การพัฒนาเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบสำหรับภาษาซี

นางสาว ทิพยา ทิพยโสต

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยเทคโนโลยีสุรนารี

ปีการศึกษา 2549

DEVELOPMENT OF CONTROL FLOW GRAPH AND TEST DATA TOOL FOR C LANGUAGE

Miss Tippaya Tippayasot

A Thesis Submitted in Partial Fulfillment of the Requirments for the

Degree of Master of Engineering in Computer Engineering

Suranaree University of Technology

Academic Year 2006

การพัฒนาเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบสำหรับภาษาซี

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

_____

(รศ. ดร.กิตติศักดิ์ เกิดประสพ)
ประธานกรรมการ

_____

(ผศ. ดร.พิชโยทัย มหัทธนาภิวัฒน์)
กรรมการ (อาจารย์ที่ปรึกษาวิทยานิพนธ์)

_____

(รศ. ดร.นิตยา เกิดประสพ)
กรรมการ

_____

(ผศ. ดร.คะชา ชาญศิลป์)
กรรมการ

_____

(รศ. ดร.เสาวณีย์ รัตนพานี)
รองอธิการบดีฝ่ายวิชาการ

_____

(รศ. น.อ. ดร.วรพจน์ ขำพิศ)
คณบดีสำนักวิชาวิศวกรรมศาสตร์

ทิพยา ทิพยโสต : การพัฒนาเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบสำหรับภาษาซี (DEVELOPMENT OF CONTROL FLOW GRAPH AND TEST DATA TOOL FOR C LANGUAGE) อาจารย์ที่ปรึกษา : ผศ. ดร.พิชโยทัย มหัทธนาภิวัฒน์, 99 หน้า.

ในการพัฒนาซอฟต์แวร์ จำเป็นต้องมีการทดสอบเพื่อหาข้อผิดพลาด และตรวจสอบเพื่อให้ถูกต้องตามความต้องการ โดยซอร์สโค้ดที่พัฒนาจากภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่งจำเป็นต้องได้รับการทดสอบที่อาศัยเทคนิคทดสอบแบบไวท์บอกซ์ เพื่อให้มั่นใจว่าโปรแกรมมีความผิดพลาดน้อยที่สุดและทำงานได้ตรงตามการออกแบบ ซึ่งจะทำการทดสอบในระยะของการทดสอบระดับหน่วยที่เป็นการทดสอบการทำงานของแต่ละหน่วย แต่ละฟังก์ชัน หรือแต่ละโปรซีเจอร์ การทดสอบในระดับนี้จะมีประสิทธิภาพได้นั้น ผู้ทดสอบจะต้องรู้ผังการทำงานของซอร์สโค้ด โดยกราฟควบคุมกระแสเป็นเครื่องมือที่แสดงผังควบคุมการทำงานและเส้นทางสำหรับทดสอบได้ชัดเจน ซึ่งจะสามารถนำไปสร้างเป็นข้อมูลทดสอบให้กับเส้นทางเหล่านั้นได้อย่างครอบคลุม และนำไปสู่การทดสอบที่มีประสิทธิภาพต่อไป งานวิจัยนี้จะศึกษาเพื่อพัฒนาเครื่องมือสร้างกราฟควบคุมกระแสที่สามารถแสดงผังควบคุมการทำงานของซอร์สโค้ดภาษาซี โดยจะนำเอากราฟที่ได้นั้นมาสร้างเส้นทางสำหรับทดสอบซอฟต์แวร์ให้ครอบคลุมทุกเส้นทางที่เป็นไปได้ และจะสร้างข้อมูลทดสอบเพื่อใช้สำหรับทดสอบตามเส้นทางทดสอบที่สร้างขึ้นให้ครบทุกเส้นทาง โดยซอร์สโค้ดที่จะนำเข้าสู่เครื่องมือนี้ไม่จำกัดจำนวนบรรทัด แต่จะมีข้อกำหนดคือ ซอร์สโค้ดที่นำเข้าสู่เครื่องมือนี้จะต้องมีโครงสร้างควบคุมซ้อนกันไม่เกิน 10 ชั้น จะต้องเขียนด้วยรูปแบบมาตรฐานภาษาซีเท่านั้น นอกจากนี้จะต้องผ่านการคอมไพล์ หรือตรวจสอบหลักการเขียนและแก้ไขให้ถูกต้องก่อน เพราะถ้าไม่เป็นไปตามข้อกำหนดนี้ กราฟควบคุมกระแส เส้นทางสำหรับทดสอบ และข้อมูลทดสอบที่ได้อาจจะมีความผิดพลาดได้ แนวคิดในการสร้างเครื่องมือเพื่อทดสอบโปรแกรมที่พัฒนาด้วยภาษาซีนี้ ได้แบ่งองค์ประกอบของเครื่องมือเป็น 5 ส่วนด้วยกันคือ ส่วนนำเข้าข้อมูลและติดต่อผู้ใช้งาน ส่วนกระจายนิพจน์ ส่วนสร้างกราฟควบคุมกระแส ส่วนสร้างเส้นทางสำหรับทดสอบ และส่วนสุดท้ายคือ ส่วนสร้างข้อมูลทดสอบ

