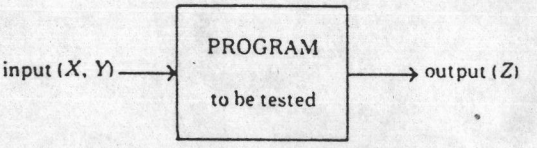
บทที่ 2

ทฤษฎีเกี่ยวกับวิธีการทดสอบโปรแกรม

2.1 บทนำ

ในการทดสอบโปรแกรมคอมพิวเตอร์นั้น ทำอย่างไรเราจึงจะแน่ใจได้ว่า โปรแกรมนั้นทำงานได้ถูกต้อง ซึ่งเป็นสิ่งสำคัญอย่างยิ่งในการทดสอบโปรแกรมคอมพิวเตอร์ วิธีการที่แน่นอนที่สุดก็คือ การทดสอบด้วยข้อมูลที่เป็นไปได้ทั้งหมด และตรวจสอบดูว่าได้ผลลัพธ์ที่ถูกต้องหรือไม่ ซึ่งโดยทั่วไปแล้ววิธีนี้ในทางปฏิบัติแทบเป็นไปไม่ได้เลย เพราะว่าจำนวนของข้อมูลเข้าที่เป็นไปได้ทั้งหมดนั้นมีจำนวนมหาศาล เราจะยกตัวอย่างเช่น สมมติว่าโปรแกรมที่จะทดสอบนั้นมีข้อมูลเข้าเป็นตัวแปรสองค่า และให้ผลลัพธ์เป็นค่าหนึ่งค่าดังรูปที่ 2.1



รูป 2.1 แสดงถึงโปรแกรมที่มีข้อมูลเข้าสองค่า และได้ผลลัพธ์หนึ่งค่า

ถ้าเรากำหนดค่าของข้อมูลเป็นตัวแปร X, Y และผลลัพธ์ที่ได้เป็นตัวแปร Z และเราจะทำการทดสอบโปรแกรมโดยกำหนดค่าที่เป็นไปได้ทั้งหมดของ X, Y เพื่อที่จะให้ได้ผลลัพธ์ที่ถูกต้องทุกกรณีซึ่งเราจะเห็นว่ามันเป็นไปได้ยากมาก โดยเราจะมาพิจารณาเฉพาะส่วนที่เป็นข้อมูลเข้าทั้งสองค่า ซึ่งจำนวนของข้อมูลที่เป็นไปได้ทั้งหมดนั้นมีจำนวนมหาศาล สมมติให้ตัวแปร X, Y เป็นตัวแปรที่เป็นค่าของจำนวนเต็ม (Integer) ซึ่งประมวลผลบนเครื่องคอมพิวเตอร์ 32 บิท ซึ่งจะมีจำนวนข้อมูลที่เป็นไปได้ทั้งหมดจำนวน $2^{32} \times 2^{32} = 2^{64}$ ค่า สำหรับค่า X, Y และสมมติให้โปรแกรมประมวลผลได้ 1 ไมโครวินาทีต่อ 1 ครั้ง ซึ่งต้องใช้เวลาหลายร้อยปีเพื่อจะให้การทดสอบนี้สมบูรณ์

จากตัวอย่างข้างบนที่ยกมาจะเห็นได้ว่าเราไม่สามารถใช้ข้อมูลจำนวนมากๆ เพื่อทดสอบความถูกต้องของโปรแกรมได้เลย ซึ่งโดยทั่วไปแล้วเราจะทดสอบโปรแกรมโดยอาศัยข้อมูลจำนวนหนึ่ง แล้วเปรียบเทียบผลลัพธ์ที่ได้ ถ้าผลลัพธ์ที่ได้ออกมาผิดนั่นแสดงว่า โปรแกรมนั้นมีข้อผิดพลาดเกิดขึ้น แต่อย่างไรก็ตามถ้าผลลัพธ์ที่ได้ถูกต้องเราก็บอกได้แต่เพียงว่าโปรแกรมนั้นเคยผ่านการทดสอบมาแล้ว และมีสถิติที่ถูกต้องในการทดสอบ

