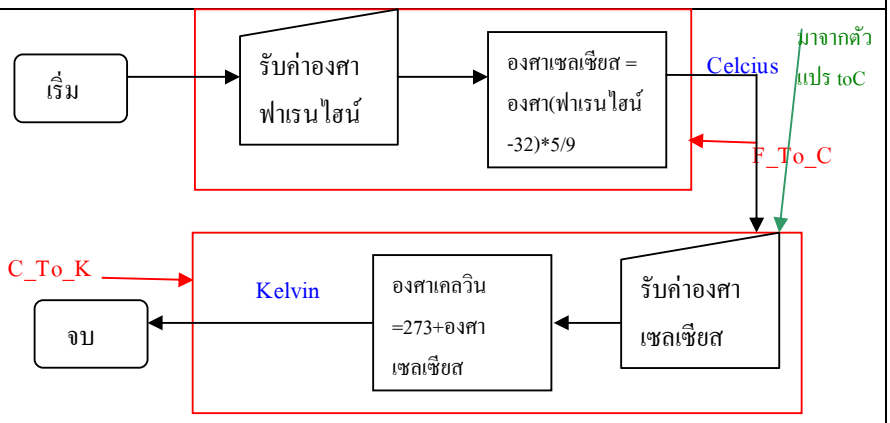
Function code	double F_to_C (double degree)
	{
	return (degree-32.0)*5/9;
	}

ออกแบบให้กับตัวละครที่ 3

Purpose	เปลี่ยนจากองศาเซลเซียสไปเป็นองศาเคลวิน
Input	องศาเซลเซียส
Output	องศาเคลวิน
Contract	C_to_K : double->double
Example	C_to_K (0) = 273.0 C_to_K (50) = 323.0 C_to_K (100) = 373.0
Flowchart	

Function code	double C_to_K (double degree)
	{
	return 273.0+degree;
	}

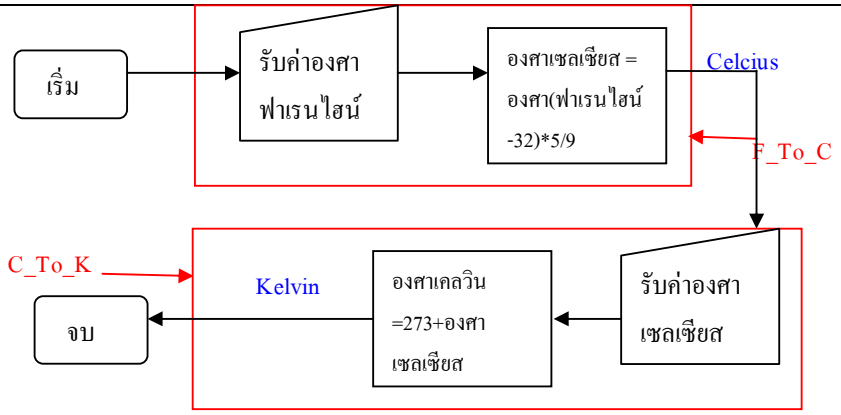
ออกแบบให้กับตัวละครที่ 1 คือผู้กำกับ

Purpose	เปลี่ยนจากองศาองศาฟาเรนไฮน์ไปเป็นองศาเคลวิน
Input	องศาฟาเรนไฮน์
Output	องศาเคลวิน
Contract	F_to_K : double->double
Example	F_to_K (32.0) = 273.0 F_to_K (122.0) = 323.0 F_to_K (212.0) = 373.0
Flowchart	

Main function code	int main()
	{
	int degree;
	cout<<" type number of farenhite degree ";
	cin>>degree;
	int toC=F_to_C(degree);
	cout<<" Kelvin degree = "<<C_to_K(toC)<<endl;
	return 0;
	}

แบบนี้คือทำการเปลี่ยนค่าองศาฟาเรนไฮน์เป็นองศาเซลเซียสก่อน เก็บไว้ในตัวแปร ชื่อ toC และทำการเปลี่ยนองศาเซลเซียสเป็นองศาเคลวินอีกทีนึง โดยที่มี input คือตัวองศาเซลเซียสในตัวแปรชื่อ toC

หรือ

Purpose	เปลี่ยนจากองศาองศาฟาเรนไฮน์ไปเป็นองศาเคลวิน
Input	องศาฟาเรนไฮน์
Output	องศาเคลวิน
Contract	F_to_K : double->double
Example	F_to_K (32.0) = 273.0 F_to_K (122.0) = 323.0 F_to_K (212.0) = 373.0
Flowchart	

Main function code	int main()
	{
	int degree;
	cout<<" type number of farenhite degree ";
	cin>>degree;
	cout<<" Kelvin degree = "<<C_to_K(F_to_C(degree))<<endl;
	return 0;
	}

แบบนี้คือทำการเปลี่ยนค่าองศาฟาเรนไฮน์เป็นองศาเซลเซียสก่อน แต่ไม่เก็บค่าตัวแปร แต่ส่งให้เป็น input ของฟังก์ชันที่ ทำการเปลี่ยนองศาเซลเซียสเป็นองศาเคลวินอีกทีนึง

สรุป จะเห็นว่าโปรแกรมเดียวกัน แต่เขียนได้ไม่เหมือนกันขึ้นอยู่กับการออกแบบโปรแกรมของผู้เขียนโปรแกรมแต่ละคน สำหรับวิธีการออกแบบที่มีการสร้างหลายฟังก์ชันจุดประสงค์คือ

1. เพื่อให้ง่ายต่อการแก้ไขโปรแกรมในอนาคต
2. เพื่อเป็นการนำเอาฟังก์ชันมาใช้ได้อีก (Reuse)
3. เพื่อสะดวกต่อการพัฒนา โดยเฉพาะการพัฒนาโปรแกรมที่มีขนาดใหญ่

บทที่ 6

การเขียนโปรแกรมแบบมีเงื่อนไข

จากบทเรียนก่อนหน้า คำตอบของโปรแกรมที่ได้มักจะเป็นคำตอบที่มาจากสมการเดียวหรือมาจากรูปแบบๆเดียว ซึ่งในความเป็นจริงแล้ว คำตอบของโปรแกรมที่ได้ อาจจะไม่ได้มาจากสมการเดียวกันหรือมาจากรูปแบบเดียวกันก็ได้ เพราะฉะนั้นเมื่อคำตอบของโปรแกรมมาจากสมการหรือรูปแบบที่มากกว่า 1 รูปแบบ ต้องใช้เทคนิคการเขียนโปรแกรมแบบมีเงื่อนไข

การเขียนโปรแกรมแบบมีเงื่อนไขจะเข้ามาช่วยในการตัดสินใจว่า
ถ้า input นี้ จะต้องใช้สมการนี้หรือรูปแบบคำตอบนี้
ถ้า input อีกอันนึงก็ใช้อีกรูปแบบนึง
เริ่มต้น เราจะเรียนรู้การเขียนโปรแกรมแบบมีเงื่อนไขสำหรับคำตอบสองรูปแบบก่อน

โครงสร้างภาษาซีพลัสพลัสที่ใช้ก็คือ

```
if( เงื่อนไข )
{
สิ่งที่จะให้ทำ ฯลฯ
}else
{
สิ่งที่จะให้ทำ ฯลฯ
}
```

ความหมายก็คือ
ถ้า เป็นอย่างนี้
{
จะให้ทำอะไร
}
ถ้าไม่ใช่กรณีแรก แล้ว(พูดง่ายๆก็คือไม่ใช่ตามเงื่อนไข)
{
จะให้ทำอะไรอีก
}

หมายเหตุ โจทย์ที่มักจะใช้การเขียนโปรแกรมแบบมีเงื่อนไขคือ โจทย์ที่มักจะมีคำว่า หรือไม่ , ตรวจสอบ , แบ่งประเภท ฯลฯ

เครื่องหมายที่ใช้กับการเปรียบเทียบเงื่อนไขต่างๆมีดังนี้

= =	เท่ากัน
!=	ไม่เท่ากัน
>	มากกว่า
<	น้อยกว่า
> =	มากกว่าหรือเท่ากับ
< =	น้อยกว่าหรือเท่ากับ

เครื่องหมายที่เกี่ยวข้องกับประพจน์ และค่าประพจน์

เครื่องหมาย && (AND)

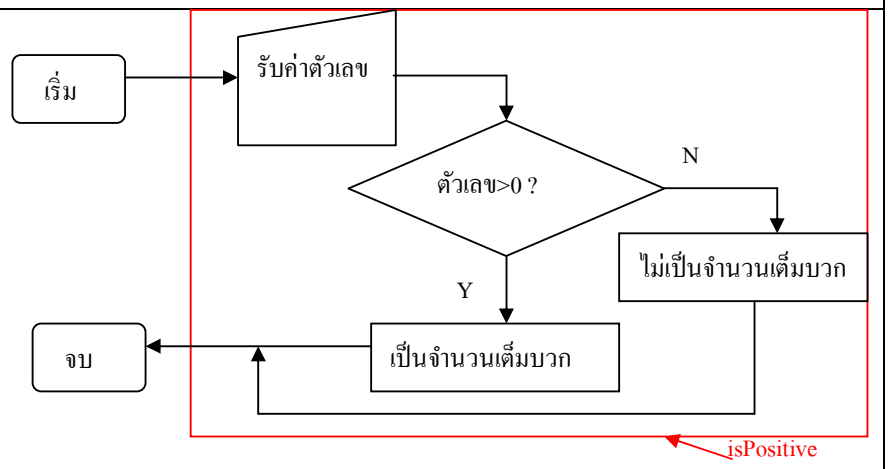
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

เครื่องหมาย || (OR)

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

เครื่องหมาย ! (NOT)

a	!a
true	false
false	true

Purpose	ตรวจสอบว่าตัวเลขที่ได้รับเป็นเลขจำนวนเต็มบวกหรือไม่
Input	ตัวเลข
Output	เป็นจำนวนเต็มบวกหรือไม่
Contract	isPositive int->bool
Example	isPositive (32) = true isPositive (-5) = false isPositive (0) = false
Flowchart	

Function code	bool isPositive (int number)
	{
	if(number>0)
	{
	return true;
	}else
	{
	return false;
	}
	}

Main function code	int main()
	{
	int number;
	cout<<"type a number for checking positive number ";
	cin>>number;
	cout<<"your number is positive ,right? "<<isPositive(number)<<endl;
	return 0;
	}

หมายเหตุ การแสดงคำตอบจะออกมาในรูป 0 กับ 1 เพราะ bool ใน C++ จะแสดง 0 หมายถึง false และ 1 หมายถึง true

การเขียนโปรแกรมแบบมีเงื่อนไขสำหรับคำตอบหลายรูปแบบ

โครงสร้างภาษาซีพลัสพลัสที่ใช้ก็คือ

```
if( เงื่อนไข )
{
สิ่งที่จะให้ทำ ฯลฯ
}else if(เงื่อนไข )
{
สิ่งที่จะให้ทำ ฯลฯ
} else if(เงื่อนไข )
{
สิ่งที่จะให้ทำ ฯลฯ
}
.
.
.
else
{
สิ่งที่จะให้ทำ ฯลฯ
}
```

ความหมายก็คือ

ถ้าเป็นอย่างนี้

{

จะให้ทำอะไร

}

ถ้าไม่ใช่กรณีแรก แล้ว(พูดง่ายๆก็คือไม่ใช่ตามเงื่อนไข)

{

จะให้ทำอะไรอีก

}

ถ้าไม่ใช่กรณีแรกและกรณีที่สอง แล้ว(พูดง่ายๆก็คือไม่ใช่ตามเงื่อนไขสองอันแรก)

{

จะให้ทำอะไรอีก

}

.

.

.

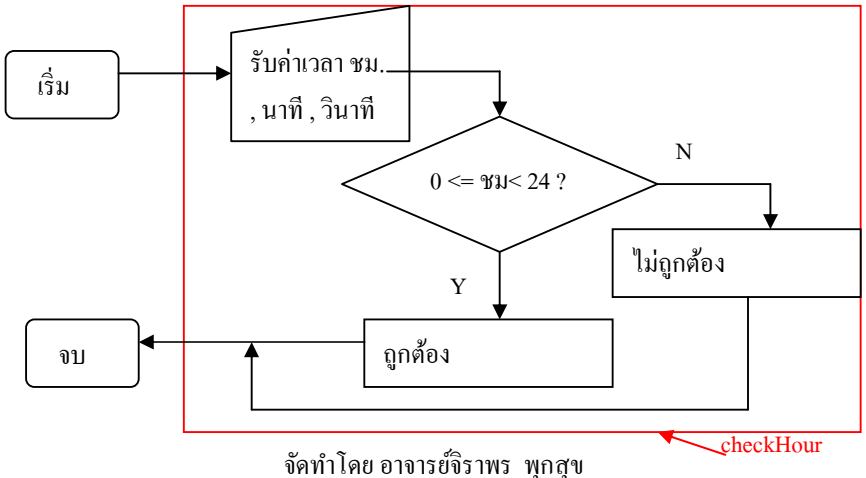
ถ้าไม่ใช่กรณีทุกกรณีข้างต้น

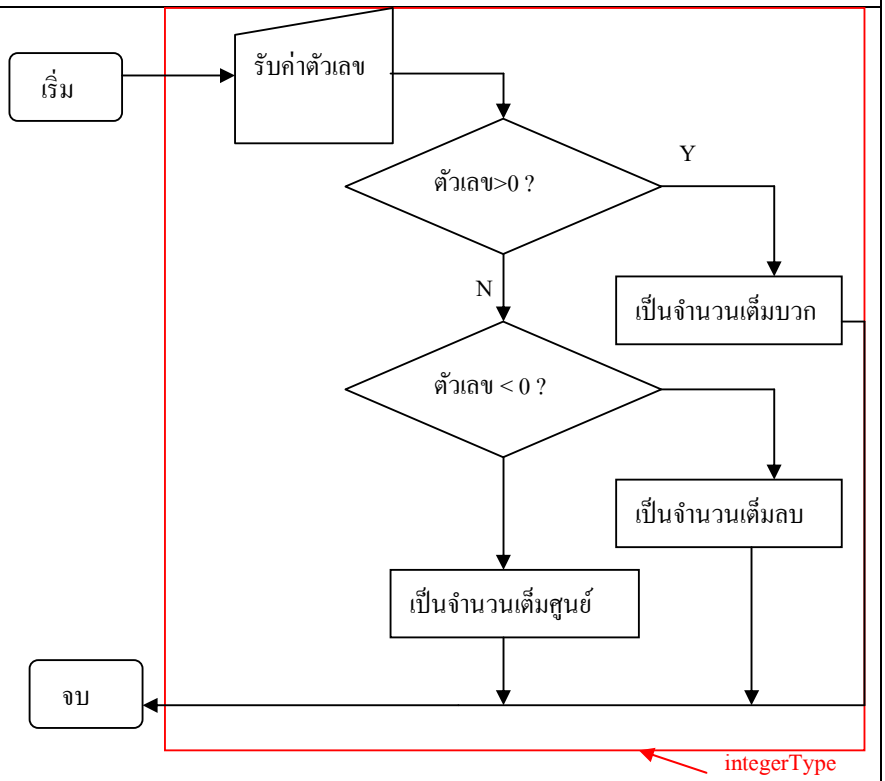
{

จะให้ทำอะไรอีก

}

หมายเหตุ จำนวนเงื่อนไขก็จะมีจำนวนเท่ากับรูปแบบของคำตอบ



Purpose	ตรวจสอบว่าตัวเลขที่ได้รับเป็นเลขจำนวนเต็มบวก จำนวนเต็มลบ หรือ จำนวนเต็มศูนย์
Input	ตัวเลข
Output	จำนวนเต็มบวก หรือ จำนวนเต็มลบ หรือ จำนวนเต็มศูนย์
Contract	integerType int->char
Example	integerType (32) = + integerType (-5) = - integerType (0) = 0
Flowchart	

