บทที่ 10 การพัฒนาโปรแกรมเพื่อใช้งานจริง

ในบทนี้ จะกล่าวถึงปัจจัยต่างๆที่จำเป็นต้องคำนึงถึงในการพัฒนาโปรแกรมเพื่อใช้งานจริง ซึ่งโดยปกติ จะมีขนาดใหญ่ และมีขั้นตอนการทำงานที่ซับซ้อน ตลอดจนประกอบด้วยส่วนโปรแกรมย่อยเป็นจำนวนมาก ในกรณีนี้ สิ่งแรกที่ควรจะทำคือ การออกแบบโปรแกรม จากนั้นก็เป็นสร้าง Source code โดยแยกเป็นเป็นหลายไฟล์ ต่อจากนั้น ก็เป็นการ debug และขั้นตอนสุดท้ายคือ การทำให้โปรแกรมมีประสิทธิภาพสูงสุด นอกจากนี้ ยังจะกล่าวถึงคุณสมบัติของผู้เขียนโปรแกรมที่ดี และการเขียนโปรแกรมเชิงวัตถุเบื้องต้น เพื่อเปรียบเทียบกับการเขียนโปรแกรมด้วยภาษา C

10.1 การออกแบบโปรแกรม

เพื่อให้การเขียนโปรแกรมขนาดใหญ่เป็นไปอย่างมีประสิทธิภาพ จำเป็นต้องมีการวางแผนและออกแบบโปรแกรมให้รัดกุม มิฉะนั้น อาจเกิดปัญหาอันได้แก่ โปรแกรมไม่สมบูรณ์ โปรแกรมมีความซ้ำซ้อน โปรแกรมบางส่วนขาดประสิทธิภาพโปรแกรมแต่ละส่วนทำงานไม่สัมพันธ์กัน มีข้อผิดพลาดในการรันโปรแกรม เป็นต้น ซึ่งส่งผลต่อประสิทธิภาพในการเขียนโปรแกรมและการรันโปรแกรม

ด้วยเหตุนี้ ก่อนที่จะเริ่มลงมือเขียนโปรแกรม จึงควรปฏิบัติตามขั้นตอนต่อไปนี้

1. กำหนดขอบเขตการทำงานของโปรแกรม อันได้แก่ ข้อมูลส่งเข้า และข้อมูลส่งออก
2. ศึกษาวิธีการแก้ปัญหาโจทย์แล้วทำการเลือกอัลกอริธึมสำหรับคำนวณหาข้อมูลส่งออกจากข้อมูลส่งเข้า
3. ออกแบบโครงสร้างข้อมูลสำหรับใช้แทนข้อมูลที่ใช้ประกอบการทำงานของโปรแกรม เช่น โครงสร้างข้อมูลสำหรับข้อมูลส่งเข้าและข้อมูลส่งออก โครงสร้างข้อมูลสำหรับข้อมูลประกอบการคำนวณ เป็นต้น
4. ออกแบบโครงสร้างของโปรแกรมให้ประกอบด้วยหลายโปรแกรมย่อย โดยเขียนผังโครงสร้างประกอบการออกแบบ พร้อมกำหนดข้อมูลส่งเข้าตลอดจนข้อมูลส่งออกของแต่ละโปรแกรมย่อย
5. ลงมือเขียนแต่ละโปรแกรมย่อยแยกกัน ควบคู่กับทดสอบการทำงาน เพื่อให้แน่ใจว่าแต่ละโปรแกรมย่อยทำงานตามที่วางไว้
6. นำโปรแกรมย่อยมารวมกันแล้วทดสอบผลการทำงานร่วมกัน
7. ออกแบบชุดข้อมูลสำหรับทดสอบการทำงานของโปรแกรม
8. ในกรณีที่เกิดข้อผิดพลาดในการรันโปรแกรม ทำการวิเคราะห์ข้อมูลส่งออกที่ได้เพื่อนำมาแก้ไขข้อผิดพลาด

10.2 Source Code ที่มีหลายไฟล์

ในการเขียนโปรแกรมขนาดใหญ่ ควรแบ่ง Source Code เป็นหลายไฟล์ เพื่อให้ประสิทธิภาพในการเขียนโปรแกรมสูงขึ้น อันเนื่องจาก

- การเรียบเรียงและแก้ไขไฟล์ขนาดเล็กจะง่ายกว่าไฟล์ขนาดใหญ่
- หน่วยความจำที่ใช้สำหรับไฟล์ขนาดเล็กจะน้อยกว่า
- การหาที่ผิดในไฟล์ขนาดเล็กจะใช้เวลาสั้นกว่า
- ง่ายต่อการแบ่งงานกัน
- โปรแกรมคอมไพเลอร์ส่วนใหญ่ในปัจจุบันมี make utility ซึ่งเป็นโปรแกรมที่คอยตรวจสอบว่า มีไฟล์ใดบ้างที่ได้รับการแก้ไขหลังการคอมไพล์ครั้งสุดท้าย แล้วจะทำการคอมไพล์เฉพาะไฟล์ที่ได้รับการแก้ไขเท่านั้น ส่งผลให้กระบวนการคอมไพล์เป็นไปอย่างมีประสิทธิภาพมากขึ้น เพราะสามารถหลีกเลี่ยงการคอมไพล์ส่วนโปรแกรมที่ไม่มีการเปลี่ยนแปลงได้