จะเห็นว่าเราไม่สามารถที่จะใช้ข้อมูลที่เป็นไปได้ทั้งหมดในการทดสอบโปรแกรม แต่เราจะเลือกเอาบางส่วนของข้อมูลเพื่อใช้ทดสอบโปรแกรม วิธีเลือกข้อมูลเพื่อใช้ในการทดสอบโปรแกรมนั้นโดยทั่วไปเราจะใช้สองวิธี คือ

1.วิธีการสุ่มข้อมูล (random) เป็นการสุ่มข้อมูลธรรมดาเพื่อให้ได้เพียงข้อมูลที่เราจะใช้ทดสอบเท่านั้น

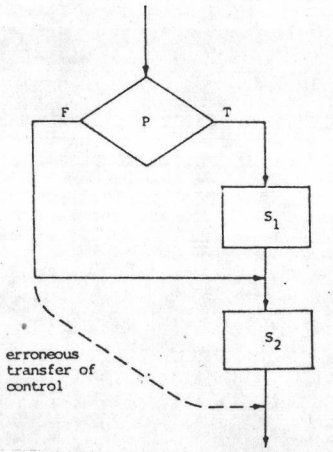
2.วิธีการเลือกข้อมูลโดยอาศัยโครงสร้างของโปรแกรม (Program structure) เป็นวิธีการเลือกข้อมูลโดยพิจารณาจากพื้นฐานโครงสร้างของโปรแกรม เพื่อให้ทุกๆ คำสั่งในโปรแกรมที่จะทดสอบผ่านการประมวลผลอย่างน้อยหนึ่งครั้ง ในระหว่างทดสอบโปรแกรมด้วยข้อมูลโดยอาศัยสาเหตุที่ว่า โปรแกรมโดยทั่วไปส่วนมากนั้น จะมีบางคำสั่งที่ไม่ถูกประมวลผล ซึ่งถ้าในโปรแกรมที่เราจะทำการทดสอบนั้นมีคำสั่งที่ผิดพลาดเกิดขึ้น และคำสั่งนั้นไม่ถูกประมวลผลในขณะที่ทำการทดสอบ เราก็ไม่สามารถที่จะหาข้อผิดพลาดของโปรแกรมนั้นได้เลย ดังนั้นวิธีการแก้ไขก็คือ ในขณะที่ทำการทดสอบโปรแกรมนั้นทุกๆคำสั่งในโปรแกรมจะต้องผ่านการประมวลผลอย่างน้อยหนึ่งครั้ง ซึ่งข้อมูลที่จะใช้ในการทดสอบเพื่อให้ทุกๆคำสั่งในโปรแกรมผ่านการประมวลผลอย่างน้อยหนึ่งครั้งนั้น เราจะเลือกข้อมูลโดยพิจารณาจากพื้นฐานโครงสร้างของโปรแกรมซึ่งจะกล่าวต่อไป

2.2 จุดสำคัญของการทดสอบโปรแกรม

จุดสำคัญในการทดสอบโปรแกรมที่จะกล่าวในที่นี้ก็คือ ทุกๆคำสั่งในโปรแกรมจะต้องผ่านการประมวลผลอย่างน้อยหนึ่งครั้งในระหว่างการทดสอบโปรแกรมด้วยข้อมูล และทุกๆทางเดินของสายงานต้องถูกสำรวจอย่างน้อยหนึ่งครั้งในระหว่างการทดสอบโปรแกรมด้วย

เราจะยกตัวอย่างง่ายๆในการอธิบาย เช่น ถ้าในโปรแกรมเราต้องการที่จะใช้คำสั่ง `x = x + y;` แต่ว่ามีข้อผิดพลาดเกิดขึ้น ซึ่งอาจเกิดจากการเขียนที่ไม่ชัดเจนทำให้คำสั่งกลายเป็น `x = x * y;` และถ้าคำสั่งนี้ไม่เคยถูกประมวลผลเลยในระหว่างการทดสอบโปรแกรมแล้วเราจะเห็นได้ว่าในโปรแกรมนั้นมีข้อผิดพลาดเกิดขึ้น แต่ว่าเราจะหาไม่พบเนื่องจากว่าคำสั่งผิดพลาดที่เกิดขึ้นนี้ไม่เคยถูกประมวลผลเลยในช่วงของการทดสอบโปรแกรม ดังนั้นจุดที่สำคัญในการทดสอบโปรแกรม คือ ทุกๆคำสั่งในโปรแกรมจะต้องถูกประมวลผลอย่างน้อยหนึ่งครั้งในระหว่างการทดสอบโปรแกรม