Function code	char integerType (int number)
	{
	if(number> 0)
	{
	return '+';
	}else if (number < 0)
	{
	return '-';
	}else
	{

	return '0'; } }

Main function code	int main()
	{
	int number;
	cout<<"type a number for checking type of number ";
	cin>>number;
	cout<<"your number is "<<integerType(number)<<endl;
	return 0;
	}

บทที่ 7
การเขียนโปรแกรมแบบมีการทำซ้ำ (Recursive programming)

ในการเขียนโปรแกรมนอกจากที่คอมไพล์เลอร์จะทำงานโดยอ่านชุดคำสั่งเรียงตามบรรทัดตามลำดับแล้ว ยังสามารถให้คอมไพล์เลอร์วนอ่านซ้ำในบางจุดที่เรากำหนดไว้ได้ ซึ่งการเขียนโปรแกรมแบบมีการทำซ้ำนี้มักจะใช้ในกรณีที่จะต้องมีการคำนวณสมการนั้นหลายๆครั้ง หรือเกิดการทำงานในรูปแบบเดิมหลายๆครั้ง ทำให้ไม่ต้องเขียนหลายบรรทัด

อย่างเช่นถ้าต้องการเขียนเลข 1 ทั้งหมด 5 บรรทัด ก็อาจจะต้องเขียนโค้ดดังนี้

```
int main()
{
cout<<" 1 "<<endl;
cout<<" 1 "<<endl;
cout<<" 1 "<<endl;
cout<<" 1 "<<endl;
cout<<" 1 "<<endl;
return 0;
}
```

ซึ่งจะพบว่ามีการทำงานแบบเดิมซ้ำๆถึง ห้าครั้ง

หรือเช่น ถ้าต้องการบวกเลขตั้งแต่ 1 ถึง 5 ก็จะต้องเขียนโค้ดดังนี้

```
int main()
{
cout << 1+2+3+4+ 5 <<endl;
return 0;
}
```

ถ้าดูลำดับการเปลี่ยนแปลงของตัวเลขก็จะพบว่า

1+2 = 1+ (1+1)

+3 = +(2+1) หรือ = + (1+1+1)

+4 = + (3+1) หรือ = +(1+1+1+1)

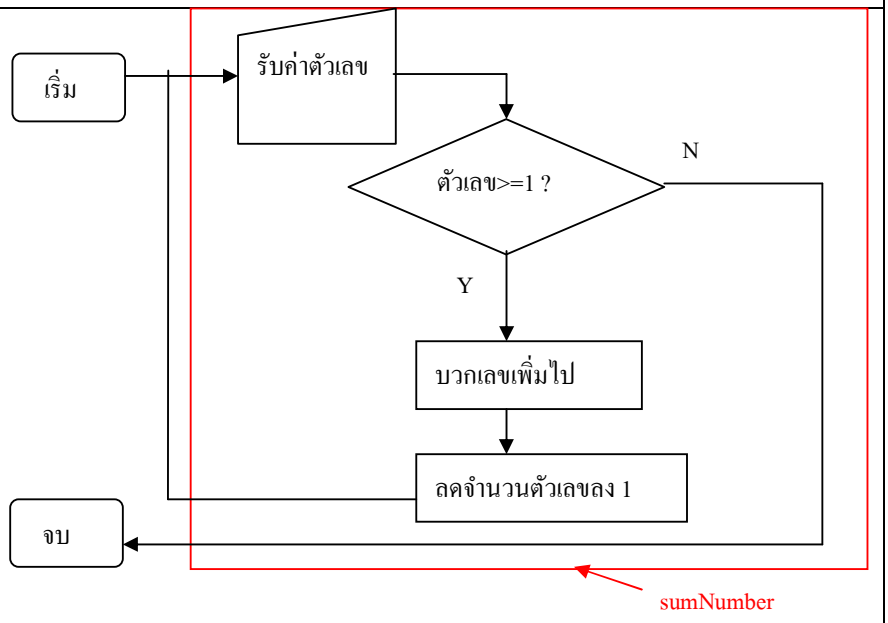
+5 = + (4+1) หรือ = + (1+1+1+1+1)

จะพบว่าการบวกเลขตัวต่อมา คือเลขตัวก่อนหน้าบวกอีก 1

เพราะฉะนั้นการทำงานซ้ำก็จะเกิดขึ้นกับการเปลี่ยนแปลงที่เปลี่ยนอย่างเป็นลำดับขั้นตอน

สำหรับการเขียนโปรแกรมแบบมีการทำซ้ำที่เราจะเขียนกันคือ เขียนแบบ recursive ก็คือการวนซ้ำใช้งานฟังก์ชันตัวเอง จนกว่าจะหมดเงื่อนไข สำหรับวิธีการคิดมีสองจุดสำคัญคือ

1. จะต้องมีจุดจบของการทำซ้ำว่า เงื่อนไขใดที่จะเป็นตัวกำหนดการจบรอบการทำงานทั้งหมด
2. จะต้องมีสมการที่จะถูกทำซ้ำ

Purpose	คำนวณหาผลรวมของเลขจำนวนเต็มบวก ตั้งแต่ 1 ถึงเลขที่ต้องการ
Input	ตัวเลข
Output	ผลรวมของเลขจำนวนเต็มบวก ตั้งแต่ 1 ถึงเลขที่ต้องการ
Contract	sumNumber int->int
Example	sumNumber (2) = 1+2=3 sumNumber (1) = 1 sumNumber (5) = 1+2+3+4+5 =15
Flowchart	

Function code	int sumNumber (int number)
	{
	if(number >= 1)
	{
	return number + sumNumber(number - 1);
	}else
	{
	return 0 ;
	}
	}

Main function code	int main()
	{
	int number;
	cout<<"type a number ";
	cin>>number;
	cout<<"sum of number is "<<sumNumber(number)<<endl;
	return 0;
	}

จะเห็นว่าตรงจุดที่โปรแกรมจะจบ ค่าที่ return คือ เลขตัวเดียวไม่มีการเรียกให้ฟังก์ชันนั้นทำงานซ้ำอีก แต่ต่างจากตรงจุดที่ต้องทำซ้ำ จะมีการ return การบวกเลขตัวนั้นเข้าไปเป็นผลลัพธ์ และเรียกให้ฟังก์ชันทำงานซ้ำอีกรอบ return คือการ return ไปยังจุดก่อนหน้าที่จะมาตรงนี้

ดังนั้นภาพการทำงานของคอมไพเลอร์ในแต่ละรอบจะเป็นดังนี้

สมมติว่า input คือ 5

รอบที่ 1 input = 5 ⟶ return 5 + sumNumber(5 - 1);

รอบที่ 2 input = 4 ⟶ return 4 + sumNumber(4 - 1);

รอบที่ 3 input = 3 ⟶ return 3 + sumNumber(3 - 1);

รอบที่ 4 input = 2 ⟶ return 2 + sumNumber(2 - 1);

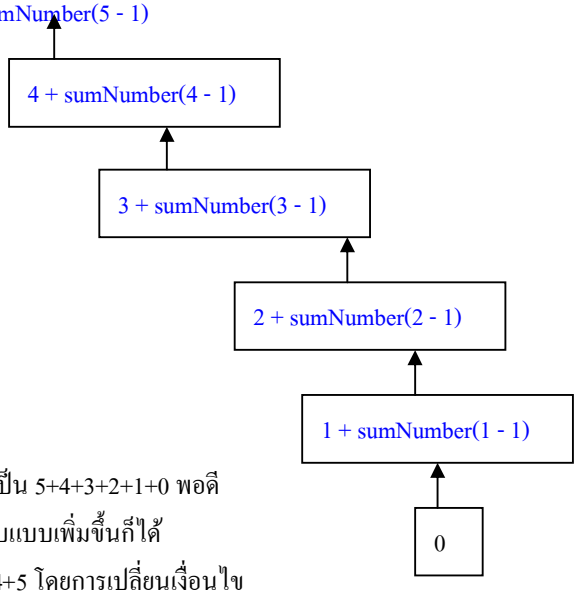
รอบที่ 5 input = 1 ⟶ return 1 + sumNumber(1 - 1);

รอบที่ 6 input = 0 ⟶ return 0 ;

คำตอบจะออกมาก็ต่อเมื่อการบวกสมบูรณ์

เพราะฉะนั้นต้องทำครบหกรอบก่อนการบวกถึงจะสมบูรณ์ได้เป็น

return 5 + sumNumber(5 - 1)



รวมแล้วก็จะเป็น 5+4+3+2+1+0 พอดี

หรืออาจจะนับแบบเพิ่มขึ้นก็ได้

เป็น 1+2+3+4+5 โดยการเปลี่ยนเงื่อนไข

บทที่ 8
การเขียนโปรแกรมแบบมีการทำซ้ำ (Loop programming)

นอกจากที่เราใช้การ recursive function เพื่อการทำซ้ำแล้ว เราจะพบว่า การใช้ recursive นี้จะไม่สามารถประกาศตัวแปรในฟังก์ชันเพื่อจำค่าได้ เพราะว่าทุกครั้งที่มีการเรียกใช้ฟังก์ชันอีก ก็จะต้อง reset ค่าตัวแปรให้เป็นค่าดั้งเดิม ดังนั้น มีชุดคำสั่งประเภทหนึ่งที่ทำให้เกิดการเขียนโปรแกรมแบบมีการทำซ้ำได้
ซึ่งมีทั้งหมดสามโครงสร้างด้วยกัน คือ

1. while
2. do while
3. for

โครงสร้าง while มีดังนี้

```
while(เงื่อนไข)
{
สิ่งที่จะทำซ้ำ
}
```

ซึ่งโครงสร้าง while จะทำซ้ำก็ต่อเมื่อ เงื่อนไขเป็นจริง

โครงสร้าง do while มีดังนี้

```
do
{
สิ่งที่จะทำซ้ำ
} while(เงื่อนไข) ;
```

ซึ่งโครงสร้าง do while จะทำก่อนหนึ่งรอบ และค่อยเช็คเงื่อนไขจะทำซ้ำรอบใหม่ก็ต่อเมื่อ เงื่อนไขเป็นจริง

โครงสร้าง for มีดังนี้

```
for(ค่าเริ่มต้น ; เงื่อนไข ; การเพิ่ม/ลดจำนวนรอบ)
{
สิ่งที่จะทำซ้ำ
}
```

ซึ่งโครงสร้าง for จะทำซ้ำก็ต่อเมื่อ เงื่อนไขเป็นจริง

การเขียนโปรแกรมด้วย 3 โครงสร้างนี้จะประกอบไปด้วยส่วนสำคัญ 4 จุดด้วยกันคือ

1. กำหนดตัวนับรอบเริ่มต้น (จะถูกใช้เพียงครั้งแรกครั้งเดียวเท่านั้น)
2. เงื่อนไขที่จะใช้ในการนับรอบ
3. สิ่งที่จะทำซ้ำ (จะมีกี่บรรทัดก็ได้ อยากให้อะไรซ้ำบ้างก็สั่งๆไป)
4. การเพิ่ม / ลด จำนวนรอบ

ดังนั้นถ้านำโครงสร้างทั้งสามมาเขียนจะพบว่ามีโครงสร้างเหมือนกัน ดังนี้

```
1. กำหนดตัวนับรอบเริ่มต้น
while(2. เงื่อนไข)
{
3. สิ่งที่จะทำซ้ำ
4. เพิ่ม/ลดจำนวนรอบ
}
```

```
1. กำหนดตัวนับรอบเริ่มต้น
do
{
3. สิ่งที่จะทำซ้ำ
4. เพิ่ม/ลดจำนวนรอบ
} while(2. เงื่อนไข);
```

```
for(1. กำหนดตัวนับรอบ ;2. เงื่อนไข ;4. เพิ่ม/ลดจำนวนรอบ )
{
3. สิ่งที่จะทำซ้ำ
}
```