- โปรแกรมคอมไพเลอร์ส่วนใหญ่ในปัจจุบันสนับสนุนการสร้าง Source Code ในลักษณะ Project กล่าวคือ เป็นการรวมไฟล์ source ไฟล์ header และส่วนประกอบอื่นๆ ที่ประกอบเป็นโปรแกรมเข้าด้วยกัน เพื่อให้ง่ายต่อการจัดการ

ในการคอมไพล์ Source Code ที่ประกอบด้วยหลายไฟล์ โปรแกรมคอมไพเลอร์จะทำการแปลไฟล์แต่ละไฟล์เป็นไฟล์ Object ก่อนจะนำมารวมกันแล้วเชื่อมต่อกับ Library เพื่อสร้างเป็นโปรแกรมสั่งงาน ข้อควรระวังในที่นี้คือ ถ้าโปรแกรมคอมไพเลอร์ไม่สามารถแปลส่วนใดส่วนหนึ่งของไฟล์ๆหนึ่ง ก็จะไม่สามารถสร้างไฟล์ Object ได้ ส่งผลให้ไม่สามารถสร้างเป็นโปรแกรมได้ ตัวอย่างกรณีเช่นนี้ได้แก่ มีการเรียกใช้ฟังก์ชันหรือตัวแปรที่ถูกประกาศในไฟล์อื่น ซึ่งสามารถแก้ไขด้วยการประกาศฟังก์ชันหรือตัวแปรในไฟล์นั้นๆ อันอาจเป็นการยุ่งยากหากมีไฟล์เป็นจำนวนมาก วิธีที่นิยมใช้เพื่อหลีกเลี่ยงปัญหายุ่งยากดังกล่าวคือ การสร้างไฟล์ Header ที่รวมการประกาศรูปแบบของฟังก์ชัน และตัวแปร แล้วใช้คำสั่ง #include ในส่วน Preprocessor เพื่อสั่งให้รวมไฟล์ Header ที่สร้างขึ้นดังกล่าว ถ้าหากทำเช่นนี้ ก็จะแก้ปัญหาข้อผิดพลาดในการคอมไพล์ได้

นอกจากนี้ ในการแบ่ง Source Code เป็นหลายไฟล์ ควรแบ่งเป็นหมวดหมู่ให้เหมาะสมตามการทำงานร่วมกัน เพื่อให้ง่ายต่อการเขียนโปรแกรมและการทดสอบการทำงาน ยิ่งไปกว่านั้น ยังสามารถนำไฟล์ที่เขียนขึ้นไปใช้ร่วมกับโปรแกรมอื่นที่มีส่วนประกอบเดียวกันได้ ยกตัวอย่างเช่น ส่วนแก้สมการเมตริกซ์สามารถนำไปใช้ทั้งในโปรแกรมแก้สมการอนุพันธ์ และในโปรแกรมแก้สมการปริพันธ์ เป็นต้น

กล่าวโดยสรุปแล้ว ข้อพึงปฏิบัติในการแบ่ง Source Code เป็นหลายไฟล์ได้แก่

- สร้างไฟล์ Header ที่รวมการประกาศฟังก์ชัน การกำหนดนิยามของข้อมูลชนิดใหม่ การกำหนดนิยามของค่าคงที่ และการประกาศตัวแปร ทั้งนี้ ควรจัดเป็นหมวดหมู่ให้เหมาะสม เพื่อหลีกเลี่ยงความซ้ำซ้อน
- สร้างไฟล์ Source โดยจัดให้ฟังก์ชันที่ทำงานร่วมกันอยู่ในไฟล์เดียวกัน ทั้งนี้ ควรยึดตามส่วนประกอบในผังโครงสร้างเป็นหลัก
- ที่ส่วนหัวของแต่ละไฟล์ Source ควรมีคำอธิบายเกี่ยวกับส่วนประกอบในไฟล์นั้นๆโดยสังเขป เพื่อให้ง่ายต่อการค้นหาส่วนประกอบที่ต้องการ
- ขนาดของไฟล์ Source ไม่ควรเกิน 200-300 บรรทัด เพื่อให้ง่ายต่อการเรียบเรียงแก้ไข นอกจากนี้ ยังสามารถลดปริมาณความซ้ำซ้อน ในกรณีที่นำไปใช้ในโปรแกรมอื่นได้อีกด้วย

10.3 การดีบัก (Debugging)