ข้อผิดพลาดที่เกิดขึ้นในโปรแกรมนั้น อาจเป็นข้อผิดพลาดที่เกิดจากการถ่ายโอนการควบคุมของโปรแกรมดังแสดงในรูป 2.2 ซึ่งจะเห็นว่า จะให้ผลลัพธ์ที่ถูกต้องเสมอตราบใดที่ตัวเงื่อนไข P เป็นจริง และจะเห็นว่าทุกๆคำสั่งก็ได้ถูกประมวลแล้วอย่างน้อยหนึ่งครั้งในกรณีที่ P เป็นจริงด้วย

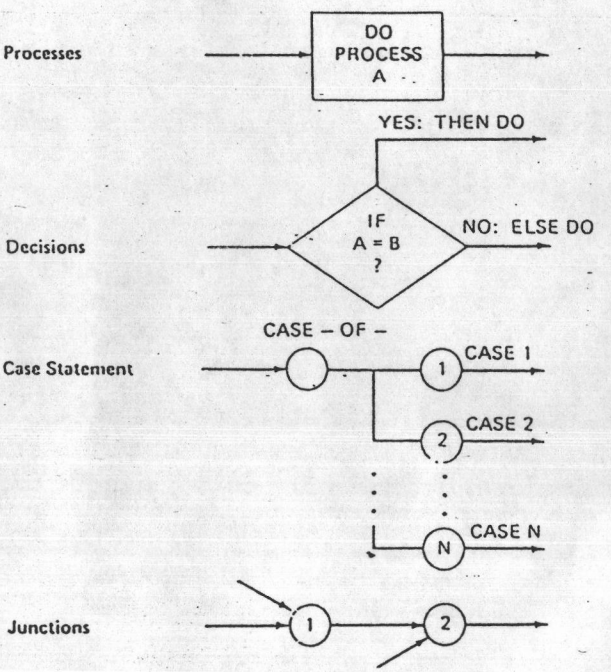


รูป 2.2 รูปโฟลว์ชาร์ตแสดงลักษณะของข้อผิดพลาดที่อาจเกิดขึ้นได้

ปัญหาที่เกิดขึ้นก็คือ สายงานบางช่วงของโปรแกรมไม่ถูกสำรวจเลยในระหว่างการทดสอบโปรแกรม ดังนั้นข้อผิดพลาดอาจเกิดขึ้นได้ถ้าสายงานบางส่วนไม่เคยถูกสำรวจเลย ดังนั้นจุดสำคัญอีกอย่างหนึ่งในการทดสอบโปรแกรมก็คือ ทุกๆทางเดินของสายงานจะต้องถูกสำรวจอย่างน้อยหนึ่งครั้งในระหว่างการทดสอบโปรแกรม

2.3 กราฟควบคุมสายงาน (Control Flowgraphs)

2.3.1 กราฟควบคุมสายงาน [1],[2] : เป็นกราฟรูปภาพ ที่ใช้แสดงลักษณะโครงสร้างการทำงานของโปรแกรม รูปแบบที่ใช้ดังรูปที่ 2.3.1