สำหรับลำดับการทำงานของคอมไพเลอร์ จะเป็นดังนี้

Loop while กับ loop for จะมีการทำงานเหมือนกันคือ เรียงตามลำดับข้อได้เลย

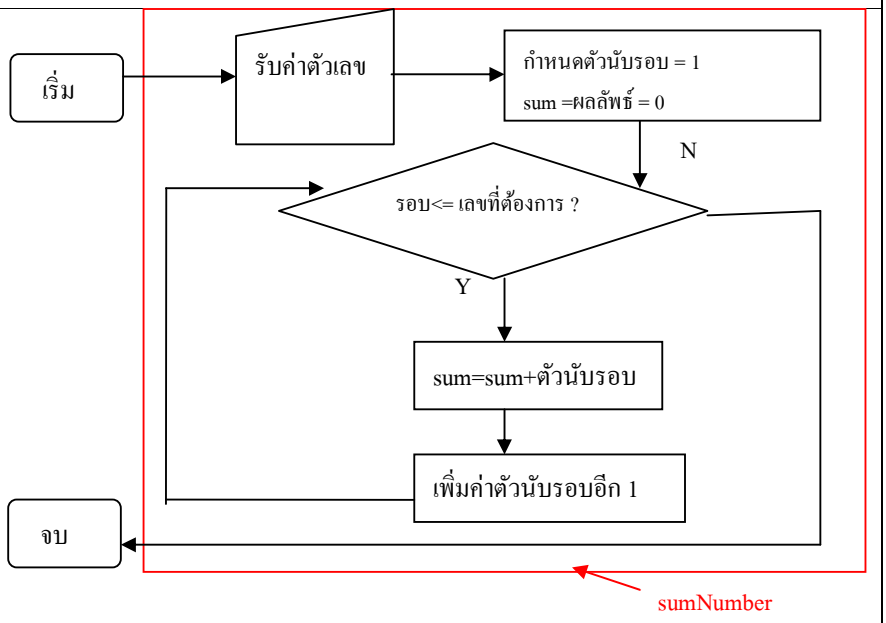
อันดับแรก เริ่มต้นโดยกำหนดตัวนับรอบก่อนว่าให้เริ่มต้นมีค่าเท่าไหร่

อันดับสอง เช็คเงื่อนไขว่าจริงหรือไม่

ถ้าเงื่อนไขเป็นจริง ก็จะทำงานในสิ่งที่ทำซ้ำ

สุดท้ายก็เพิ่ม / ลดจำนวนรอบ

สังเกตว่า ใน loop while และ do while ตัวเพิ่มลดจำนวนรอบอยู่ใน { } ซึ่งจะถูกทำซ้ำด้วย จะมาก่อนหน้า ข้อ 3 คือสิ่งที่จะทำซ้ำก็ได้หรือข้างหลังก็ได้ ขึ้นอยู่กับจุดประสงค์ของการเขียนโปรแกรม ไม่เหมือนกับ loop for ที่จะกำหนดตายตัวเลยว่าตัวเพิ่มลดจำนวนรอบจะถูกทำงานตอนสุดท้ายของแต่ละรอบ

Purpose	คำนวณหาผลรวมของเลขจำนวนเต็มบวก ตั้งแต่ 1 ถึงเลขที่ต้องการ
Input	ตัวเลข
Output	ผลรวมของเลขจำนวนเต็มบวก ตั้งแต่ 1 ถึงเลขที่ต้องการ
Contract	sumNumber int->int
Example	sumNumber (2) = 1+2=3 sumNumber (1) = 1 sumNumber (5) = 1+2+3+4+5 =15
Flowchart	

Function code	int sumNumber (int number)
	{
	int round=1 , sum =0;
	while(round < = number)
	{
	sum = sum + round; (บวกด้วย round เพราะ round เปลี่ยนค่าจาก 1 ถึง เลขที่กำหนดคือ number)
	round=round+1; (หรือจะเขียน round ++ ก็มีค่าเท่ากัน)
	}
	return sum;
	}
	จุดที่ 1 จุดที่ 2 จุดที่ 3 จุดที่ 4

Main function code	int main()
	{
	int number;
	cout<<"type a number ";
	cin>>number;
	cout<<"sum of number is "<<sumNumber(number)<<endl;
	return 0;
	}

บทที่ 9

โครงสร้างข้อมูลแบบ Array

จากบทเรียนที่ผ่านมาเรารู้จักกับชนิดข้อมูลแล้ว ก็คือแบ่งข้อมูลออกเป็นประเภทต่างๆ
แต่ว่าคำศัพท์ใหม่ที่เราจะเรียนคือ โครงสร้างข้อมูลนี้ ไม่เหมือนกันกับชนิดข้อมูล เพราะฉะนั้นอย่าสับสน
โครงสร้างข้อมูล คือแนวความคิดที่จะจัดรูปแบบข้อมูลให้เป็นแบบต่างๆ
ซึ่งโครงสร้างแบบ Array นี้ มีแนวคิดที่จะรวมเอาข้อมูลที่มีชนิดเดียวกัน อยู่รวมกันเป็นกลุ่มภายใต้ตัวแปรชื่อเดียวกัน แต่เพิ่มตัวบอกตำแหน่งเข้ามาเท่านั้นเอง

ตัวแปร ชื่อหนึ่งจะเก็บค่าข้อมูลได้หนึ่งค่า แต่ว่าพอตัวแปรตัวนั้นมีโครงสร้างเป็น array แล้วจะสามารถเก็บข้อมูลได้หลายค่า จะกี่ค่าก็ขึ้นอยู่กับว่าตัวแปรตัวนั้นมีขนาดเท่าไหร่ แต่ว่าถึงแม้ว่าหนึ่งตัวแปรเก็บได้หลายค่า แต่ก็ยังคงต้องมีตัวบอกตำแหน่งเพิ่มขึ้นมาเพื่อที่จะบอกว่า ค่าไหน อยู่ตรงไหน
ซึ่งตัวบอกตำแหน่งนี้ จะมีค่าเริ่มต้นตั้งแต่ 0 ถึง N-1 เมื่อให้ N คือขนาดตัวแปรที่มีโครงสร้างเป็น array
อธิบายดังรูป

ตัวแปร ชื่อ X

ตำแหน่ง 0	ตำแหน่ง 1	ตำแหน่ง 2	ตำแหน่ง 3

จาก 0 -> N -1 เมื่อ N = ขนาด อาร์เรย์ ซึ่งตอนนี้ ขนาด N = 4
เพราะฉะนั้นแสดงว่าตัวแปร X สามารถเก็บข้อมูลได้ 4 ค่า

อาเรย์หนึ่งมิติ

9.1 การประกาศตัวแปรที่มีโครงสร้างเป็นอาเรย์
ปกติ การประกาศตัวแปร ทำได้โดย
ชนิดข้อมูล ชื่อตัวแปร ;
การประกาศให้เป็นอารเรย์ ก็แค่เพิ่ม สัญญลักษณ์ [ ] ซึ่งข้างในวงเล็บคือขนาดอารเรย์นั่นเอง
จะได้
ชนิดข้อมูล ชื่อตัวแปร [ขนาดอารเรย์] ;
เช่น int x[4] ;
char y [10];
หรืออาจจะประกาศตัวแปรและกำหนดค่าให้เลยก็ได้ทำได้โดย
ชนิดข้อมูล ชื่อตัวแปร [ขนาดอารเรย์] = { ค่าของข้อมูลเรียงตามลำดับของตำแหน่งในอาเรย์ } ;
เช่น int x[4] = {1,2,3,4};

9.2 การเข้าถึงค่าข้อมูลในตัวแปรอารเรย์

ถ้าตัวแปรปกติ ก็ใช้ ชื่อตัวแปร แล้วจะกำหนดค่าให้หรือนำค่าแสดงมาดู ก็ตามแต่

แต่พอเป็นตัวแปรแบบอารเรย์ ก็เพิ่มตำแหน่งที่ต้องการเข้าไป

เช่น กำหนดค่าให้กับตัวแปรอารเรย์

```
int x[4];
x[0] = 1;
x[1]=2;
x[2]=3;
x[3]=4;
```

หรือแสดงค่าออกมา

```
cout<<x[0];
```

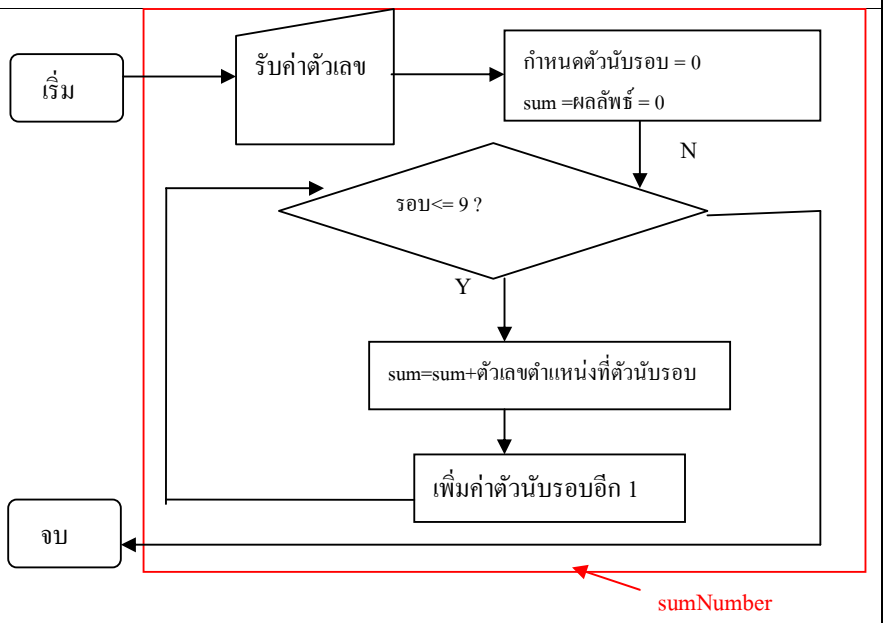
จะเห็นว่าไม่แตกต่างจากตัวแปรปกติ เพียงแค่ต้องแสดงสัญญลักษณ์ [ ] และบอกตำแหน่งข้อมูลเท่านั้นเอง

การเขียนโปรแกรมกับอารเรย์

ถ้าเราต้องการบวกค่าตัวเลขที่อยู่ในอารเรย์ ทั้งหมด ก็สามารถทำได้โดย

```
int main()
{
int x[4];
x[0] = 1;
x[1]=2;
x[2]=3;
x[3]=4;
cout<<x[0]+x[1]+x[2]+x[3]<<endl;
return 0;
}
```

ถ้ากรณีที่อาเรย์ของเรามีขนาดมากๆ อาจจะ 100 หรือ 1000 ก็จะต้องเขียนยาวมาก เพราะฉะนั้นการเขียนโปรแกรมจะไม่สะดวก และยากต่อการแก้ไข ถ้าสังเกตดูจะพบว่า ตัวแปรอารเรย์ มีตำแหน่งเริ่มจาก 0 ถึง N-1 เมื่อ N คือขนาดอารเรย์ ดังนั้นจะเห็นว่าตำแหน่งของอารเรย์เพิ่มขึ้นทีละ 1 ซึ่งตรงนี้สามารถนำการเขียนโปรแกรมแบบมีการทำซ้ำเข้ามาใช้ได้

Purpose	คำนวณหาผลรวมของเลขจำนวนเต็มใดๆ 10 ตัว
Input	ชุดตัวเลข 10 ตัว
Output	ผลรวมของเลข
Contract	sumNumber int->int
Example	sumNumber ({1,2,3,4,5,6,7,8,9,10}) = 55 sumNumber ({0,1,2,3,4,5,6,7,8,9}) = 45 sumNumber ({-5,-4,-3,-2,-1,0,1,2,3,4}) = -5
Flowchart	

Function code	int sumNumber (int number[]) (ใส่ [ ] หลังตัวแปรที่เป็นอาเรย์ทุกครั้ง)
	{
	จุดที่ 1 เริ่มที่ 0 เพราะตำแหน่งอาเรย์เริ่มที่ 0
	int round=0 , sum =0;
	จุดที่ 2 แค่ 9 เพราะตำแหน่งอาเรย์จบที่ 9 เพราะขนาด 10
	while(round < = 9)
	{
	จุดที่ 3
	sum = sum + number[round]; (บวกด้วย เลขตำแหน่งที่ round เพราะ round เปลี่ยนค่าจาก 0 ถึง เลขที่กำหนด คือ 9)
	จุดที่ 4
	round=round+1; (หรือจะเขียน round ++ ก็มีค่าเท่ากัน)
	}
	return sum;
	}

Main function code	int main()
	{
	int number[10];
	for(int i =0 ;i< 10 ;i++) (ประกาศตัวนับรอบใน loop for ก็ได้ โดยใช้ int ชื่อตัวแปร แต่ว่าตัวแปรนี้จะมองเห็นเฉพาะใน loop for ของมันเท่านั้น)
	{
	cout<<"type a number ["<<i<<"] “;
	cin>>number[i]; (เก็บค่าไว้ในตัวแปรอาเรย์ตำแหน่งที่ i)
	}
	cout<<"sum of number is "<<sumNumber(number)<<endl;
	return 0;
	}

ใช้เงื่อนไข < 10 ก็มีค่าเท่ากับ <= 9

อาเรย์สองมิติ

จากเดิมที่ตัวแปรเก็บค่าได้มากขึ้นแล้ว การทำเป็นสองมิติ คือเพิ่มมิติที่สองให้เก็บค่าได้อีก หมายความว่าจากเดิมตัวแปรเก็บได้ N ค่า ในพื้นที่ N ตำแหน่งนั้นก็จะเก็บได้อีก อันละ M ค่า
เพราะฉะนั้นตัวแปรจะเก็บค่าได้ทั้งหมด N * M ค่า เมื่อ N คือขนาดอาเรย์มิติที่หนึ่ง และ M คือขนาดอาเรย์มิติสอง

ตัวแปร ชื่อ X

ตำแหน่ง 0	ตำแหน่ง 1	ตำแหน่ง 2	ตำแหน่ง 3
ตำแหน่งที่ 0	ตำแหน่งที่ 0	ตำแหน่งที่ 0	ตำแหน่งที่ 0
ตำแหน่งที่ 1	ตำแหน่งที่ 1	ตำแหน่งที่ 1	ตำแหน่งที่ 1
ตำแหน่งที่ 2	ตำแหน่งที่ 2	ตำแหน่งที่ 2	ตำแหน่งที่ 2

มิติที่ 1 จาก 0 -> N -1 เมื่อ N = ขนาด อาร์เรย์มิติที่ 1 ซึ่งตอนนี้ ขนาด N = 4
มิติที่ 2 จาก 0 -> M -1 เมื่อ M = ขนาด อาร์เรย์มิติที่ 2 ซึ่งตอนนี้ ขนาด M = 3
เพราะฉะนั้นแสดงว่าตัวแปร X สามารถเก็บข้อมูลได้ 4 * 3 ค่า