การดีบัก (Debugging) หมายถึง การกำจัดบัก (Bug) หรือข้อผิดพลาดของโปรแกรม ซึ่งข้อผิดพลาดเหล่านี้มีสาเหตุหลายประการ เช่น อัลกอริธึมที่ไม่ถูกต้อง การเขียนโปรแกรมที่ไม่ถูกหลักการ การใช้หน่วยความจำมากเกินไป การใช้คำสั่งที่ไม่เหมาะสม เป็นต้น

ข้อผิดพลาดของโค้ด (Code Errors) หมายถึง ข้อผิดพลาดที่เกิดจากความผิดพลาดในการเขียนโปรแกรม ได้แก่

- สัญลักษณ์ เช่น ;, {, (,), /*, */ etc. มีน้อยหรือมากไป
- ไม่ได้ทำการกำหนดค่าเริ่มต้นของตัวแปรก่อนใช้
- ใช้รูปแบบที่ไม่เหมาะสมกับฟังก์ชัน scanf หรือ printf เช่น ใช้ %d ในการแสดงค่าของตัวแปรชนิด float ใช้ %f ในการรับค่าของตัวแปรชนิด double
- อ้างอิงถึงตัวประกอบของอาร์เรย์ ที่ไม่มี เช่นอาร์เรย์a มีจำนวนตัวประกอบเพียง 10 ตัว แต่อ้างอิงถึง a[10] เป็นต้น

- ใช้ตัวดำเนินการไม่เหมาะสม ซึ่งโดยมากจะเป็นการใช้ผิดระหว่าง = กับ ==, & กับ &&, | กับ || หรืออื่น ๆ หรือไม่ก็ลำดับก่อนหลังของตัวดำเนินการไม่ถูกต้อง
- การเรียกใช้ฟังก์ชันไม่ถูกต้อง ซึ่งโดยมากจะเกิดจากการส่งอาร์กิวเมนต์ที่ไม่ถูกต้อง เช่น จำนวนไม่ได้ตามนิยามของฟังก์ชันหรือชนิดของค่าที่ส่งไม่เป็นไปตามที่กำหนดไว้

ข้อผิดพลาดขณะรัน (Runtime Error) หมายถึงข้อผิดพลาดที่เกิดขึ้นระหว่างการรันโปรแกรม ได้แก่

- ข้อผิดพลาดเกี่ยวกับเลขทศนิยม (Floating point error) หมายถึง ข้อผิดพลาดที่เกิดขึ้นในการคำนวณเลขทศนิยม ซึ่งโดยมากจะเกิดจากการหารด้วย 0 (Divide by 0), ข้อผิดพลาด Overflow และ Underflow, ข้อผิดพลาดปัดเศษ (Round-Off Error)
- Segmentation Violation ข้อผิดพลาดประเภทนี้จะเกิดขึ้นจากการที่โปรแกรมพยายามอ้างอิงถึงหน่วยความจำที่บันทึกค่าที่ผิดปกติ
- Stack Overflow เกิดจากการที่โปรแกรมใช้จำนวนตัวแปรหรือปริมาณหน่วยความจำมากเกินกว่าที่จะทำได้ และในบางครั้ง อาจหมายถึง โปรแกรมใหญ่เกินไป หรือฟังก์ชันที่เรียกใช้ในขณะนั้นใหญ่เกินไป หรือใช้อาร์เรย์ที่มีขนาดใหญ่มาก แต่โดยส่วนใหญ่ เกิดขึ้นใน ฟังก์ชันเรียกตัวเอง (Recursive Function) ที่โปรแกรมไม่สามารถไปถึงจุดสิ้นสุดการเรียกตัวเองได้
- การสิ้นสุดการทำงานของโปรแกรมอย่างผิดปกติ (Abnormal program termination) เกิดขึ้นจากการที่หน่วยความจำที่มีอยู่ไม่เพียงพอต่อการรันโปรแกรม

วิธีเขียนโค้ดสำหรับป้องกันไม่ให้เกิดข้อผิดพลาด

- เขียนโปรแกรมในลักษณะที่อ่านง่าย อันได้แก่
  - ♦ ตั้งชื่อตัวแปร และฟังก์ชันให้สื่อความหมาย
  - ♦ เขียนคอมเมนต์ (Comment) ในส่วนของโปรแกรมที่อาจก่อให้เกิดความสับสนได้ง่าย
  - ♦ เขียนโปรแกรมโดยเว้นช่องว่าง และขึ้นย่อหน้าให้ชัดเจน
- ใช้ { } ในการกำหนดบล็อกของ if, else, for, while, switch และอื่น ๆ เพื่อให้เกิดความสับสนในการเขียนน้อยที่สุด
- ไม่ควรเขียนเป็นบรรทัดที่ยาวเกินไป คำสั่งที่ยาวเกินไป ฟังก์ชันที่ยาวเกินไป หรือฟังก์ชันที่มีจำนวนพารามิเตอร์มาก
- โดยทั่วไป ไม่ควรเขียนหลายๆ คำสั่งใน 1 บรรทัด
- ในการใช้อาร์เรย์ขนาดใหญ่ เช่น array1[100000] ควรจะใช้ Pointer และฟังก์ชัน *malloc (or calloc)* ในการจองเนื้อที่ในหน่วยความจำสำหรับใช้เก็บอาร์เรย์นี้ แทนที่จะกำหนดเป็นอาร์เรย์ธรรมดาโดยใช้การจองเนื้อที่แบบ static (Static Memory Allocation)

```
int    array1[10000] ;            /* array declaration */
/* pointer and malloc */
int    *p_array1 ;
p_array1 = (int)malloc(10000 * sizeof(int)) ;
if ( p_array1 == NULL ) {
       printf("Not enough memory\n") ;
       exit(1) ;
}
```