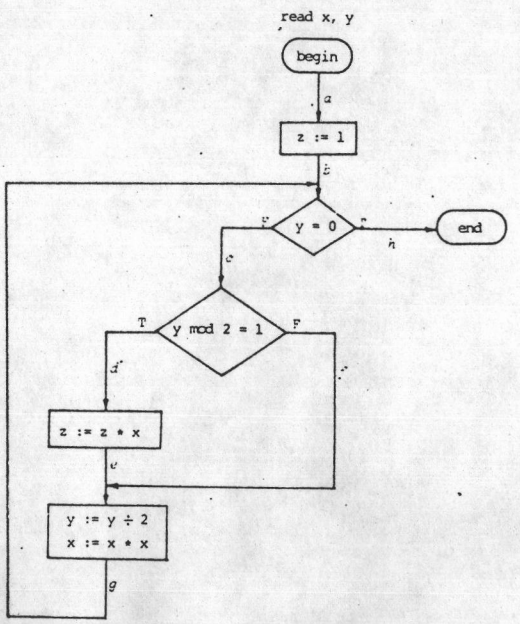
รูปที่ 2.3.1 แสดงรูปของโฟลว์กราฟ

2.3.2 **ชุดกระบวนการ** : เป็นชุดคำสั่งที่เรียงลำดับกัน ซึ่งถ้ามีคำสั่งใดคำสั่งหนึ่งถูกประมวลผล แล้วทุกๆคำสั่งในชุดกระบวนการก็จะถูกประมวลผลทั้งหมดด้วย จะมีลักษณะเป็นมีทางเข้าทางเดียว และทางออกทางเดียว

2.3.3 **การตัดสินใจ** : เป็นจุดเปลี่ยนการควบคุมในโปรแกรม ซึ่งอาจเป็นการกระโดดไปตามเงื่อนไขที่ต้องการ หรืออาจข้ามบางคำสั่งตามเงื่อนไข โดยปกติแล้วจะมีลักษณะทางเลือกสองทางคือ จริงและเท็จ

2.3.4 **คำสั่งเงื่อนไข** : เป็นลักษณะที่มีการตัดสินใจทางเลือกหลายทาง ขึ้นอยู่กับตัวเงื่อนไขที่กำหนด โดยทั่วไปแล้วจะมีทางเข้าได้ทางเดียว และมีทางออกได้หลายทาง

2.3.5 **จุดร่วม** : เป็นจุดที่การควบคุมการทำงานรวมเข้าหากัน ซึ่งอาจเกิดจากการใช้คำสั่งที่ควบคุมการทำงานโดยตรง หรือจุดร่วมที่เกิดจากการตัดสินใจ เป็นต้น



รูป 2.3.2 แสดงตัวอย่างกราฟสายงาน

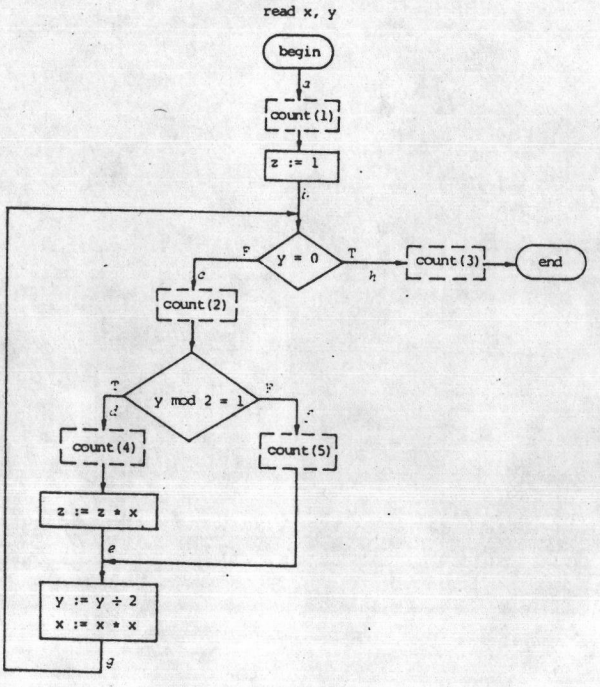
2.3.6 **เส้นทาง (Path)** [1],[2] : คือลำดับของคำสั่งที่เริ่มจาก ตำแหน่งเข้า (entry), จุดร่วม หรือจุดตัดสินใจ และจบลงที่ จุดร่วม, จุดตัดสินใจ หรือทางออก (exit) ในเส้นทางสามารถมี จุดร่วม, กระบวนการ หรือ จุดตัดสินใจ ได้หลายๆครั้ง