9.3 การประกาศตัวแปรที่มีโครงสร้างเป็นอาเรย์
จากการประกาศของอารเรย์หนึ่งมิติ
การประกาศให้เป็นอารเรย์ ก็แค่เพิ่ม สัญญลักษณ์ [ ] ซึ่งข้างในวงเล็บคือขนาดอารเรย์ของมิตินั้นๆนั่นเอง
จะได้
ชนิดข้อมูล ชื่อตัวแปร [ขนาดอารเรย์มิติที่ 1] [ขนาดอารเรย์มิติที่ 2] ;
เช่น int x[4] [3];
char y [10][5];

หรืออาจจะประกาศตัวแปรและกำหนดค่าให้เลยก็ได้ทำได้โดย

ชนิดข้อมูล ชื่อตัวแปร [ขนาดอารเรย์มิติที่ 1] [ขนาดอารเรย์มิติที่ 2] = { {ค่าของข้อมูลเรียงตามลำดับของตำแหน่งในอาเรย์มิติที่สอง } , {ค่าของข้อมูลเรียงตามลำดับของตำแหน่งในอาเรย์มิติที่สอง } } ;

ซึ่งคู่ { } ข้างในแต่ละคู่คือตำแหน่งในอาเรย์มิติที่หนึ่งเรียงตามลำดับ

เช่น int x[4][3] = {{1,2,3} , {4,5,6} ,{7,8,9} , {10,11,12} };

วงเล็บ {1,2,3} คืออาเรย์มิติที่หนึ่ง ตำแหน่งที่ 0

และค่าข้างใน 1, 2, 3 คือค่าในอาเรย์มิติที่สอง เรียงตำแหน่งตามลำดับ จะได้ 1 คือตำแหน่งที่ 0 และ 2 คือตำแหน่งที่ 1 และ 3 คือตำแหน่งที่ 2

* *เปรียบเทียบง่ายๆ เหมือน ตำแหน่งมิติที่ 1 คือ บ้านเลขที่ และตำแหน่งในมิติที่สอง คือ / นั่นเอง เช่น 123/7

9.4 การเข้าถึงค่าข้อมูลในตัวแปรอารเรย์

ก็เพิ่มตำแหน่งที่ต้องการเข้าไปแต่ต้องอ้างถึงตำแหน่งทั้งในมิติที่ 1 และในมิติที่สอง

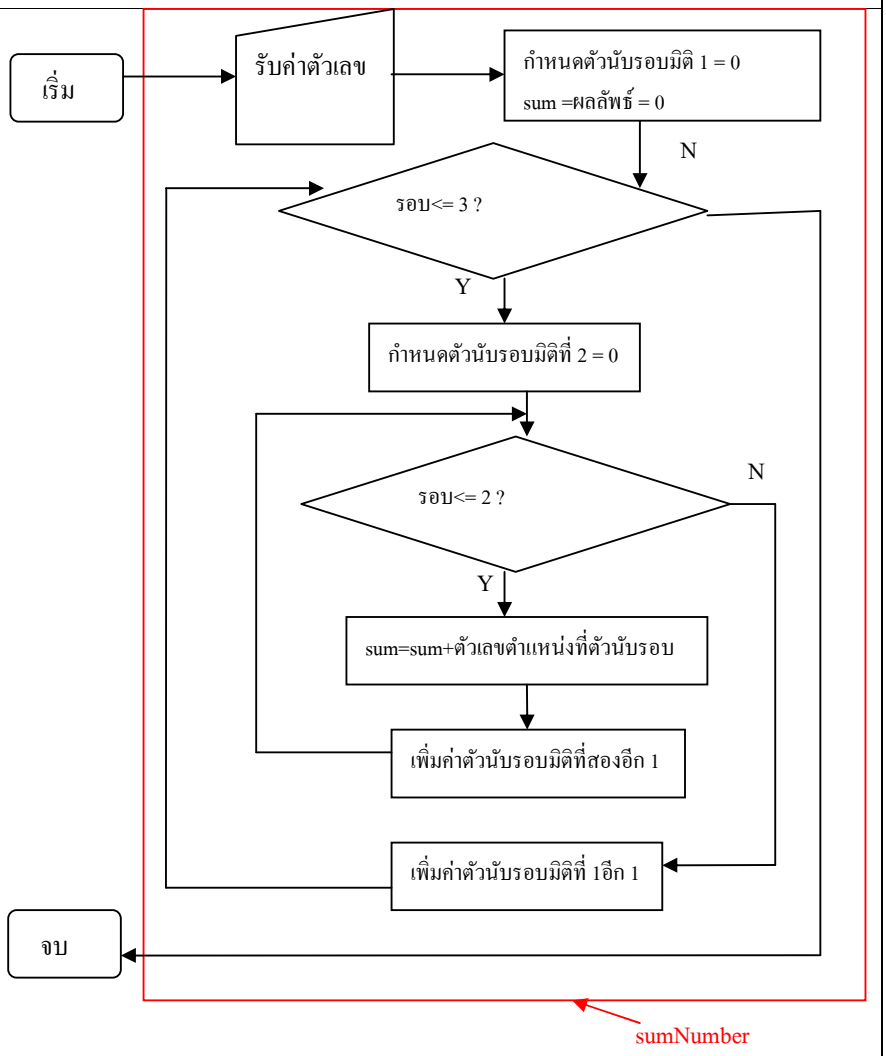
เช่น กำหนดค่าให้กับตัวแปรอารเรย์

```
int x[4][3];
x[0][0] = 1;
x[0][1] = 2;
x[0][2] = 3;
x[1] [0] =4;
x[1][1] = 5;
x[1][2] = 6;
x[2] [0] =7;
x[2][1] = 8;
x[2][2] = 9;
x[3] [0] =10;
x[3][1] = 11;
x[3][2] = 12;
```

หรือแสดงค่าออกมา

```
cout<<x[0][2];
```

ก็จะเห็นว่าเพียงแค่ต้องแสดงสัญญลักษณ์ [ ] ของทั้งสองมิติ และบอกตำแหน่งข้อมูลในทั้งสองมิติเท่านั้นเอง

Purpose	คำนวณหาผลรวมของเลขจำนวนเต็มใดๆ 12 ตัว
Input	ชุดตัวเลข 12 ตัว
Output	ผลรวมของเลข
Contract	sumNumber int->int
Example	sumNumber ({{1,2,3} , {4,5,6} ,{7,8,9} , {10,11,12} }) = 78 sumNumber ({{1,2,3} , {4,5,6} ,{7,8,9} , {10,11,0} }) = 66 sumNumber ({{1,2,3} , {4,5,6} ,{7,-8,9} , {10,11,12} }) = 70
Flowchart	

Function code

```
int sumNumber (int number[][3]) (ใส่ [ ] หลังตัวแปรที่เป็นอาเรย์ทุกครั้งและต้องใส่ขนาดของมิติที่สองด้วย)   จุดที่ 1 เริ่มที่ 0 เพราะตำแหน่งอาเรย์เริ่มที่ 0
{
        int  round1=0 , sum =0;                    จุดที่ 2 แค่ 3 เพราะตำแหน่งอาเรย์จบที่ 3 เพราะขนาดมิติที่ 1 = 4
        while(round1 < = 3)
        {                                          จุดที่ 1 เริ่มที่ 0 เพราะตำแหน่งอาเรย์เริ่มที่ 0
                int  round2=0;
        while(round2 < = 2)                        จุดที่ 2 แค่ 3 เพราะตำแหน่งอาเรย์จบที่ 2 เพราะขนาดมิติที่ 2 = 3
        {
จุดที่ 3 sum = sum + number[round1] [round2]; ( บวกด้วย เลขตำแหน่งที่ round 1 และ round2)
        round2=round2+1; (หรือจะเขียน round2 ++ ก็มีค่าเท่ากัน)
จุดที่ 4 }
        round1=round1+1; (หรือจะเขียน round1 ++ ก็มีค่าเท่ากัน)
จุดที่ 4 }
        return sum;
}
```

จุดที่ 3

Main function code

```
int main()
{                                  ใช้เงื่อนไข < 4 ก็มีค่าเท่ากับ <= 3
int number[4][3];
for(int i =0 ;i< 4 ;i++) (ประกาศตัวนับรอบใน loop for ก็ได้ โดยใช้ int ชื่อตัวแปร แต่ว่าตัวแปรนี้จะมองเห็นเฉพาะใน loop for ของมันเท่านั้น )
{
        for(int j =0 ; j< 3 ; j++) (ประกาศตัวนับรอบใน loop for ก็ได้ โดยใช้ int ชื่อตัวแปร แต่ว่าตัวแปรนี้จะมองเห็นเฉพาะใน loop for ของมันเท่านั้น )
        {
        cout<<"type a number ["<<i<<"][ "<<j<<"] ";
        cin>>number[i][j]; (เก็บค่าไว้ในตัวแปรอาเรย์ตำแหน่งที่ i j )
        }
}
cout<<"sum of  number is  "<<sumNumber(number)<<endl;
return 0;
}
```

บทที่ 10

การเขียนโปรแกรมแบบมีโครงสร้าง (struct)

จากบทเรียนที่ผ่านมาได้รู้จักกับชนิดข้อมูลทั้งหมดแล้ว เราจะพบว่า ชนิดข้อมูลอย่างหนึ่งก็เก็บค่าประเภทนึง ทีนี้ถ้าเราจะรวมเอาข้อมูลหลายๆอย่างไว้ด้วยกัน ก็ไม่สามารถทำได้ ดังนั้นเราจะใช้โครงสร้างข้อมูลชนิดหนึ่งเข้ามาช่วย ก็คือ struct

การเขียนstruct เป็นการเขียนโครงสร้างที่เราสร้างขึ้นเองในโปรแกรม โดยที่เราสามารถกำหนดได้ว่าจะให้ประกอบไปด้วยอะไรบ้าง ก่อนอื่นเราก็ต้องเริ่มออกแบบก่อนว่าจะให้โครงสร้างเราประกอบไปด้วยอะไรบ้าง ซึ่งมีขั้นตอนดังนี้

1. ออกแบบโครงสร้าง
2. เขียนโค้ดเพื่อให้เกิดโครงสร้าง
3. วิธีการสร้างวัตถุออกมาจากโครงสร้างที่กำหนดไว้

เช่น จะทำโครงสร้างของกล่องสี่เหลี่ยม

1. ออกแบบโครงสร้าง(struct pattern) เป็นคำนิยามที่เราให้ไว้กับโครงสร้างของเรา

ก็กำหนดว่า กล่องสี่เหลี่ยมเป็นโครงสร้างที่ ประกอบไปด้วย ความกว้าง ความยาว และ ความสูง

2. เขียนโค้ดเพื่อให้เกิด โครงสร้าง(struct code) เป็นโค้ดตามข้อกำหนดของภาษาซีพลัสพลัส

* เปรียบเทียบได้กับโรงงานสร้างวัตถุ มีแต่โรงงานแต่ยังไม่มีกล่อง

ซึ่งโครงสร้างในการเขียน struct มีดังนี้

```
struct ชื่อ
{
ตัวแปรที่เป็นองค์ประกอบ
};
```

ดังนั้นถ้าเขียนโครงสร้างของกล่องก็จะได้

```
struct  box
{
double width;
double length;
double height;
};
```

3. วิธีการสร้างวัตถุออกมาจากโครงสร้างที่กำหนดไว้ (struct example) เป็นวิธีการเขียนโค้ดเพื่อสร้างวัตถุจาก struct ที่สร้างไว้ตามข้อกำหนดของภาษาซีพลัสพลัส

การประกาศตัวแปรที่เป็น struct เหมือนกับการประกาศตัวแปรปกติ เพียงแต่ชนิดข้อมูลคือ ชื่อ struct

เช่น box b;

การเข้าถึงข้อมูลที่อยู่ในตัวแปร struct สามารถทำได้โดยใช้ . แล้วตามด้วยชื่อตัวแปรข้างใน

เช่น

b.width หรือ b.length หรือ b.height

เพราะฉะนั้นถ้าเราจะกำหนดค่าต่างๆให้กับกล่อง ที่ชื่อ b

ก็ทำได้โดย

b.width = 2.0;

b.legth = 3.0 ;

b.height = 2.0;

สรุปทั้ง 3 ขั้นตอนได้

Struct pattern	box เป็นโครงสร้างที่ประกอบไปด้วย - width เป็นข้อมูลเลขทศนิยม - length เป็นข้อมูลเลขทศนิยม - height เป็นข้อมูลเลขทศนิยม

Struct code	struct box
	{
	double width;
	double length;
	double height;
	};

Struct example	box b;
	b.width= 2.0;
	b.length = 3.0;
	b.height = 2.0 ;

ต่อมาเราจะทำการเขียนโปรแกรมที่ต้องนำเอาวัตถุเข้ามาใช้

เมื่อเราทำการสร้าง struct เสร็จแล้วตามขั้นตอน 3 ขั้นตอนด้านบน

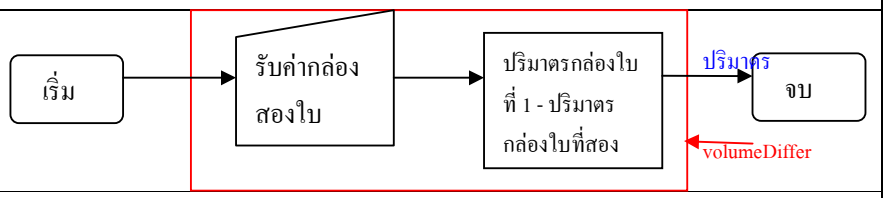
(เราสร้างโรงงานสร้างกล่องไว้แล้ว ดังนั้นต่อไปเราจะทำอะไรที่เกี่ยวกับกล่องก็ใช้โรงงานนี้)

เช่น โปรแกรมคำนวณหาความต่างของปริมาตรกล่องสองใบ

เมื่อเราจะเขียนโปรแกรม เราก็ต้องทำการวิเคราะห์โจทย์ด้วย 8 ขั้นตอนที่เราเรียนไว้แล้วตอนต้น

Purpose	คำนวณหาความต่างของปริมาตรกล่องสองใบ
Input	กล่องใบที่ 1 , กล่องใบที่ 2
Output	ความต่างปริมาตร
Contract	volumeDiffer : box,box->double