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา_____

ปีการศึกษา 2549 ลายมือชื่ออาจารย์ที่ปรึกษา_____

ลายมือชื่ออาจารย์ที่ปรึกษาร่วม_____

TIPPAYA TIPPAYASOT : DEVELOPMENT OF CONTROL FLOW GRAPH AND TEST DATA TOOL FOR C LANGUAGE. THESIS ADVISOR : ASST. PROF. PICHAYOTAI MAHATTHANAPIWAT, Ph. D., 99 PP.

WHITEBOX TESTING/CONTROL FLOW GRAPH/TEST PATH/TEST DATA

In software development process, validation and verification is necessary for software testing, i.e. how software performs according to the right specification and how the right process is performed to develop the software. The more complex system, there are difficult for validation and verification. In unit testing phase, source code of the program in the module, for example, function or procedure, will be exercised if every path in the source code is traversed. Control flow graph generated from source code and test data will be used to visualize the flow of a program to check each path in the flow graph is exercised. Therefore, the tool to generate control flow graph and test data will be useful for the tester in unit testing phase. This tool is not limited lines in source code, but source code is coded only standard C language and not more than 10 layers. Anyhow, source code to be necessary to compiling, if it is not, it will have effect to path and data testing. There are 5 sections of this tools that have been developed by using C language, first is import and user interface, second is parser, third is control flow graph, fourth is testing path and last is data testing.

School of Computer Engineering Student's Signature_____

Academic Year 2006 Advisor's

Signature_____

Co-advisor’s

Signature_____

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี เนื่องจากผู้วิจัยได้รับการสนับสนุนในหลายด้าน จากบุคคลหลายท่าน ซึ่งผู้วิจัยขอกราบขอบพระคุณดังนี้

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร. พิชโยทัย มหัทธนาภิวัฒน์, อาจารย์ที่ปรึกษาวิทยานิพนธ์

ขอขอบพระคุณ อาจารย์ เจษฎา รัตนสุพร ที่ให้คำปรึกษาและคอยให้คำแนะนำ

ขอขอบพระคุณ มหาวิทยาลัยราชภัฏนครราชสีมา ที่ให้การสนับสนุนทุนการศึกษา

ขอบคุณ เพื่อนที่แสนดี ที่ให้แนวทางและคำปรึกษาด้านเทคนิค รวมถึงคอยให้กำลังใจในการทำงาน

ท้ายนี้ ขอกราบขอบพระคุณบิดา มารดา ที่คอยให้ความรักความห่วงใย ให้กำลังใจ และคอยกระตุ้นเตือน สร้างแรงใจในการทำงานมาโดยตลอด จนทำให้งานวิจัยนี้สำเร็จลุล่วงด้วยดี