2.3.7 **เชื่อมโยง (link)** [1],[2] : คือกระบวนการ (process) ที่เชื่อมต่อระหว่างจุดแตกกิ่ง (node) สองจุด เช่นจุดร่วม-กระบวนการ-จุดร่วม (junction-process-junction), จุดร่วม-กระบวนการ-จุดตัดสินใจ (junction-process-decision) เป็นต้น

2.3.8 เส้นทางตัดสินใจ (Decision-to-Decision path) [1],[2] : ได้แก่ เส้นทาง ที่เริ่มต้นจากจุดตำแหน่งเข้า หรือบล็อกตัดสินใจ (decision block) และจบลงที่จุดทางออก หรือบล็อกตัดสินใจ และไม่มีบล็อกตัดสินใจอยู่ภายในเส้นทางนั้น

2.3.9 การตรวจสอบเส้นทาง (Path Instrumentation) [1],[2] : เป็นการตรวจสอบการทำงานของคำสั่ง หรือเส้นทาง ที่เราต้องการทราบว่าได้ถูกประมวลผลแล้วหรือไม่ เช่น เราอาจจะแทรกคำสั่งที่ให้พิมพ์ข้อความใดข้อความหนึ่งโดยแทรกไว้ยังหลังคำสั่งที่เราต้องการ แล้วตรวจสอบดูผลลัพธ์ที่ได้ว่ามีข้อความที่เราแทรกไว้หรือไม่ โดยทั่วไปลักษณะการแทรกตัวตรวจสอบนี้ จะแทรกไว้หลังคำสั่งที่เป็นตัวควบคุมสายงาน (control flow) เช่นคำสั่งเงื่อนไขเป็นต้น

2.3.10 ตัวนับ (Counters) [1],[2] : เป็นเครื่องมือที่ใช้ในการตรวจสอบเส้นทางอย่างหนึ่ง โดยตัวนับจะถูกแทรกไว้ที่ต้นของเส้นทางที่เราต้องการตรวจสอบ และตัวนับจะถูกบวกทีละหนึ่งเมื่อมีการประมวลผลตัวนับ การตรวจสอบที่สำคัญของการใส่ตัวนับก็คือ การตรวจสอบค่าตัวนับว่าเป็นศูนย์ ซึ่งเป็นค่าเริ่มต้นของตัวนับหรือไม่ ถ้าเป็นศูนย์ก็แสดงว่าเส้นทางนั้นยังไม่เคยถูกสำรวจ หรือคำสั่งนั้นยังไม่เคยถูกประมวลผลเลย โดยทั่วไปตัวนับจะถูกแทรกไว้ในเส้นทางที่เป็นเส้นทางตัดสินใจ ดังตัวอย่างในรูป 2.3.3



รูปที่ 2.3.3 ตัวอย่างการแทรกตัวนับในกราฟสายงาน

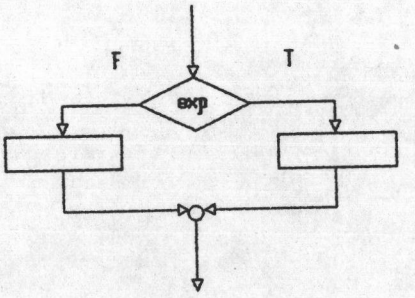
2.4 คำสั่งควบคุมสายงานของโปรแกรมภาษาซี

ในหัวข้อนี้ เราจะพิจารณาถึงคำสั่งควบคุมสายงานอย่างมีเงื่อนไขของโปรแกรมภาษาซีในแต่ละคำสั่ง และวิธีการแทรกตัวนับลงในคำสั่งควบคุมสายงานอย่างมีเงื่อนไขของโปรแกรมภาษาซีเหล่านั้น ในการแทรกตัวนับนั้นเราจะแทรกในช่วงต้นของ **เส้นทางตัดสินใจ** (Dicision-to-Dicision Path) ดังตัวอย่างที่กล่าวมาแล้วในรูปที่ 2.3.3