Example	box b1,b2;
	b1.width= 4.0;
	b1.length = 5.0;
	b1.height = 2.0 ;
	b2.width= 2.0;
	b2.length = 3.0;
	b2.height = 2.0 ;
	volumeDiffer (b1,b2) = 273.0

Flowchart	

Function code	double volumeDiffer (box box1,box box2)
	{
	return (box1.width*box1.length*box1.height)- (box2.width*box2.length*box2.height);
	}
	*การหาปริมาตรเกิดขึ้นสองครั้ง อาจเขียนเป็นฟังก์ชันไว้แยกก็ได้

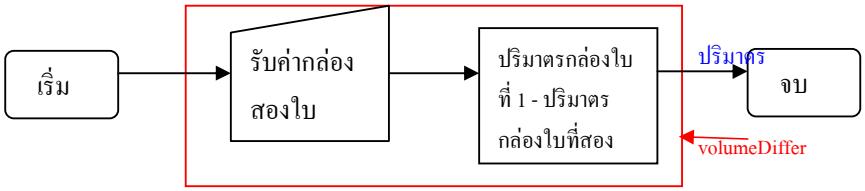
Main function code	int main()
	{
	box b1,b2;
	cout<<" type properties of box 1 width ,length , height ";
	cin>>b1.width;
	cin>>b1.length;
	cin>>b1.height ;
	cout<<" type properties of box 2 width ,length , height ";
	cin>>b2.width; ค่าของ input จะกลายเป็นค่า box1 หรือ box2 ในฟังก์ชันนั้นขึ้นอยู่กับ
	cin>>b2.length ; ลำดับการเรียง input อย่างนี้ box1 ในฟังก์ชัน ก็จะเท่ากับ b1 และ
	cin>>b2.height ; box2 ในฟังก์ชัน ก็จะเท่ากับ b2
	cout<<" box 1 have volume more than box 2 = "<<volumeDiffer(b1,b2)<<endl;
	return 0;
	}

สรุปโค้ดทั้งหมดในโปรแกรม

```
/*
Struct pattern
box เป็นโครงสร้างที่ประกอบไปด้วย
        - width เป็นข้อมูลเลขทศนิยม
        - length เป็นข้อมูลเลขทศนิยม
        - height เป็นข้อมูลเลขทศนิยม
*/
// Struct code
struct box
{
    double width;
    double length;
    double height;
    };
/*
Struct example
box b;
b.width= 2.0;
b.length = 3.0;
b.height = 2.0 ;

    Purpose คำนวณหาความต่างของปริมาตรกล่องสองใบ
    Input กล่องใบที่ 1 , กล่องใบที่ 2
    Output ความต่างปริมาตร
    Contract volumeDiffer : box,box->double
    Example
    box b1,b2;
    b1.width= 4.0;
    b1.length = 5.0;
    b1.height = 2.0 ;
    b2.width= 2.0;
    b2.length = 3.0;
    b2.height = 2.0 ;
    volumeDiffer (b1,b2) = 273.0
```

Flowchart

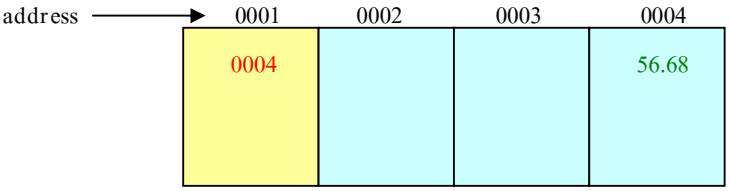



```
*/
// Function code
double volumeDiffer (box box1,box box2 )
{
return  (box1.width*box1.length*box1.height)- (box2.width*box2.length*box2.height);
}
// Main function code
void main()
{
box b1,b2;
cout<<" type properties of box 1  width ,length , height ";
cin>>b1.width;
cin>>b1.length;
cin>>b1.height  ;
cout<<" type properties of box 2  width ,length , height ";
cin>>b2.width;
cin>>b2.length ;
cin>>b2.height  ;
cout<<" box 1 have volume more than box 2  = "<<volumeDiffer(b1,b2)<<endl;
}
```

บทที่ 11

โครงสร้างข้อมูลแบบ Pointer

โครงสร้างข้อมูลแบบพอยเตอร์ (pointer) มีลักษณะคือตัวของพอยเตอร์เองไม่ได้ทำหน้าที่เก็บค่าข้อมูลใดๆไว้ แต่ว่าจะทำการเก็บค่าตำแหน่งของ memory ของตัวแปรข้อมูลตัวใดตัวหนึ่งไว้ ดังนั้นไม่ว่า ณ ตำแหน่ง memory ตำแหน่งนั้นจะมีค่าข้อมูลใดอยู่ ตัวพอยเตอร์ก็จะเห็นค่าข้อมูลนั้นๆด้วย อธิบายได้ดังรูป



ถ้าสมมติให้กล่องสี่เหลี่ยมอันใหญ่นี้คือหน่วยความจำ (memory) ดังนั้น memory ก็จะแบ่งเนื้อที่ออกเป็นส่วนๆในการเก็บข้อมูล สมมติให้เก็บข้อมูล 4 ส่วน ดังนั้นก็จะมีตำแหน่งของแต่ละส่วน ให้เป็นตำแหน่ง address คือ 0001 ,0002,0003 และ 0004 ดังนั้นถ้าให้ที่ตำแหน่ง 0001 คือข้อมูลที่มีโครงสร้างเป็นพอยเตอร์แล้ว (บริเวณสีเหลือง) และถูกกำหนดให้ข้อมูลนี้ชี้ค่าไปยัง address ตำแหน่ง 0004 ดังนั้น ข้อมูลพอยเตอร์ก็จะเก็บค่าข้อมูลคือ address ของmemory ที่มันอ้างถึง ซึ่งก็คือ 0004 (ตัวหนังสือสีแดง) ซึ่ง ณ ตำแหน่ง 0004 เองมีการเก็บค่าข้อมูลเป็นชนิดเลขทศนิยม มีค่าคือ 56.68 ถ้าต้องการแสดงค่า 56.68 ก็สามารถที่จะแสดงได้สองวิธีคือ

1. โดยการอ้างถึงค่าใน address ตำแหน่ง 0004 โดยตรง หรือ
2. โดยการอ้างผ่านตัวแปร พอยเตอร์ ใน address ตำแหน่ง 0001

และไม่ว่าที่ตำแหน่ง 0004 จะเปลี่ยนข้อมูลเป็นค่าใดก็ตาม ตัวแปรพอยเตอร์ก็จะเห็นข้อมูลนั้นเสมอ เนื่องจากมันอ้างด้วยค่า address ของ memory ไม่ได้อ้างตรงๆไปที่ตัวข้อมูลนั้นๆ (เช่น เปลี่ยนจาก 56.68 เป็น 12.01 ที่ตำแหน่ง 0001 ก็จะเห็นเป็นเลข 12.01 เช่นกัน)

11.1 การสร้างตัวแปรพอยเตอร์

การประกาศตัวแปรทั่วๆไปแล้วมีวิธีการคือ

ชนิดข้อมูล ชื่อตัวแปร ; เช่น int x ;

สำหรับโครงสร้างข้อมูลแบบพอยเตอร์ก็เช่นกัน ยังคงใช้วิธีเดียวกัน แต่ต่างกันตรงที่ จะต้องมีเครื่องหมาย * ด้านหน้าของชื่อตัวแปร ดังนี้

ชนิดข้อมูล * ชื่อตัวแปร ; เช่น int *x ;

ซึ่งหมายความว่า ตัวแปร ชื่อ x มีโครงสร้างเป็นพอยเตอร์ มีชนิดข้อมูลเป็น integer ดังนั้น ตัวแปร พอยเตอร์ x จะสามารถอ้างถึง memory ตำแหน่งที่มีชนิดข้อมูลเป็น integer ได้เหมือนกันเท่านั้น

หมายเหตุ ตัวแปรพอยเตอร์มีลักษณะพิเศษคือ จะสามารถอ้างอิงข้อมูลได้เฉพาะตัวแปรที่มีชนิดข้อมูลเดียวกันเท่านั้น

ตัวอย่างเช่น

```
int x ;
char y;
```

ถ้าต้องการสร้างตัวแปรพอยเตอร์เพื่ออ้างข้อมูลใน x และ y ก็จำเป็นจะต้องสร้างสองตัวให้มีชนิดเป็น integer และ character ตามลำดับ ก็คือ

```
int *pt_x;
char *pt_y;
```

11.2 การกำหนดให้ตัวแปรพอยเตอร์อ้างถึงข้อมูลอื่นๆ

มี 2 วิธีดังนี้

1. กำหนดตั้งแต่เริ่มประกาศตัวแปรพอยเตอร์ โดยการใช้เครื่องหมายวงเล็บ () และด้านในก็ใส่ชื่อ address ของตัวแปรที่ต้องการให้ชี้ไป ซึ่ง address ของตัวแปรใดๆ สามารถเข้าถึงได้โดยใช้เครื่องหมาย & นำหน้าชื่อตัวแปรนั้นๆ

   เช่น int x = 10 , *pt_x(&x);

2. กำหนดหลังจากประกาศตัวแปร ทำได้เหมือนกับการกำหนดค่าให้กับตัวแปรปกติ คือ ใช้เครื่องหมายเท่ากับ = แล้วตามด้วย address ของตัวแปรที่ต้องการให้ชี้ไป

   เช่น int x = 10, *pt_x;

   pt_x = &x;

ดังนั้นสรุปได้ว่า ตัวแปร x และ pt_x มีคุณสมบัติ ดังนี้

Address ใน memory	ชื่อตัวแปร	ค่าข้อมูลที่เก็บไว้
0001	x	10
0002	pt_x	0001

11.3 การอ้างถึงข้อมูลกับตัวแปรพอยเตอร์

การอ้างถึงข้อมูลที่ตัวแปรพอยเตอร์ชี้ สามารถทำได้โดยใช้เครื่องหมาย * นำหน้าชื่อตัวแปรพอยเตอร์ เช่น

```
int x = 10, *pt_x;
pt_x = &x;
```

ถ้าต้องการแสดงค่าว่าตัวแปรพอยเตอร์มองเห็นข้อมูลอะไรอยู่ จะได้ว่า

```
cout<< *pt_x<<endl;   //ได้คำตอบคือ 10
```

ถ้าต้องการแสดงว่าตัวแปรพอยเตอร์เก็บข้อมูลอะไรอยู่ จะได้ว่า

```
cout<< *pt_x<<endl;   //ได้คำตอบคือ 0001
```

ถ้าต้องการดู address ของ memory ของตัวแปรใดๆ จะได้ว่า

```
cout<<&x<<endl;      //ได้คำตอบคือ 0001
cout<<&pt_x<<endl;  //ได้คำตอบคือ 0002
```

สมมติว่าเปลี่ยนค่าตัวแปร x เป็น 15

```
x = 15;
```

และถ้า

```
cout<< *pt_x<<endl;   //ได้คำตอบคือ 15
cout<< *pt_x<<endl;   //ได้คำตอบคือ 0001
cout<<&pt_x<<endl;   //ได้คำตอบคือ 0002
cout<<&x<<endl;       //ได้คำตอบคือ 0001
```

บทที่ 12

โครงสร้างข้อมูลแบบ linked-list

โครงสร้างข้อมูลแบบลิงก์ลิส (linked-list) คือโครงสร้างข้อมูลที่มีการเชื่อมต่อกับตัวแปรที่มีโครงสร้างข้อมูลชนิดเดียวกันได้อย่างไม่สิ้นสุด

ลิงก์ลิสมีสองแบบ คือ ลิงก์ลิสแบบชี้ไปทางเดียว (singly linked-list) กับ ชี้ไปสองทาง (doubly linked-list)

ในที่นี้เราจะพูดถึง ลิงก์ลิสแบบชี้ไปทางเดียวก่อน

ลิงก์ลิส คือโครงสร้างข้อมูลชนิดหนึ่ง ซึ่งอาจจะเป็น **ลิสว่างเปล่า** หรือ

เป็น **โครงสร้างข้อมูลที่ประกอบไปด้วย ข้อมูลสองส่วน ก็คือ**

- **ส่วนเก็บข้อมูล**
- **ส่วนที่ทำหน้าที่ชี้ไปยังลิงก์ลิสตัวต่อไป**

วิธีการสร้างโครงสร้างข้อมูลแบบลิงก์ลิส สามารถทำได้โดย การประกาศโครงสร้างขึ้นมาตามรูปแบบของ struct ที่เรียนไปแล้วในบทที่ 10 ดังนี้

1. ออกแบบโครงสร้าง(struct pattern) เป็นคำนิยามที่เราให้ไว้กับโครงสร้างของเรา

ก็กำหนดว่า linkedList เป็นโครงสร้างที่ ประกอบไปด้วย

ส่วนเก็บข้อมูล

ส่วนที่ทำหน้าที่ชี้ไปยังลิงก์ลิสตัวต่อไป

2. เขียนโค้ดเพื่อให้เกิดโครงสร้าง(struct code) เป็นโค้ดตามข้อกำหนดของภาษาซีพลัสพลัส

ซึ่งโครงสร้างในการเขียน struct มีดังนี้

```
struct ชื่อ
{
ตัวแปรที่เป็นองค์ประกอบ
};
```

3. วิธีการสร้างวัตถุออกมาจากโครงสร้างที่กำหนดไว้ (struct example) เป็นวิธีการเขียนโค้ดเพื่อสร้างวัตถุจาก struct ที่สร้างไว้ตามข้อกำหนดของภาษาซีพลัสพลัส

การประกาศตัวแปรที่เป็น linkedListเหมือนกับการประกาศตัวแปร struct โดยชนิดข้อมูลคือ ชื่อstruct แต่จะต้องมีเครื่องหมาย * (คือพอยเตอร์นั่นเอง) หน้าชื่อตัวแปรที่เป็นโครงสร้าง linkedList

ต้องมีการใช้ keyword คือ new เพื่อเป็นการกำหนดค่าเริ่มต้นให้กับโครงสร้าง ดังนี้

ชื่อโครงสร้าง * ชื่อตัวแปร = new ชื่อโครงสร้าง (แบบนี้เป็นการประกาศตัวแปร พร้อมกับทำการ new ด้วย)