- ถ้าเป็นไปได้ ควรใช้ชนิดข้อมูล double แทน float ในการคำนวณเลขทศนิยม เพื่อป้องกันข้อผิดพลาดอันเนื่องจากการคำนวณเลขทศนิยม เช่น ข้อผิดพลาด Overflow ข้อผิดพลาด Underflow เป็นต้น
- กรณีที่ไม่แน่ใจลำดับก่อนหลังของตัวดำเนินการควรใช้สัญลักษณ์วงเล็บช่วย

10.4 Optimization

Optimization ในการเขียนโปรแกรม หมายถึง การเขียนโปรแกรมให้มีประสิทธิภาพสูงที่สุดเท่าที่ทำได้ ซึ่งโดยทั่วไป จะมุ่งไปถึงการใช้เนื้อที่ในหน่วยความจำให้เกิดประโยชน์สูงสุด การพัฒนาโปรแกรมให้สามารถรันได้รวดเร็วที่สุด

How to optimize

- ลำดับของ Loop ในกรณีที่มีหลาย Loop ซ้อนกันอยู่ Loop ที่ซับซ้อนมากที่สุดควรจะอยู่ในสุด และ Loop ที่มีความซับซ้อนน้อยสุดควรจะอยู่นอกสุด
- ลด Cost ที่ใช้ในการรันโปรแกรม ถ้าสามารถสั่งให้โปรแกรมทำงานเดียวกันได้ โดยลดปริมาณงานลง ก็จะทำให้โปรแกรมรันได้รวดเร็วขึ้น (ดูตารางที่ 10.1)

ตารางที่ 10.1 ปริมาณงานสัมพัทธ์ของการดำเนินงาน (Relative cost of operations)

การดำเนินงาน	ปริมาณงานสัมพัทธ์
printf, scanf	1000
Trigonometric functions (sin, cos)	500
Floating point (any operation)	100
Integer divide	30
Integer multiple	20
Function call	10
Simple array index	6
Shifts	5
Add/substract	5
Pointer dereference	2
Bitwise and, or, not	1
Logical and, or, not	1

- Powers of 2 ถ้าสามารถใช้ Power of 2 ในการเขียนโปรแกรมได้ จะสามารถทำให้โปรแกรมทำงานได้เร็วขึ้น
- Pointer ควรใช้ Pointer แทนอาร์เรย์เพราะการอ้างอิงถึง Pointer โดยตรง จะเร็วกว่าที่โปรแกรมจะต้องอ้างอิงจาก Index ของ Array
- Macros ในกรณีที่เป็นงานที่ไม่ซับซ้อน การกำหนดเป็น Macro จะดีกว่าฟังก์ชันเพราะจะได้ไม่ต้องเสียเวลาในการเรียกใช้ฟังก์ชัน(แต่ทั้งนี้ จะต้องระวังเรื่องการใช้ เพราะ Macro ไม่มีการตรวจสอบชนิดข้อมูลของอาร์กิวเมนต์)

โปรแกรมคอมไพเลอร์ส่วนใหญ่ในปัจจุบันได้รวมส่วน Optimization ไว้แล้ว ซึ่งโดยมากเป็นการจัดหน่วยความจำให้โปรแกรมสามารถคำนวณได้เร็วที่สุด ผู้เขียนโปรแกรมสามารถเลือกที่จะให้โปรแกรมทำการ optimize หรือไม่ทำก็ได้ แต่โดยทั่วไป โปรแกรมหลังผ่านการตรวจสอบข้อผิดพลาดที่พร้อมใช้งานควรได้รับการ optimize เพื่อให้สามารถทำงานได้อย่างมีประสิทธิภาพสูงสุด

10.5 คุณสมบัติของผู้เขียนโปรแกรมที่ดี

ผู้เขียนโปรแกรมที่ดี นอกจากจะต้องมีความรู้และประสบการณ์ที่เพียงพอต่อการพัฒนาโปรแกรมแล้ว ควรจะมีคุณสมบัติดังต่อไปนี้