2.4.1 การแทรกตัวนับในคำสั่งควบคุมสายงาน if

คำสั่ง if มีรูปแบบทั่วไปของคำสั่งและกราฟสายงาน (Flowgraphs) ของคำสั่งดังรูปที่ 2.4.1.1

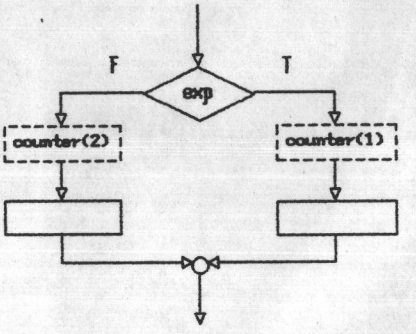
```
if (value) {
     statement1;
} else {
     statement2;
}
```



รูปที่ 2.4.1.1 แสดงรูปแบบ และกราฟสายงาน ของคำสั่ง if

ในการแทรกตัวนับลงในคำสั่ง if จะแสดงรูปแบบและกราฟสายงาน ของการทำงานได้ดังรูปที่ 2.4.1.2

```
if (value) {
   counter(1);
   statement1;
} else {
   counter(2);
   statement2;
}
```

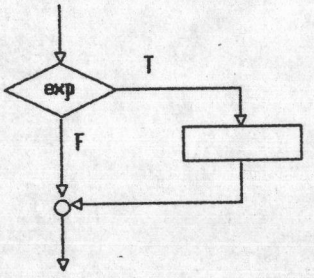


รูปที่ 2.4.1.2 แสดงรูปแบบ และกราฟสายงาน ของการแทรกตัวนับในคำสั่ง if

ความหมายสำหรับค่าของตัวนับนั้น *counter(1)* จะบอกถึงจำนวนครั้งของคำสั่ง if หรือคำสั่งใน statement1 ที่ถูกประมวลผล และ *counter(2)* จะบอกถึงจำนวนครั้งของคำสั่ง else หรือคำสั่งใน statement2 ที่ถูกประมวลผล

ในบางครั้งในส่วนของคำสั่ง if จะไม่มีส่วนของ else ดังรูปแบบและกราฟสายงานในรูปที่ 2.4.1.3

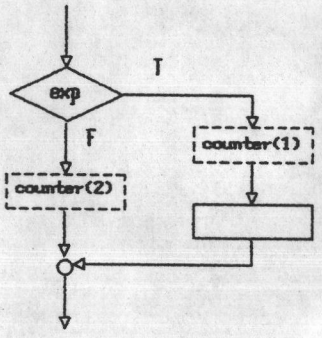
```
if (exp) {
     statement;
}
```



รูปที่ 2.4.1.3 แสดงรูปแบบ และกราฟสายงานของคำสั่ง if ที่ไม่มีส่วนของ else

ในการแทรกตัวนับลงในคำสั่ง if ที่ไม่มีส่วนของ else เราจะเพิ่มส่วนของ else เข้าไปในคำสั่ง เพื่อตรวจสอบทางเดินของโปรแกรมดังรูปที่ 2.4.1.4

```
if (exp) {
     counter(1);
     statement;
} else {
     counter(2);
}
```

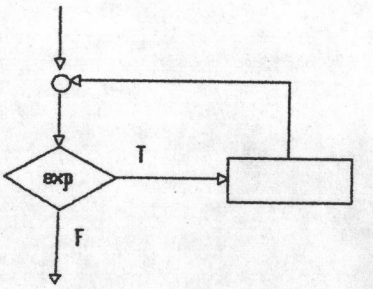


รูปที่ 2.4.1.4 แสดงรูปแบบ และกราฟสายงานของการแทรกตัวนับในคำสั่ง if ที่ไม่มีส่วนของ else