หรือจะประกาศตัวแปรก่อน แล้วค่อยทำ

ชื่อตัวแปร = new ชื่อโครงสร้าง

การเข้าถึงข้อมูลที่อยู่ในตัวแปร struct สามารถทำได้โดยใช้ -> แล้วตามด้วยชื่อตัวแปรข้างใน

ตัวอย่างเช่น ทำการสร้างลิงก์ลิสที่เก็บข้อมูลเพียงอย่างเดียว ก็คือ ตัวเลขจำนวนเต็ม 1 จำนวน ดังนั้นจะได้

ขั้นตอน 1 ออกแบบโครงสร้าง(struct pattern) ได้ทำไปแล้วด้านบน

ขั้นตอน 2 เขียนโค้ดเพื่อให้เกิดโครงสร้าง(struct code) ได้คือ

```
struct linkedList
{
int number;
struct linkedList *next;  /* ส่วนที่ชี้ไปยังข้อมูลลิงก์ลิสตัวต่อไป มีโครงสร้างแบบพอยเตอร์ด้วย ดังนั้นจึงต้อง มี * นำหน้าชื่อ และตรงชนิดข้อมูลของตัวแปร สังเกตว่า ไม่ใช้คำว่า linkedList เฉยๆเนื่องจากว่า มันซ้อนอยู่ใน struct จึงต้องอ้างชนิดข้อมูลเต็มๆคือ struct linkedList*/
};
```

ขั้นตอน 3 การสร้างวัตถุออกมาจากโครงสร้างที่กำหนดไว้ (struct example)

เมื่อมีโครงสร้างของลิงก์ลิสแล้ว เราก็สามารถกระทำการสร้างตัวแปรที่มีโครงสร้างแบบนี้ได้ ซึ่งเหมือนกับการประกาศตัวแปรปกติ

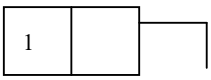
สมมติให้ตัวแปร ลิงก์ลิสนี้เก็บตัวเลข คือ เลข 1

จะได้

```
linkedList  *first; // ประกาศตัวแปรตามปกติก่อน
first = new linkedList; // การสร้างตัวแปรที่เป็น struct pointer ต้องมีการกำหนดค่าเริ่มต้นให้ตัวแปรก่อน จึงจะนำไปใช้ได้ โดยการใช้คีย์เวิร์ด new
first->number = 1; // การเข้าถึงองค์ประกอบที่อยู่ใน struct pointer ใช้เครื่องหมาย ->
first->next= NULL; // คีย์เวิร์ด NULL หมายถึงว่างเปล่า ใช้กับการกำหนดตัวแปรที่เป็นวัตถุให้มีค่าว่าง หมายถึง first ไม่มีข้อมูลตัวถัดไปอีก
```

เพื่อความเข้าใจ ถ้าสมมติให้รูปร่างลิงก์ลิสเป็นดังนี้

จะได้



สร้างลิงก์ลิงก์ลิส ที่มีจำนวนตัวเลข 1 ต่อด้วย 2

มีสองวิธี

1. ไม่ประกาศชื่อตัวแปรใหม่

จะได้

```
linkedList *first,*second;
first= new linkedList;
second = new linkedList;
first->number = 1 ;
first->next=new linkedList ; //กำหนดให้ตัวแปร first ชี้ข้อมูลถัดไปคือตัวแปรใหม่
```

first->next->number = 2; // ตรงตำแหน่ง first->next เป็นการชี้ตัวลิงก์ลิสตัวใหม่แล้ว ถ้าจะกำหนดค่าตัวเลขให้กับลิสใหม่ก็ต้องอ้าง first->next ก่อน ถึงจะอ้าง ->number ต่อ (คล้ายๆกับ first->number แต่ตรงตำแหน่ง first ถูกขยายเป็น first->next)

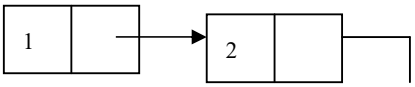
first->next->next = NULL;

2. ประกาศชื่อตัวแปรสำหรับทุกลิงก์ลิสใหม่

จะได้

```
linkedList *first,*second;
first= new linkedList;
second = new linkedList;
first->number = 1 ;
first->next=second; //กำหนดให้ตัวแปร first ชี้ข้อมูลถัดไปคือตัวแปร second
second->number=2;
second->next=NULL;
```

จะได้ดังรูป



หมายเหตุ ตัวพอยเตอร์ที่ทำหน้าที่ชี้ลิงก์ลิสตัวถัดไป จะมองเห็นลิงก์ลิสทั้งหมด ไม่ใช้เห็นเพียงส่วนข้อมูลส่วนแรกหรือส่วนหลังเพียงอย่างเดียว เพราะฉะนั้นรูปลูกศรจากรูปสมมติ จึงชี้อยู่เพียงด้านหน้าของลิงก์ลิส

สรุปทั้งสามขั้นตอนได้

Struct pattern	linkedList เป็นโครงสร้างที่ประกอบไปด้วย - number เป็นข้อมูลเลขจำนวนเต็ม - next เป็นตัวแปรโครงสร้างพอยเตอร์ที่ชี้ลิงก์ลิสตัวถัดไป

Struct code	struct linkedList
	{
	int number;
	struct linkedList *next;
	};

Struct example	linkedList *first;
	first = new linkedList;
	first->number = 1;
	first->next = NULL;

บทที่ 13

การเขียนโปรแกรม โครงสร้างข้อมูลแบบ singly linked-list

ก่อนอื่นก็ทำการสร้างโครงสร้างลิงก์ลิสก่อน

1. ออกแบบโครงสร้าง(struct pattern) เป็นคำนิยามที่เราให้ไว้กับโครงสร้างของเรา

ก็กำหนดว่า linkedList เป็นโครงสร้างที่ ประกอบไปด้วย

ส่วนเก็บข้อมูล คือชนิด integer

ส่วนที่ทำหน้าที่ชี้ไปยังลิงก์ลิสตัวต่อไป คือ ชนิด linkedList

2. เขียนโค้ดเพื่อให้เกิดโครงสร้าง(struct code)

```
struct linkedList
{
int number;
struct linkedList *next;
};
```

3. วิธีการสร้างวัตถุออกมาจากโครงสร้างที่กำหนดไว้ (struct example)

```
linkedList  *first; // ประกาศตัวแปรตามปกติก่อน
first = new linkedList; // การสร้างตัวแปรที่เป็น struct pointer ต้องมีการกำหนดค่าเริ่มต้นให้ตัวแปรก่อน จึงจะนำไปใช้ได้ โดยการใช้คีย์เวิร์ด new
first->number = 1; // การเข้าถึงองค์ประกอบที่อยู่ใน struct pointer ใช้เครื่องหมาย ->
first->next= NULL; // คีย์เวิร์ด NULL หมายถึงว่างเปล่า ใช้กับการกำหนดตัวแปรที่เป็นวัตถุให้มีค่าว่าง
```

สรุปทั้ง 3 ขั้นตอนได้

Struct pattern	linkedList เป็นโครงสร้างที่ประกอบไปด้วย - number เป็นข้อมูลเลขจำนวนเต็ม - next เป็นตัวแปร โครงสร้างพอยเตอร์ที่ชี้ลิงก์ลิสตัวถัดไป

Struct code	struct linkedList
	{
	int number;
	struct linkedList *next;
	};

Struct example	linkedList *first;
	first = new linkedList;
	first->number = 1;
	first->next = NULL;

ต่อมาเราจะทำการเขียนโปรแกรมที่ต้องนำเอาวัตถุเข้ามาใช้

เมื่อเราทำการสร้าง struct เสร็จแล้วตามขั้นตอน 3 ขั้นตอนด้านบน

(เราสร้างโรงงานสร้างลิงก์ลิสสำหรับเลขจำนวนเต็ม 1 จำนวนไว้แล้ว ดังนั้นต่อไปเราจะทำอะไรที่เกี่ยวกับลิงก์ลิสที่เก็บเลขจำนวนเต็มก็ใช้โรงงานนี้)

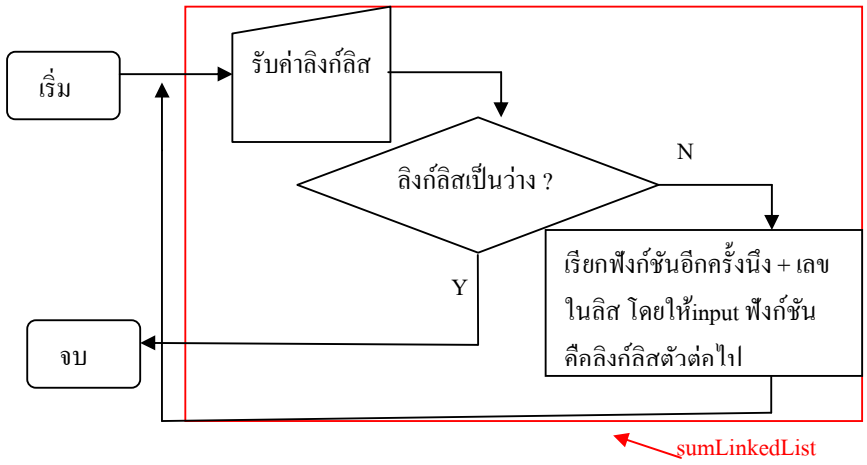
เมื่อเราจะเขียนโปรแกรม เราก็ต้องทำการวิเคราะห์โจทย์ด้วย 8 ขั้นตอนที่เราเรียนไว้แล้วตอนต้น

โจทย์คือ ให้เขียนโปรแกรมคำนวณหาผลรวมของเลขจำนวนเต็มทั้งหมดในลิงก์ลิส 1 ลิส

วิธีที่ 1 เขียนแบบ recursive function จะได้

Purpose	คำนวณหาผลรวมของเลขจำนวนเต็มทั้งหมดในลิงก์ลิส 1 ลิส
Input	ลิงก์ลิส 1 ลิงก์ลิส
Output	ผลรวมของเลขจำนวนเต็มทั้งหมดในลิงก์ลิส
Contract	sumLinkedList : linkedList->int

Example	linkedList *first,*second;
	first = new linkedList;
	second = new linkedList;
	first->number = 1;
	first->next =second;
	second->number = 2;
	second->next = NULL;
	sumLinkedList (first) = 3

Flowchart	

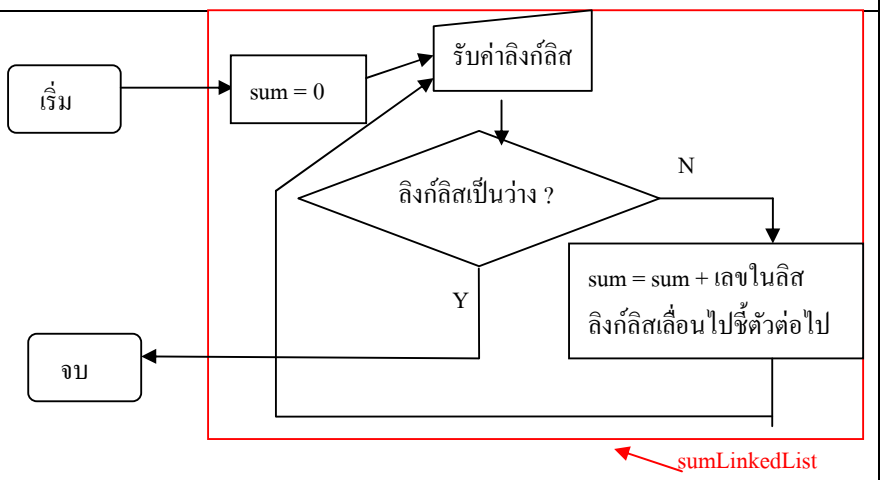
Function code	int sumLinkedList (linkedList lList)
	{
	if (lList!=NULL) // ตรวจสอบว่าวัถตุเป็นว่างหรือไม่
	{
	return 0;
	}else
	{
	return lList->number + sumLinkedList(lList->next); // ให้ input คือ ลิสตัวถัดไป
	}
	}

Main function code	int main()
	{
	linkedList *first,*second;
	first = new linkedList;
	second = new linkedList;
	first->number = 1;
	first->next =second;
	second->number = 2;
	second->next = NULL;
	cout<<sumLinkedList (first) <<endl;
	return 0;
	}
	จะเห็นว่าตัวแปรที่ใส่เป็น input ให้กับฟังก์ชัน sumLinkedList คือ first เนื่องจากว่า first เป็นชื่อตัวแปรลิงก์ลิสลำดับแรกสุด จะทำให้ได้ input ตั้งแต่เลข 1

วิธีที่ 2 เขียนแบบ ใช้ loop programmingจะได้

Purpose	คำนวณหาผลรวมของเลขจำนวนเต็มทั้งหมดในลิงก์ลิส 1 ลิส
Input	ลิงก์ลิส 1 ลิงก์ลิส
Output	ผลรวมของเลขจำนวนเต็มทั้งหมดในลิงก์ลิส
Contract	sumLinkedList : linkedList->int

Example	linkedList *first,*second;
	first = new linkedList;
	second = new linkedList;
	first->number = 1;
	first->next =second;
	second->number = 2;
	second->next = NULL;
	sumLinkedList (first) = 3

Flowchart	

Function code	int sumLinkedList (linkedList lList)
	{
	int sum= 0;
	while(lList!=NULL) // ตรวจสอบว่าวัถตุเป็นว่างหรือไม่
	{
	sum=sum+lList->number; // บวกค่า sum กับค่าตัวเลขในลิงก์ลิส
	lList=lList->next; // เลื่อนพอยเตอร์ไปยังลิงก์ลิสถัดไป
	}
	return sum;
	}

Main function code	int main()
	{
	linkedList *first,*second;
	first = new linkedList;
	second = new linkedList;
	first->number = 1;
	first->next =second;
	second->number = 2;
	second->next = NULL;
	cout<<sumLinkedList (first) <<endl;
	return 0;
	}

จะเห็นว่าตัวแปรที่ใส่เป็น input ให้กับฟังก์ชัน sumLinkedList คือ first เนื่องจากว่า first เป็นชื่อตัวแปรลิงก์ลิสลำดับแรกสุด จะทำให้ได้ input ตั้งแต่เลข 1