- มีการวางโครงร่างของโปรแกรมก่อนที่ลงมือเขียน เพื่อให้การพัฒนาโปรแกรมเป็นไปอย่างมีประสิทธิภาพ
- มีการซ่อนส่วนย่อยของโปรแกรมไว้ ที่เรียกกันว่า *การซ่อนปลีกย่อย* (*Encapsulation*) กล่าวคือ รายละเอียดปลีกย่อยของโปรแกรมจะถูกซ่อนไว้ โปรแกรมจะแสดงเฉพาะส่วนหลัก ๆ เท่านั้น คล้าย ๆ กับที่ส่วนประกอบ (Component) ของระบบสเตอริโอ มีส่วนเชื่อมต่อคือ ส่วนตัวเชื่อมต่อ (Connector) เท่านั้น ผู้ใช้

ไม่จำเป็นต้องไปยุ่งกับสายไฟในเครื่อง ทั้งนี้ เพื่อให้ผู้ใช้สามารถใช้หรือแก้ไขโปรแกรมได้โดยไม่ต้องรู้รายละเอียดทั้งหมดของโปรแกรม เป็นการอำนวยความสะดวกแก่ผู้ใช้โปรแกรม อีกทั้งยังเป็นการป้องกันการผิดพลาดอันอาจเกิดจากความเข้าใจผิดได้อีกด้วย ยกตัวอย่าง เช่น อาจจะทำการแก้ไขชนิดของตัวแปรหรือโครงสร้างของข้อมูล โดยไม่รู้ว่า จะมีผลกระทบต่อโปรแกรมส่วนอื่น เป็นต้น อนึ่ง การโปรแกรมเชิงวัตถุ (Object-oriented programming) เป็นวิธีการโปรแกรมที่มุ่งเน้นให้สามารถบรรลุวัตถุประสงค์ดังกล่าวข้างต้นนี้

- ใช้หรือแก้ไขโปรแกรมที่มีอยู่แล้ว คุณสมบัติข้อนี้ เป็นสิ่งสำคัญในการพัฒนาโปรแกรม และทำให้เกิดงานสร้างสรรค์ในการเขียนโปรแกรมมากขึ้น
- ใช้ฟังก์ชันที่มีอยู่แล้วให้เกิดประโยชน์สูงสุด แทนที่จะต้องทำการเขียนฟังก์ชันใหม่ทุกครั้ง เราควรจะนำเอาสิ่งที่มีอยู่แล้วมาใช้ให้เกิดประโยชน์สูงสุดเสียก่อน กรณีนี้ เราจะต้องมีนิสัยในการเขียนโปรแกรมแบบ Modular Programming เพื่อให้เราทำการเขียนโปรแกรมเป็นฟังก์ชันย่อยให้มากที่สุดเท่าที่ทำได้
- เขียนโปรแกรมที่ง่ายต่อการทำความเข้าใจ เช่น ต้องมี Comment สำหรับอธิบายส่วนต่าง ๆ ของโปรแกรม เขียนให้อยู่ในรูปแบบที่อ่านง่าย ใช้ชื่อตัวแปรและฟังก์ชันที่เหมาะสม

10.6 การเขียนโปรแกรมเชิงวัตถุเบื้องต้น

การเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming; OOP) เป็นวิธีการเขียนโปรแกรม โดยอาศัยแนวคิดของวัตถุชิ้นหนึ่ง มีความสามารถในการปกป้องข้อมูล และการสืบทอดคุณสมบัติ ซึ่งทำให้แนวโน้มของ OOP ได้รับการยอมรับและพัฒนามาใช้ในระบบต่าง ๆ มากมาย เช่น ระบบปฏิบัติการ ระบบฐานข้อมูล และซอฟต์แวร์ขนาดใหญ่

ความเป็นมา

แนวความคิดดั้งเดิมของการเขียนโปรแกรมคือ การแก้ปัญหาโดยใช้คอมพิวเตอร์เป็นเครื่องมือ คล้ายกับการใช้เครื่องคิดเลขในการแก้ปัญหาทางคณิตศาสตร์

แนวความคิดแบบใหม่ที่ใช้ในการเขียนโปรแกรมคือ การเน้นที่ปัญหาและองค์ประกอบของปัญหา (Problem Space) โดยทำการสร้างวัตถุ เพื่อทำหน้าที่แก้ปัญหา แทนที่จะมองหาวิธีการแก้ปัญหานั้น ๆ หรือขั้นตอนในการแก้ปัญหา (Solution Space) ซึ่งเป็นวิธีการเขียนโปรแกรมแบบเก่านั่นเอง กล่าวโดยสรุปคือ แนวคิดใหม่นี้คล้ายกับการสร้างคนเพื่อช่วยแก้ปัญหาให้ แทนที่จะทำการแก้ปัญหาโดยตรงเอง