ทิพยา ทิพยโสต

สารบัญ

หน้า

บทคัดย่อ (ภาษาไทย) ..... ก

บทคัดย่อ (ภาษาอังกฤษ) ..... ข

กิตติกรรมประกาศ ..... ค

สารบัญ ..... ง

สารบัญตาราง ..... ซ

สารบัญรูป ..... ฌ

บทที่

1 บทนำ 1

1.1 ความสำคัญและที่มาของปัญหาการวิจัย 1

1.2 วัตถุประสงค์การวิจัย 2

1.3 ข้อตกลงเบื้องต้น 3

1.4 ขอบเขตของการวิจัย 3

1.5 ประโยชน์ที่คาดว่าจะได้รับ 3

2 ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง 5

2.1 งานวิจัยที่เกี่ยวข้อง 5

2.2 การตรวจสอบและการยืนยันความถูกต้อง 7

2.2.1 เทคนิคแบบ Static 8

2.2.2 เทคนิคแบบ Dynamic 8

2.3 การทดสอบโปรแกรม 8

2.3.1 การทดสอบทางสถิติ 8

2.3.2 การทดสอบหาข้อบกพร่อง 9

2.4 ระยะของการทดสอบโปรแกรม 10

2.4.1 การทดสอบระดับหน่วยหรือทดสอบส่วนย่อย 10

2.4.2 การทดสอบระดับโมดูล 10

สารบัญ (ต่อ)

หน้า

2.4.3 การทดสอบระบบย่อย 10
2.4.4 การทดสอบระบบ 10
2.4.5 การทดสอบการยอมรับของผู้ใช้ระบบ 10
2.5 กลยุทธ์ในการทดสอบโปรแกรม 10
2.5.1 การทดสอบการรวมระบบจากบนลงล่าง 11
2.5.2 การทดสอบการรวมระบบจากล่างขึ้นบน 12
2.5.3 การทดสอบแบบแบล็กบอกซ์ 13
2.5.4 การทดสอบแบบไวท์บอกซ์ 16
2.5.5 การทดสอบความเกี่ยวเนื่องกัน 18
2.5.6 การทดสอบการเชื่อมต่อของระบบ 18
2.5.7 การทดสอบความคงทนของระบบ 19
2.5.8 การทดสอบที่เสริมกัน 20
2.5.9 การทดสอบแบบฝังคำสั่ง 20
2.5.10 การทดสอบโครงแบบ 20
2.5.11 การทดสอบการกู้ระบบ 20
2.5.12 การทดสอบประสิทธิภาพ 20
2.5.13 การทดสอบความปลอดภัย 20
2.6 การเขียนโปรแกรมด้วยภาษาซีเบื้องต้น 21
2.6.1 ชนิดข้อมูลและค่าคงที่ 21
2.6.2 ตัวดำเนินการ 23
2.6.3 คำสั่งควบคุม 25
3 การออกแบบและพัฒนาเครื่องมือสร้างกราฟควบคุมกระแส และข้อมูลทดสอบสำหรับภาษาซี 37
3.1 องค์ประกอบหลักของเครื่องมือ 37
3.1.1 ส่วนของการพัฒนาเครื่องมือ 37
3.1.2 ส่วนของการจัดเก็บข้อมูล 38

สารบัญ (ต่อ)

หน้า

3.2 รายละเอียดของขั้นตอนการทำงานของเครื่องมือ 39
3.2.1 ส่วนติดต่อกับผู้ใช้ 40
3.2.2 ส่วนกระจายนิพจน์ 42
3.2.3 ส่วนสร้างกราฟควบคุมกระแส 49
3.2.4 ส่วนสร้างเส้นทางสำหรับทดสอบ 51
3.2.5 ส่วนสร้างข้อมูลทดสอบ 54
3.3 โครงสร้างของเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบ 60
3.3.1 เมนูคำสั่ง 60
3.3.2 การแสดงผล 61
3.4 รายละเอียดการออกแบบส่วนการจัดเก็บข้อมูลพื้นฐานภาษาซี 63
4 ผลการดำเนินการวิจัยและการอภิปรายผล 67
4.1 ส่วนนำเข้าซอร์สโค้ดและใช้ติดต่อกับผู้ใช้ 67
4.2 ส่วนกระจายนิพจน์ 72
4.2.1 กรองฟังก์ชัน 73
4.2.2 แยกโหนด 75
4.2.3 แยกโทเคน 76
4.3 ส่วนสร้างกราฟควบคุมกระแส 77
4.3.1 คำสั่งควบคุมแบบเลือก 77
4.3.2 คำสั่งควบคุมแบบวนซ้ำ 81
4.4 ส่วนสร้างเส้นทางทดสอบ 84
4.5 ส่วนสร้างข้อมูลทดสอบ 86
5 สรุปผลการวิจัยและข้อเสนอแนะ 89
5.1 สรุปผลการวิจัย 89
5.1.1 สรุปผลการทำงานส่วนนำเข้าซอร์สโค้ดและติดต่อผู้ใช้งาน 89
5.1.2 สรุปผลการทำงานส่วนกระจายนิพจน์ 90
5.1.3 สรุปผลการทำงานส่วนสร้างกราฟควบคุมกระแส 90