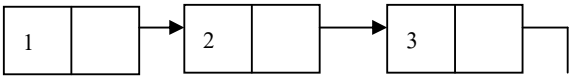
การแทรกลิงก์ลิส

การแทรกลิงก์ลิสที่เป็นแบบไปทางเดียว มีหลักการง่ายๆ คือ

> next ของลิงก์ลิสใหม่ จะเท่ากับ next ของลิงก์ลิสเก่า
> next ของลิงก์ลิสเก่า จะเท่ากับ ลิงก์ลิสใหม่

ความหมายสามารถอธิบายได้ดังรูป

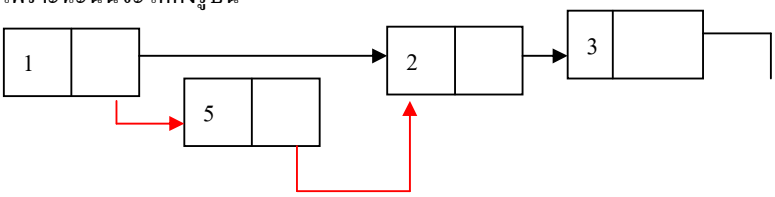
สมมติให้ลิงก์ลิสเก่าเป็นดังรูปนี้



และลิงก์ลิสใหม่ เป็นดังนี้

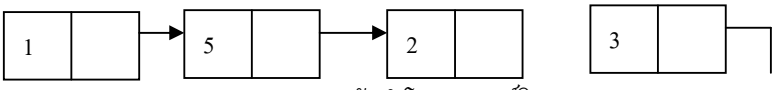
เราจะนำลิงก์ลิสใหม่ไปแทรกระหว่างเลข 1 กับ 2

เพราะฉะนั้นจะได้ดังรูปนี้



หมายเหตุเส้นสีแดงคือเส้นใหม่ตามหลักการด้านบน

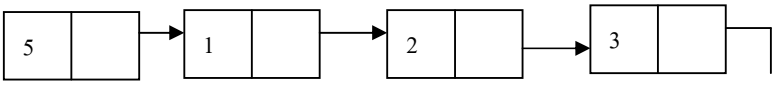
ก็จะได้เป็นดังนี้



แต่จะมีข้อยกเว้นอยู่ 1 กรณี คือ กรณีที่ลิงก์ลิสใหม่ถูกแทรกให้อยู่ด้านหน้าสุด ก็จะได้หลักการดังนี้

next ของลิงก์ลิสใหม่ จะเท่ากับ ลิงก์ลิสเก่า

เพราะะนั้นจะได้เป็นดังนี้



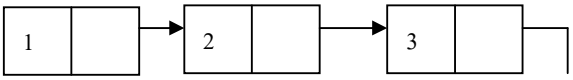
การลบลิงก์ลิส

การลบลิงก์ลิสที่เป็นแบบไปทางเดียว มีหลักการง่ายๆ คือ

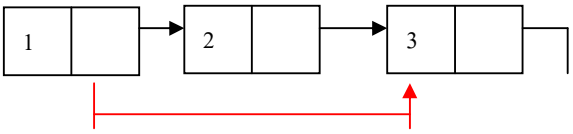
> next ของลิงก์ลิสตัวก่อนหน้า จะเท่ากับ next ของลิงก์ลิสตัวที่ถูกลบ

ความหมายสามารถอธิบายได้ดังรูป

สมมติให้ลิงก์ลิสเก่าเป็นดังรูปนี้

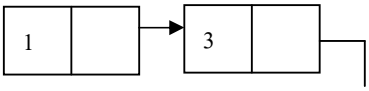


แล้วทำการลบตำแหน่งที่ 2 ก็จะได้เป็นดังนี้



หมายเหตุเส้นสีแดงคือเส้นใหม่ตามหลักการด้านบน

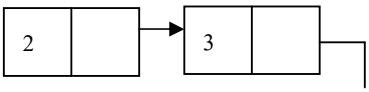
ก็จะได้เป็นดังนี้



แต่จะมีข้อยกเว้นอยู่ 1 กรณี คือ กรณีที่ลบลิงก์ลิสที่อยู่ด้านหน้าสุด ก็จะได้หลักการดังนี้

> ลิงก์ลิสตัวแรก จะเท่ากับ next ของลิงก์ลิสตัวแรก

เพราะะนั้นจะได้เป็นดังนี้



บทที่ 14
โครงสร้างข้อมูลแบบ linked-list แบบชี้ไปสองทาง

ลิงก์ลิสมีสองแบบ คือ ลิงก์ลิสแบบชี้ไปทางเดียว (singly linked-list) กับ ชี้ไปสองทาง (doubly linked-list)
สำหรับโครงสร้างลิงก์ลิสแบบชี้ไปสองทางมีดังนี้
ลิงก์ลิส คือโครงสร้างข้อมูลชนิดหนึ่ง ซึ่งอาจจะเป็น **ลิสว่างเปล่า** หรือ
เป็น **โครงสร้างข้อมูลที่ประกอบไปด้วย ข้อมูลสองส่วน ก็คือ**

- **ส่วนเก็บข้อมูล**
- **ส่วนที่ทำหน้าที่ชี้ไปยังลิงก์ลิสตัวก่อนหน้า**
- **ส่วนที่ทำหน้าที่ชี้ไปยังลิงก์ลิสตัวต่อไป**

วิธีการสร้างโครงสร้างข้อมูลแบบลิงก์ลิส สามารถทำได้โดย การประกาศโครงสร้างขึ้นมาตามรูปแบบของ struct ที่เรียนไปแล้วในบทที่ 10 ดังนี้

1. ออกแบบโครงสร้าง(struct pattern) เป็นคำนิยามที่เราให้ไว้กับโครงสร้างของเรา

ก็กำหนดว่า linkedList เป็นโครงสร้างที่ ประกอบไปด้วย
ส่วนเก็บข้อมูล
ส่วนที่ทำหน้าที่ชี้ไปยังลิงก์ลิสตัวก่อนหน้า
ส่วนที่ทำหน้าที่ชี้ไปยังลิงก์ลิสตัวต่อไป

2. เขียนโค้ดเพื่อให้เกิดโครงสร้าง(struct code) เป็นโค้ดตามข้อกำหนดของภาษาซีพลัสพลัส

ซึ่งโครงสร้างในการเขียน struct มีดังนี้
struct ชื่อ
{
ตัวแปรที่เป็นองค์ประกอบ
};

4. วิธีการสร้างวัตถุออกมาจากโครงสร้างที่กำหนดไว้ (struct example) เป็นวิธีการเขียนโค้ดเพื่อสร้างวัตถุจาก struct ที่สร้างไว้ตามข้อกำหนดของภาษาซีพลัสพลัส

การประกาศตัวแปรที่เป็น linkedListเหมือนกับการประกาศตัวแปร struct โดยชนิดข้อมูลคือ ชื่อstruct แต่จะต้องมีเครื่องหมาย * (คือพอยเตอร์นั่นเอง) หน้าชื่อตัวแปรที่เป็นโครงสร้าง linkedList
ต้องมีการใช้ keyword คือ new เพื่อเป็นการกำหนดค่าเริ่มต้นให้กับโครงสร้าง ดังนี้
ชื่อโครงสร้าง * ชื่อตัวแปร = new ชื่อโครงสร้าง (แบบนี้เป็นการประกาศตัวแปร พร้อมกับทำการ new ด้วย)
หรือจะประกาศตัวแปรก่อน แล้วค่อยทำ
ชื่อตัวแปร = new ชื่อโครงสร้าง
การเข้าถึงข้อมูลที่อยู่ในตัวแปร struct สามารถทำได้โดยใช้ -> แล้วตามด้วยชื่อตัวแปรข้างใน

ตัวอย่างเช่น ทำการสร้างลิงก์ลิสที่เก็บข้อมูลเพียงอย่างเดียว ก็คือ ตัวเลขจำนวนเต็ม 1 จำนวน ดังนั้นจะได้

ขั้นตอน 1 ออกแบบโครงสร้าง(struct pattern) ได้ทำไปแล้วด้านบน

ขั้นตอน 2 เขียนโค้ดเพื่อให้เกิดโครงสร้าง(struct code) ได้คือ

```
struct linkedList
{
int number;
struct linkedList *prev;  /* ส่วนที่ชี้ไปยังข้อมูลลิงก์ลิสตัวก่อนหน้า
struct linkedList *next;  /* ส่วนที่ชี้ไปยังข้อมูลลิงก์ลิสตัวต่อไป
};
```

ขั้นตอน 3 การสร้างวัตถุออกมาจากโครงสร้างที่กำหนดไว้ (struct example)

เมื่อมีโครงสร้างของลิงก์ลิสแล้ว เราก็สามารถกระทำการสร้างตัวแปรที่มีโครงสร้างแบบนี้ได้ ซึ่งเหมือนกับการประกาศตัวแปรปกติ

สมมติให้ตัวแปร ลิงก์ลิสนี้เก็บตัวเลข คือ เลข 1

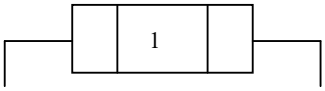
จะได้

```
linkedList  *first; // ประกาศตัวแปรตามปกติก่อน
first = new linkedList; // การสร้างตัวแปรที่เป็น struct pointer ต้องมีการกำหนดค่าเริ่มต้นให้ตัวแปรก่อน จึงจะนำไปใช้ได้ โดยการใช้คีย์เวิร์ด new
first->number = 1; // การเข้าถึงองค์ประกอบที่อยู่ใน struct pointer ใช้เครื่องหมาย ->
first->prev= NULL; // คีย์เวิร์ด NULL หมายถึงว่างเปล่า ใช้กับการกำหนดตัวแปรที่เป็นวัตถุให้มีค่าว่าง หมายถึง first ไม่มีข้อมูลตัวก่อนหน้า
first->next= NULL; // คีย์เวิร์ด NULL หมายถึงว่างเปล่า ใช้กับการกำหนดตัวแปรที่เป็นวัตถุให้มีค่าว่าง หมายถึง first ไม่มีข้อมูลตัวถัดไปอีก
```

เพื่อความเข้าใจ ถ้าสมมติให้รูปร่างลิงก์ลิสเป็นดังนี้

ส่วนเก็บพอยเตอร์ชี้ตัวก่อนหน้า	ส่วนเก็บตัวเลข	ส่วนเก็บพอยเตอร์ชี้ตัว

จะได้



สร้างลิงก์ลิงก์ลิส ที่มีจำนวนตัวเลข 1 ต่อด้วย 2

มีสองวิธี

1. ไม่ประกาศชื่อตัวแปรใหม่

จะได้

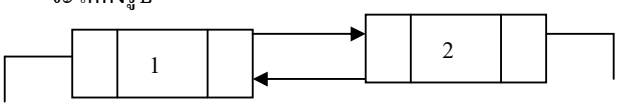
```
linkedList *first,*second;
first= new linkedList;
second = new linkedList;
first->number = 1 ;
first->prev=NULL ; //กำหนดให้ตัวแปร first ชี้ข้อมูลก่อนหน้าคือว่าง
first->next=new linkedList ; //กำหนดให้ตัวแปร first ชี้ข้อมูลถัดไปคือตัวแปรใหม่
first->next->number = 2;  // ตรงตำแหน่ง first->next เป็นการชี้ตัวลิงก์ลิสตัวใหม่แล้ว ถ้าจะกำหนดค่าตัวเลขให้กับลิสใหม่ก็ต้องอ้าง first->next ก่อน ถึงจะอ้าง ->number ต่อ (คล้ายๆกับ first->number แต่ตรงตำแหน่ง first ถูกขยายเป็น first->next )
first->next->prev = first;
first->next->next = NULL;
```

2. ประกาศชื่อตัวแปรสำหรับทุกลิงก์ลิสใหม่

จะได้

```
linkedList *first,*second;
first= new linkedList;
second = new linkedList;
first->number = 1 ;
first->prev=NULL; //กำหนดให้ตัวแปร first ชี้ข้อมูลก่อนหน้าคือว่าง
first->next=second; //กำหนดให้ตัวแปร first ชี้ข้อมูลถัดไปคือตัวแปร second
second->number=2;
second->prev=first;  //กำหนดให้ตัวแปร second ชี้ข้อมูลก่อนหน้าคือ first
second->next=NULL;
```

จะได้ดังรูป



หมายเหตุ ตัวพอยเตอร์ที่ทำหน้าที่ชี้ลิงก์ลิสตัวถัดไป จะมองเห็นลิงก์ลิสทั้งหมด ไม่ใช้เห็นเพียงส่วนข้อมูลส่วนแรกหรือส่วนหลังเพียงอย่างเดียว เพราะฉะนั้นรูปลูกศรจากรูปสมมติ จึงชี้อยู่เพียงด้านหน้าของลิงก์ลิส

สรุปทั้งสามขั้นตอนได้

Struct pattern	linkedList เป็นโครงสร้างที่ประกอบไปด้วย - number เป็นข้อมูลเลขจำนวนเต็ม - prev เป็นตัวแปรโครงสร้างพอยเตอร์ที่ชี้ลิงก์ลิสตัวก่อนหน้า - next เป็นตัวแปรโครงสร้างพอยเตอร์ที่ชี้ลิงก์ลิสตัวถัดไป

Struct code	struct linkedList
	{
	int number;
	struct linkedList *next , *prev ;
	};

Struct example	linkedList *first;
	first = new linkedList;
	first->number = 1;
	first->prev = NULL;
	first->next = NULL;

บทที่ 15

การเขียนโปรแกรมโครงสร้างข้อมูลแบบ doubly linked-list

ก่อนอื่นก็ทำการสร้างโครงสร้างลิงก์ลิสก่อน

1. ออกแบบโครงสร้าง(struct pattern) เป็นคำนิยามที่เราให้ไว้กับโครงสร้างของเรา

ก็กำหนดว่า linkedList เป็นโครงสร้างที่ ประกอบไปด้วย

ส่วนเก็บข้อมูล คือชนิด integer

ส่วนที่ทำหน้าที่ชี้ไปยังลิงก์ลิสตัวก่อนหน้า คือ ชนิด linkedList

ส่วนที่ทำหน้าที่ชี้ไปยังลิงก์ลิสตัวต่อไป คือ ชนิด linkedList

2. เขียนโค้ดเพื่อให้เกิดโครงสร้าง(struct code)

```
struct linkedList
{
int number;
struct linkedList *prev,*next;
};
```