อลัน เคย์ (Alan Kay) ได้เสนอกฎ 5 ข้อ ที่เป็นแนวทางของ Object-Oriented Programming ไว้ดังนี้

1. ทุก ๆ สิ่งเป็นวัตถุ (Everything is an Object)
2. โปรแกรม ก็คือ กลุ่มของวัตถุที่ส่งข่าวสารบอกกันและกันให้ทำงาน (A Program is a Bunch of Object Telling Each Other What to do by Sending Messages)
3. ในวัตถุแต่ละวัตถุจะต้องมีหน่วยความจำและประกอบไปด้วยวัตถุอื่น ๆ (Each Object has Its Own Memory Made Up of Other Objects)
4. วัตถุทุกชนิดจะต้องจัดอยู่ในประเภทใดประเภทหนึ่ง (Every Object has a Type)
5. วัตถุที่จัดอยู่ในประเภทเดียวกันย่อมได้รับข่าวสารเสมือนกัน (All Object of a Particular Type Can Receive the Same Messages)

แนวคิด

หัวใจหลักของ OOP คือ “ธรรมชาติของวัตถุ” ซึ่งหมายความว่า OOP จะมองทุกสิ่งในฐานะ “วัตถุ” (Object) มันจะมีสีแดงหรือสีเขียว ยาวหรือสั้น มันก็คือวัตถุชิ้นหนึ่งเหมือนกัน และเราสามารถกำหนดประเภทหรือคลาส (Class) ให้กับวัตถุเหล่านั้นได้

นอกจากนี้ เมื่อ OOP มองทุกสิ่งเป็นวัตถุแล้ว ยังสามารถพัฒนาแนวคิดต่อไปอีกว่า **“วัตถุแต่ละอย่างนั้น ต่างมีลักษณะและวิธีการใช้งานเป็นของตัวเอง”** กล่าวคือ วัตถุแต่ละชนิดหรือแต่ละชิ้นต่างก็มีรูปร่าง ลักษณะ และการใช้งาน (การกระทำ) ที่แตกต่างกันออกไป เราจะเรียกคุณลักษณะของวัตถุว่า แอตทริบิวต์ (Attribute) และจะเรียกวิธีการใช้งานวัตถุว่า เมธทอด (Method) ตัวอย่างเช่น “ดินสอเป็นวัตถุที่มีลักษณะเรียวยาว ภายในเป็นไส้ถ่านใช้สำหรับเขียน การใช้ดินสอทำได้โดยใช้มือจับและเขียนลงบนวัสดุรองรับ” จากข้อกำหนดนี้ คุณลักษณะ (Attribute) ของวัตถุ “ดินสอ” คือ “ยาวเรียว ภายในเป็นไส้ถ่าน” ส่วนการใช้งาน (Method) คือ “ใช้มือจับและเขียนลงบนวัสดุรองรับ”

จะเห็นได้ว่าแนวคิดของ OOP นั้นจะมีลักษณะที่คล้ายกับธรรมชาติของสิ่งหนึ่งซึ่งสามารถแบ่งแยกสิ่งต่าง ๆ ออกเป็นแต่ละประเภทได้ ถ้านำเอาแนวคิดของ OOP มาใช้ในการเขียนโปรแกรมและการจัดการข้อมูล จะพบว่าโปรแกรมหรือฟังก์ชันจะมีความเป็นอิสระต่อกันอย่างเห็นได้ชัด กล่าวคือ โปรแกรมหรือฟังก์ชันแต่ละอันถึงแม้จะมาจากที่เดียวกันแต่สามารถทำงานในหน้าที่ต่างกัน เก็บข้อมูลต่างค่ากันได้ โดยไม่มายุ่งเกี่ยวกันแต่อย่างใด

ส่วนประกอบสำคัญของ OOP

เพื่อให้วัตถุที่สร้างขึ้นประกอบกันเป็นโปรแกรมที่ใช้แก้ปัญหาได้ OOP จำเป็นต้องมีส่วนประกอบสำคัญ ดังนี้

<u>การเชื่อมต่อ (Interface)</u>

อินเตอร์เฟส (Interface) หมายถึง การเชื่อมต่อ ถ้าเป็นการเชื่อมต่อระหว่างผู้ใช้กับคอมพิวเตอร์ จะเรียกการเชื่อมต่อนั้นว่า ยูสเซอร์อินเตอร์เฟส (User Interface) แต่ในการเขียนโปรแกรมเชิงวัตถุ การเชื่อมต่อยังรวมไปถึงวัตถุ (Object) เพราะในวัตถุจะต้องมีอินเตอร์เฟส อันเป็นส่วนที่วัตถุนั้น ๆ จะให้บริการหรือเป็นส่วนที่บอกว่าวัตถุนั้น ๆ สามารถทำอะไรได้บ้าง ซึ่งบางครั้งเรียกว่า เมธทอด (Method)