2.4.2 <u>การแทรกตัวนับลงในคำสั่งควบคุมสายงาน while</u>

คำสั่ง while มีรูปแบบทั่วไปของคำสั่งและกราฟสายงานของคำสั่งดังรูป 2.4.2.1

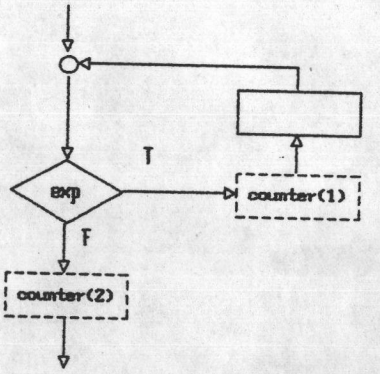
```
while (expression) {
     statement;
}
```



รูปที่ 2.4.2.1 แสดงรูปแบบ และกราฟสายงาน ของการแทรกตัวนับในคำสั่ง while

ในการแทรกตัวนับลงในคำสั่ง while จะแสดงรูปแบบ และกราฟสายงานของการทำงานได้ดังรูปที่ 2.4.2.2

```
while (expression) {
     counter(1);
     statement;
}
counter(2);
```



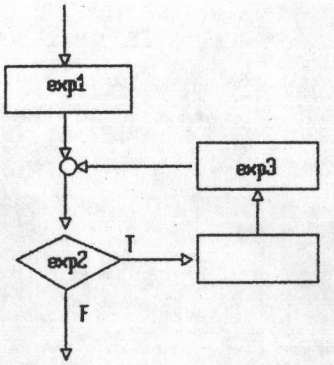
รูปที่ 2.4.2.2 แสดงรูปแบบ และกราฟสายงาน ของการแทรกตัวนับในคำสั่ง while

ความหมายสำหรับค่าของตัวนับนั้น *counter(1)* จะบอกถึงจำนวนครั้งของคำสั่ง while หรือ statement ในคำสั่ง while ที่ถูกประมวลผล และ *counter(2)* บอกถึงจำนวนครั้งในการประมวลผลของคำสั่งที่อยู่หลังคำสั่ง while

2.4.3 <u>การแทรกตัวนับลงในคำสั่งควบคุมสายงาน for</u>

คำสั่ง for มีรูปแบบทั่วไปของคำสั่ง และกราฟสายงานของคำสั่งดังรูปที่ 2.4.3.1

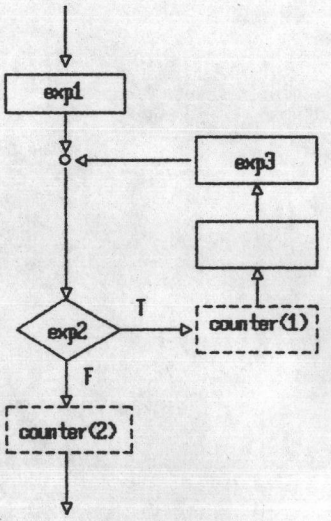
```
for ( exp1; exp2; exp3 ) {
      statement;
}
```



รูปที่ 2.4.3.1 แสดงรูปแบบ และกราฟสายงาน ของการแทรกตัวนับในคำสั่ง for

ในการแทรกตัวนับลงในคำสั่ง for จะแสดงรูปแบบ และกราฟสายงาน ของการทำงานได้ดังรูปที่ 2.4.3.2

```
for (exp1; exp2; exp3) {
      counter(1);
      statement;
}
counter(2);
```



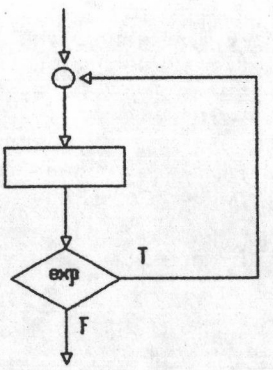
รูปที่ 2.4.3.2 แสดงรูปแบบ และกราฟสายงาน ของการแทรกตัวนับในคำสั่ง for

ความหมายสำหรับค่าของตัวนับนั้น *counter(1)* จะบอกถึงจำนวนครั้งของคำสั่ง for หรือ statement ในคำสั่ง for ที่ถูกประมวลผล และ *counter(2)* จะบอกถึงจำนวนครั้งของคำสั่งในการประมวลผลของคำสั่งที่อยู่หลังคำสั่ง for