3. วิธีการสร้างวัตถุออกมาจากโครงสร้างที่กำหนดไว้ (struct example)

```
linkedList  *first; // ประกาศตัวแปรตามปกติก่อน
first = new linkedList; // การสร้างตัวแปรที่เป็น struct pointer ต้องมีการกำหนดค่าเริ่มต้นให้ตัวแปรก่อน จึงจะนำไปใช้ได้ โดยการใช้คีย์เวิร์ด new
first->number = 1; // การเข้าถึงองค์ประกอบที่อยู่ใน struct pointer ใช้เครื่องหมาย ->
first->prev= NULL; // คีย์เวิร์ด NULL หมายถึงว่างเปล่า ใช้กับการกำหนดตัวแปรที่เป็นวัตถุให้มีค่าว่าง
first->next= NULL; // คีย์เวิร์ด NULL หมายถึงว่างเปล่า ใช้กับการกำหนดตัวแปรที่เป็นวัตถุให้มีค่าว่าง
```

สรุปทั้ง 3 ขั้นตอนได้

Struct pattern	linkedList เป็นโครงสร้างที่ประกอบไปด้วย - number เป็นข้อมูลเลขจำนวนเต็ม - prev เป็นตัวแปรโครงสร้างพอยเตอร์ที่ชี้ลิงก์ลิสตัวถัดไป next เป็นตัวแปรโครงสร้างพอยเตอร์ที่ชี้ลิงก์ลิสตัวถัดไป

Struct code	struct linkedList
	{
	int number;
	struct linkedList *prev,*next;
	};

Struct example	linkedList *first;
	first = new linkedList;
	first->number = 1;

	first->prev = NULL; first->next = NULL;

ต่อมาเราจะทำการเขียนโปรแกรมที่ต้องนำเอาวัตถุเข้ามาใช้

เมื่อเราทำการสร้าง struct เสร็จแล้วตามขั้นตอน 3 ขั้นตอนด้านบน

(เราสร้างโรงงานสร้างลิงก์ลิสสำหรับเลขจำนวนเต็ม 1 จำนวนไว้แล้ว ดังนั้นต่อไปเราจะทำอะไรที่เกี่ยวกับลิงก์ลิสที่เก็บเลขจำนวนเต็มก็ใช้โรงงานนี้)

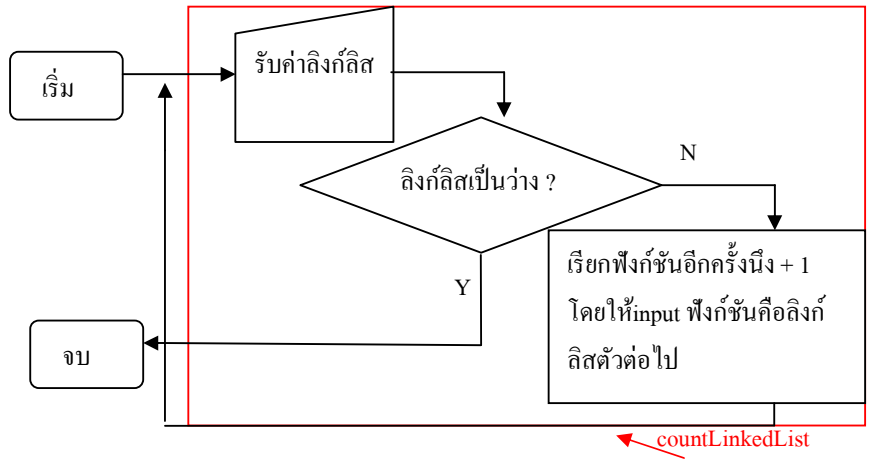
เมื่อเราจะเขียนโปรแกรม เราก็ต้องทำการวิเคราะห์โจทย์ด้วย 8 ขั้นตอนที่เราเรียนไว้แล้วตอนต้น

โจทย์คือ ให้เขียนโปรแกรมคำนวณหาจำนวน node ของลิงก์ลิสในลิงก์ลิส 1 ลิส (node คือแต่ละส่วนของลิงก์ลิสหรือลิงก์ลิสย่อยๆนั่นเอง)

วิธีที่ 1 เขียนแบบ recursive function นับไปข้างหน้า จะได้

Purpose	คำนวณหาจำนวน node ของลิงก์ลิสในลิงก์ลิส 1 ลิส
Input	ลิงก์ลิส 1 ลิงก์ลิส
Output	จำนวน node ทั้งหมดในลิงก์ลิส
Contract	countLinkedList : linkedList->int

Example	linkedList *first,*second;
	first = new linkedList;
	second = new linkedList;
	first->number = 1;
	first->next =second;
	second->number = 2;
	second->next = NULL;
	countLinkedList (first) = 2

Flowchart	

Function code	int countLinkedList (linkedList lList)
	{
	if (lList!=NULL) // ตรวจสอบว่าวัถตุเป็นว่างหรือไม่
	{
	return 0;
	}else
	{
	return 1 + countLinkedList(lList->next); // ให้ input คือ ลิสตัวถัดไป
	}
	}

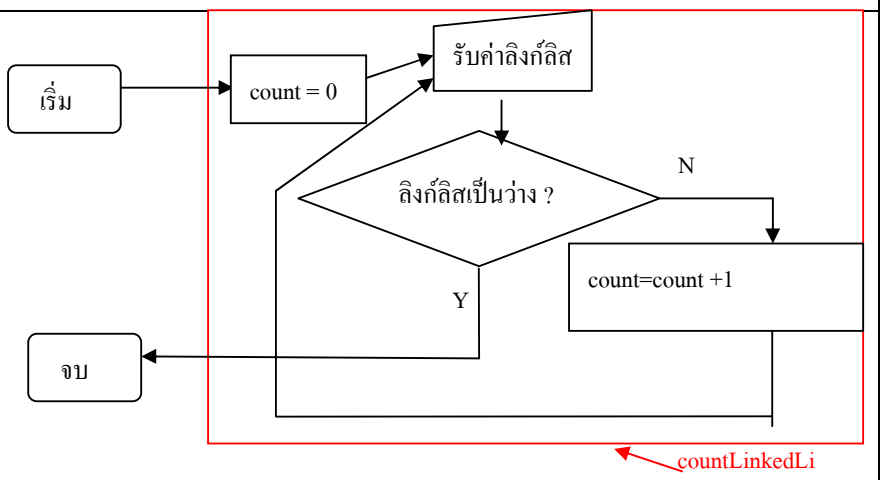
Main function code	int main()
	{
	linkedList *first,*second;
	first = new linkedList;
	second = new linkedList;
	first->number = 1;
	first->prev =NULL;
	first->next =second;
	second->number = 2;
	second->prev = first;
	second->next = NULL;
	cout<<countLinkedList (first) <<endl;
	return 0;
	}

จะเห็นว่าตัวแปรที่ใส่เป็น input ให้กับฟังก์ชัน countLinkedList คือ first เนื่องจากว่า first เป็นชื่อตัวแปรลิงก์ลิสลำดับแรกสุด จะทำให้ได้นับตั้งแต่ node แรก

วิธีที่ 2 เขียนแบบ ใช้ loop programming นับถอยหลัง จะได้

Purpose	คำนวณหาจำนวน node ของลิงก์ลิสในลิงก์ลิส 1 ลิส
Input	ลิงก์ลิส 1 ลิงก์ลิส
Output	จำนวน node ทั้งหมดในลิงก์ลิส
Contract	countLinkedList : linkedList->int

Example	linkedList *first,*second;
	first = new linkedList;
	second = new linkedList;
	first->number = 1;
	first->next =second;
	second->number = 2;
	second->next = NULL;
	countLinkedList (first) = 2

Flowchart	

Function code	int sumLinkedList (linkedList lList)
	{
	int count= 0;
	while(lList!=NULL) // ตรวจสอบว่าวัถตุเป็นว่างหรือไม่
	{
	count=count+1 ; // บวกค่า count เพิ่มอีก 1
	lList=lList->prev; // เลื่อนพอยเตอร์ไปยังลิงก์ลิสก่อนหน้า
	}
	return sum;
	}

Main function code	int main()
	{
	linkedList *first,*second;
	first = new linkedList;
	second = new linkedList;
	first->number = 1;
	first->next =second;
	second->number = 2;
	second->next = NULL;
	cout<<sumLinkedList (second) <<endl;
	return 0;
	}

จะเห็นว่าตัวแปรที่ใส่เป็น input ให้กับฟังก์ชัน countLinkedList คือ secondเนื่องจากว่า first เป็นชื่อตัวแปรลิงก์ลิสลำดับสุดท้าย เราต้องการแสดงให้เห็นว่าการเลื่อนพอยเตอร์ไปยังตัวแปรก่อนหน้านั้นทำได้

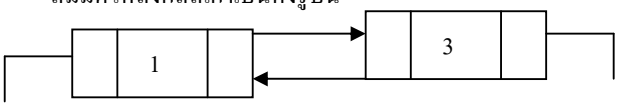
การแทรกลิงก์ลิส

การแทรกลิงก์ลิสที่เป็นแบบชี้สองทาง มีหลักการง่ายๆ คือ

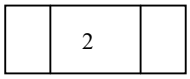
> next ของลิงก์ลิสใหม่ จะเท่ากับ next ของลิงก์ลิสก่อนหน้า
> prev ของลิสก์ใหม่ จะเท่ากับ ลิงก์ลิสก่อนหน้า (หรือเท่ากับ prev ของลิงก์ลิสข้างหลัง)
> next ของลิงก์ลิสก่อนหน้า จะเท่ากับ ลิงก์ลิสใหม่
> prev ของลิสก์ข้างหลัง จะเท่ากับ ลิงก์ลิสใหม่

ความหมายสามารถอธิบายได้ดังรูป

สมมติให้ลิงก์ลิสเก่าเป็นดังรูปนี้

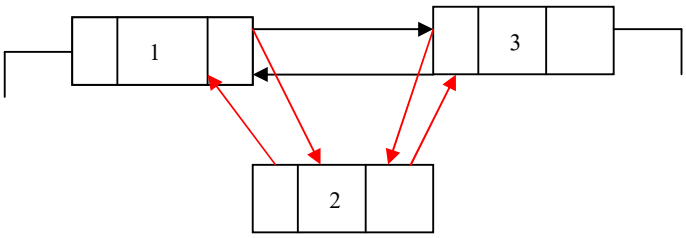


และลิงก์ลิสใหม่ เป็นดังนี้

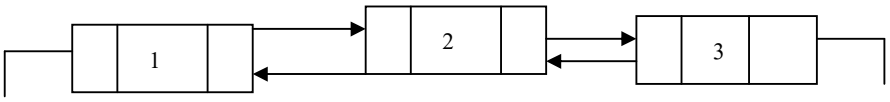


เราจะนำลิงก์ลิสใหม่ไปแทรกระหว่างเลข 1 กับ 2

เพราะฉะนั้นจะได้ดังรูปนี้



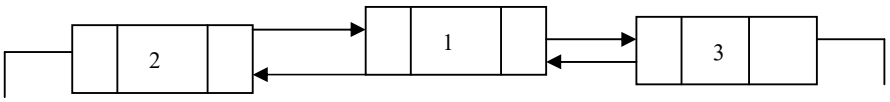
หมายเหตุเส้นสีแดงคือเส้นใหม่ตามหลักการด้านบน
ก็จะได้เป็นดังนี้



แต่จะมีข้อยกเว้นอยู่ 1 กรณี คือ กรณีที่ลิงก์ลิสใหม่ถูกแทรกให้อยู่ด้านหน้าสุด ก็จะได้หลักการดังนี้

> Prev ของลิงก์ลิสข้างหลัง จะเท่ากับ ลิงก์ลิสใหม่
> next ของลิงก์ลิสใหม่ จะเท่ากับ ลิงก์ลิสข้างหลัง

เพราะะนั้นจะได้เป็นดังนี้

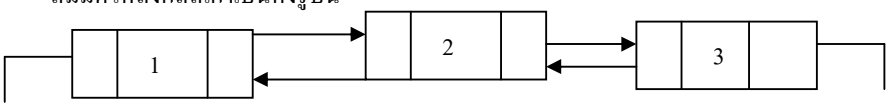


การลบลิงก์ลิส

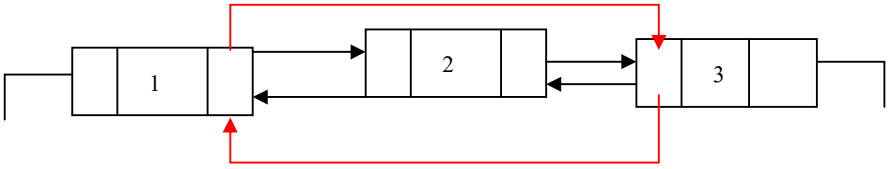
การลบลิงก์ลิสที่เป็นแบบชี้สองทาง มีหลักการง่ายๆ คือ

> prev ของลิงก์ลิสข้างหลัง จะเท่ากับ prev ของลิงก์ลิสตัวที่ถูกลบ
> next ของลิงก์ลิสตัวก่อนหน้า จะเท่ากับ next ของลิงก์ลิสตัวที่ถูกลบ

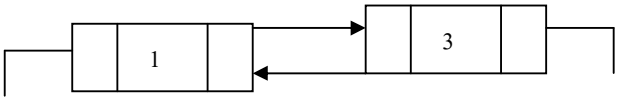
ความหมายสามารถอธิบายได้ดังรูป
สมมติให้ลิงก์ลิสเก่าเป็นดังรูปนี้



แล้วทำการลบตำแหน่งที่ 2 ก็จะได้เป็นดังนี้



หมายเหตุเส้นสีแดงคือเส้นใหม่ตามหลักการด้านบน
ก็จะได้เป็นดังนี้



แต่จะมีข้อยกเว้นอยู่ 1 กรณี คือ กรณีที่ลบลิงก์ลิสที่อยู่ด้านหน้าสุด ก็จะได้หลักการดังนี้

prev ของลิงก์ลิสข้างหลัง จะเท่ากับค่าว่าง

ลิงก์ลิสตัวแรก จะเท่ากับ next ของลิงก์ลิสตัวแรก

เพราะฉะนั้นจะได้เป็นดังนี้