ข้อดีของการมีอินเตอร์เฟส ก็คือ การเปลี่ยนแปลงที่เกิดขึ้นภายในวัตถุจะไม่กระทบต่ออินเตอร์เฟส ดังนั้นภายในวัตถุผู้เขียนคำสั่งสามารถดัดแปลง แก้ไข หรือเพิ่มเติมได้ตลอดเวลา นอกจากนี้ ภายในวัตถุยังสามารถเก็บค่าต่าง ๆได้อีกด้วย

<u>การซ่อนรายละเอียด (Encapsulation)</u>

ส่วนประกอบของวัตถุตามแนวความคิดการเขียนโปรแกรมเชิงวัตถุ จะต้องประกอบไปด้วยสองส่วนเป็นอย่างน้อย คือ ส่วนของคุณสมบัติใช้เก็บข้อมูลรายละเอียด สถานะ โอยใช้ตัวแปรเก็บค่าต่าง ๆ ไว้ และส่วนของเมธทอดทีเป็นตัวเชื่อมการทำงานของวัตถุนั้น ๆ โดยผู้ใช้จะไม่สามารถติดต่อใช้งานกับตัวแปรที่อยู่ข้างในได้ ในภาษา C++ จะใช้คำ Public,Private และ Protected เข้ามาช่วยกำหนดขอบเขตการใช้

<u>การนำวัตถุมาใช้ใหม่ (Object Reuse)</u>

จุดประสงค์หลักของการเขียนโปรแกรมเชิงวัตถุ ก็คือ การนำส่วนต่าง ๆ ของวัตถุที่สร้างขึ้นกลับมาใช้ใหม่ หรือที่เรียกในภาษาอังกฤษว่า “Reuse” เมื่อผู้เขียนโปรแกรมสร้างวัตถุมีจำนวนมากพอก็สามารถนำวัตถุที่สร้างขึ้นมา ประกอบเป็นวัตถุใหม่ หรือที่เรียกว่าคอมโพสิตชั่น “Composition”

นอกจากวิธีการคอมโพสิตแล้ว ผู้ใช้ยังสามารถ Reuse ส่วนของวัตถุโดยการใช้การสืบทอดคุณสมบัติ

(Inheritance) จากคลาส ลักษณะเช่นนี้ คือ เป็นการนำส่วนของวัตถุทั้งหมดมาใช้ ซึ่งปกติแล้ววัตถุที่นำมาใช้ในลักษณะนี้จะมีขนาดใหญ่ ถ้าเป็นการคอมโพสิตจะประกอบขึ้นจากส่วนของวัตถุที่มีขนาดเล็กกว่า อย่างไรก็ตาม ขนาดของวัตถุมิได้เป็นตัวกำหนดที่แน่นอนตายตัวเสมอไป

การสืบทอดคุณสมบัติ (Inheritance)

หลักการของ “Inheritance” คือการที่ “Class” ใดๆยอมให้มีการสืบทอด (Inheritance) ทั้งคุณสมบัติ (Attributes) และกาทำงาน (Method) ไปยัง “Class” อื่น ยกตัวอย่างเช่น สมมุติว่าเรากำลังพิจารณาเรื่องโครงสร้างข้อมูลสำหรับเก็บข้อมูลนักศึกษา เราสามารถสร้างคลาสของนักศึกษาที่มี Attribute เช่น ชื่อ เลขทะเบียน ที่อยู่ เบอร์โทรศัพท์ และอื่นๆ อันเป็นข้อมูลพื้นฐาน และมี Method เช่น แสดงที่อยู่ พิมพ์ประวัติ และอื่นๆ จากนั้นสร้างคลาสของนักศึกษาคณะวิศวกรรมศาสตร์ที่สืบทอด Attribute จากคลาสนักศึกษา พร้อมเพิ่มเติม Attribute เช่น โค้ดใช้ห้องคอมพิวเตอร์ โค้ดใช้ Wi-Fi ของคณะฯ เป็นต้น พร้อมเพิ่ม Method เช่น การตรวจสอบโค้ด การเปลี่ยนแปลงโค้ด การตรวจสอบประวัติการใช้ Wi-Fi เป็นต้น การสืบทอดทำให้สะดวกต่อการสร้างโครงสร้างข้อมูลที่มีลักษณะเป็นลำดับชั้น และลดความซ้ำซ้อนของ Source Code เมื่อเปรียบเทียบกับการเขียนโปรแกรมในแนวคิดเดิม ซึ่งใช้ Structure จะเห็นว่า ถึงแม้จะเพิ่มส่วนประกอบอีกเพียงหนึ่งเดียว ก็ต้องทำการสร้าง Structure อันใหม่ขึ้น พร้อมกับฟังก์ชันที่ประกอบการใช้งาน Structure ใหม่ทั้งหมดทำให้เกิดความยุ่งยากและความซ้ำซ้อนของโปรแกรมมาก

การพ้องรูป (Polymorphism)