2.4.4 <u>การแทรกตัวนับลงในคำสั่งควบคุมสายงาน do... while</u>

คำสั่ง do... while มีรูปแบบทั่วไปของคำสั่ง และกราฟสายงานของคำสั่งดังรูปที่ 2.4.4.1

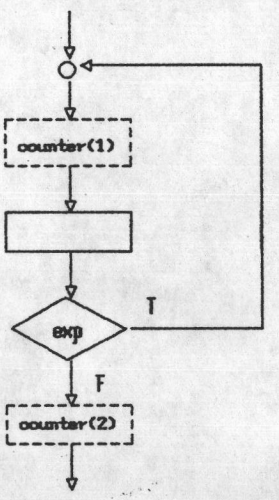
```
do {
     statement;
} while ( exp );
```



รูปที่ 2.4.4.1 แสดงรูปแบบและกราฟสายงานของการแทรกตัวนับลงในคำสั่ง do...while

ในการแทรกตัวนับลงในคำสั่ง do... while จะแสดงรูปแบบ และกราฟสายงานของการทำงานได้ดังรูปที่ 2.4.4.2

```
do {
     counter(1);
     statement;
} while ( exp );
counter(2);
```



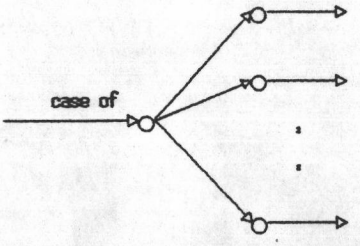
รูปที่ 2.4.4.2 แสดงรูปแบบ และกราฟสายงาน ของการแทรกตัวนับลงในคำสั่ง do...while

ความหมายสำหรับค่าของตัวนับนั้น *counter(1)* จะบอกถึงจำนวนครั้งของคำสั่ง do... while หรือ statement ในคำสั่ง do... while ที่ถูกประมวลผล และ *counter(2)* จะบอกถึงจำนวนครั้งของคำสั่งในการประมวลผลของคำสั่งที่อยู่หลังคำสั่ง do... while

2.4.5 <u>การแทรกตัวนับลงในคำสั่งควบคุมสายงาน switch</u>

คำสั่ง switch มีรูปแบบทั่วไปของคำสั่งและกราฟสายงานของคำสั่งดังรูปที่ 2.4.5.1

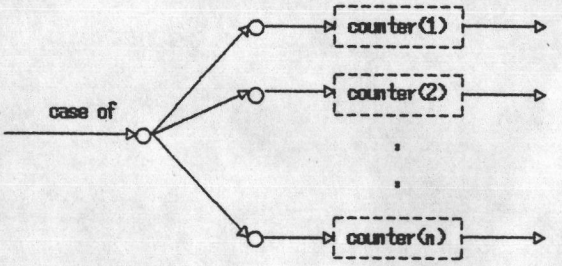
```
switch (value) {
     case value1:
          statement1;
     case value2:
          statement2;
          ...
     default :
          statement;
}
```



รูปที่ 2.4.5.1 แสดงรูปแบบ และกราฟสายงาน ของการแทรกตัวนับลงในคำสั่ง switch

ในการแทรกตัวนับลงในคำสั่ง switch นั้นเราจะแทรกตัวนับลงในหลังคำสั่ง case และหลังคำสั่ง default ในกรณีที่ switch ไม่มีคำสั่ง defualt เราจะทำการแทรกคำสั่ง default เข้าไปพร้อมกับตัวนับต่อท้ายคำสั่ง default ในการแทรกตัวนับนั้นเราจะแสดงดังนี้

```
switch (value) {
     case value1:
          counter(1);
          statement1;
     case value2:
          counter(2);
          statement2;
          ...
     default :
          counter(n);
          statement;
}
```



รูปที่ 2.4.5.2 แสดงรูปแบบและกราฟสายงานของการแทรกตัวนับลงในในคำสั่ง switch

ความหมายสำหรับค่าของตัวนับ *counter()* แต่ละนั้นจะบอกถึงจำนวนครั้งของการประมวลผลในแต่ละเงื่อนไขของ case หรือ default ในคำสั่ง switch นั้น