Polymorphism เป็นคำที่มาจากภาษากรีก “Polymorphic” โดย Poly หมายถึง Many (หลาย) Morphos หมายถึง Forms (รูปแบบ) ถ้าแปลตรงตัวจึงหมายถึง หลายรูปแบบ ซึ่งหมายถึง Method สามารถมีหลายรูปแบบได้ ยกตัวอย่างเช่น สมมุติว่าทำการสร้าง Method ชื่อว่า draw สำหรับวาดรูปทรงเรขาคณิตต่างๆ เช่น สามเหลี่ยม สี่เหลี่ยม วงกลม วงรี และอื่นๆ เวลาเรียก Method นี้จะต้องระบุรูปทรงเรขาคณิตที่จะวาด โดยรูปแบบการทำงานของ draw จะเปลี่ยนไปตามรูปทรงเรขาคณิต Polymorphism เป็นเครื่องมือที่มีประโยชน์มาก เพราะทำให้ง่ายต่อการทำความเข้าใจโปรแกรมมากขึ้น เนื่องจากใช้ Method ที่มีชื่อซ้ำกันได้ ในทางตรงกันข้าม ถ้าใช้ Structure จะต้องเขียนฟังก์ชันสำหรับ Structure แต่ละอัน และต้องใช้กำหนดชื่อฟังก์ชันให้ต่างกันด้วย ส่งผลให้เกิดความซับซ้อนมากขึ้น

การเขียนทับตัวดำเนินการ (Operator Overloading)

ตัวดำเนินการoverloading เป็นการกำหนดให้การทำงานของตัวดำเนินการ (Operator) เปลี่ยนไปตามตัวถูกดำเนินการ (Operand) ซึ่งเป็นส่วนหนึ่งของ Polymorphism ยกตัวอย่างเช่น วิธีการทำงานของตัวดำเนินการคูณ (*) เมื่อใช้กับจำนวนจริง จำนวนเชิงซ้อน เว็กเตอร์ หรือเมตริกซ์ล้วนแล้วต่างกันทั้งสิ้นตัวดำเนินการoverloading ทำให้การเขียนโปรแกรมเป็นไปอย่างมีประสิทธิภาพและเรียบง่ายขึ้น

การเขียนโปรแกรมและการออกแบบระบบงาน

ก่อนที่ผู้เขียนโปรแกรมจะสามารถเขียนคำสั่งได้ จะต้องมีการออกแบบระบบงานก่อนแล้วจึงเขียนโปรแกรมเป็นภาษาต่าง ๆ ตามชนิดของงานและความเหมาะสม การเขียนโปรแกรมเชิงวัตถุก็เช่นกัน ตะต้องมีการออกแบบระบบงานก่อน หลักสำคัญสำหรับการออกแบบเชิงวัตถุ ก็คือ การหาวัตถุให้พบ เมื่อพบแล้วจะต้องจำแนกวัตถุออกเป็นส่วนที่เปลี่ยนแปลงและส่วนที่อยู่คงที่ วัตถุที่ไม่เปลี่ยนแปลงสามารถนำไปใช้ได้เมื่อมีการปรับปรุงระบบงานใหม่ นั่นคือเหตุผลที่ทำให้ต้องมีการออกแบบระบบงาน วัตถุที่มีการเปลี่ยนแปลงบ่อยก็ได้แก่ วัตถุที่ทำ

หน้าที่เป็นอินเตอร์เฟส เป็นต้น

ประโยชน์ของการเขียนโปรแกรมแบบ OOP

เทคโนโลยีของ OOP ก่อให้เกิดประโยชน์ต่อการพัฒนาซอฟต์แวร์ ดังนี้

- ความสามารถในการเรียกใช้ได้หลายครั้ง ออบเจ็กต์ได้ถูกออกแบบตามหลักการที่ว่าสามารถเรียกใช้งานได้หลาย ๆ ครั้ง ในหลักการนี้ทำให้ Application ของ OOP ตัวแรกอาจจะทำได้ยาก แต่ว่าโปรแกรมแอปพลิเคชันที่เขียนภายหลังจะสร้างง่ายเพราะสามารถเรียกใช้ออบเจ็กต์ที่ถูกสร้างไว้ตั้งแต่โครงงานแรกได้
- ความเชื่อถือได้ โปรแกรมแอปพลิเคชันของ OOP จะมีความเชื่อถือได้สูงเพราะจะรวมเอาส่วนย่อยที่ทดสอบจนได้มาตรฐานแล้วมารวม เข้าไว้ด้วยกัน โค้ด (Code) ที่เขียนขึ้นมาใหม่ในแต่ละแอพลิเคชั่นจะมีไม่มากนัก เนื่องจากโค้ดส่วนใหญ่จะถูกดึงมาจากไลบรารีที่มีความเชื่อถือได้สูงอยู่แล้ว และในการโปรแกรมภาษา C++ ยังมีคุณประโยชน์อื่นอีก