สารบัญ (ต่อ)

หน้า

5.1.4 สรุปผลการทำงานส่วนสร้างเส้นทางทดสอบ 91
5.1.5 สรุปผลการทำงานส่วนสร้างข้อมูลทดสอบ 91
5.2 การประยุกต์ผลการวิจัย 91
5.3 ข้อเสนอแนะในการวิจัยต่อไป 92
รายการอ้างอิง 93
ภาคผนวก 95
ประวัติผู้เขียน 99

สารบัญตาราง

ตารางที่ **หน้า**

2.1 แสดงตัวอย่างการกำหนดข้อมูลให้กับอาร์เรย์และผลลัพธ์ที่จะเกิดขึ้นในการค้นหาข้อมูลที่อยู่ในอาร์เรย์ 15
2.2 แสดงตัวอย่างการกำหนดข้อมูล และผลลัพธ์ที่จะได้ในการค้นหาข้อมูล 15
2.3 แสดงชนิดข้อมูลพื้นฐาน 21
2.4 แสดงตัวขยายของชนิดข้อมูล 22
2.5 แสดงตัวดำเนินการเลขคณิต 24
3.1 แสดงโครงสร้างของตารางกำหนดค่าตั้งต้นระบบ 63
3.2 แสดงโครงสร้างของตาราง Token 63
3.3 แสดงโครงสร้างของข้อกำหนดของคำสั่งควบคุม 65
3.4 แสดงโครงสร้างของตารางข้อกำหนดชนิดข้อมูล 65
3.5 แสดงโครงสร้างของตารางตัวดำเนินการ 65

สารบัญรูป

รูปที่ **หน้า**

2.1 แสดงการใช้งาน Static verification และ Dynamic verification ในส่วนต่าง ๆ 8
2.2 แสดงกระบวนการแก้ไขข้อผิดพลาด 9
2.3 แสดงกระบวนการทดสอบโปรแกรม 9
2.4 แสดงการทดสอบโดยการรวมโมดูลเพิ่มในระบบทีละโมดูล 11
2.5 แสดงลักษณะการทดสอบโปรแกรมแบบบนลงล่าง 12
2.6 แสดงลักษณะการทดสอบโปรแกรมแบบล่างขึ้นบน 12
2.7 แสดงแบบจำลองการทดสอบแบบแบล็กบอกซ์ 13
2.8 แสดงการแบ่งช่วงข้อมูลเข้า ที่ปกติและที่ทำให้เกิดข้อผิดพลาด 13
2.9 แสดงการแบ่งช่วงข้อมูลที่กำหนดไว้และข้อมูลเข้าที่เป็นไปได้และใช้สำหรับการทดสอบซอร์สโค้ด 14
2.10 แสดงตัวอย่างโปรแกรมค้นหาข้อมูลเขียนด้วยภาษาจาวา 17
2.11 แสดงตัวอย่างกราฟควบคุมกระแส และเส้นทางทดสอบที่สร้างจากรูป 2.10 17
2.12 แสดงกระบวนการที่รับส่งข้อมูลระหว่างกัน 18
2.13 แสดงภาพจำลองการติดต่อกันระหว่างระบบย่อยในระบบ 19
2.14 แสดงโครงสร้างการดำเนินการของคำสั่งดำเนินการแบบหนึ่งทางเลือก 26
2.15 แสดงตัวอย่างคำสั่งดำเนินการแบบหนึ่งทางเลือก 26
2.16 แสดงโครงสร้างการดำเนินการของคำสั่งดำเนินการแบบสองทางเลือก 27
2.17 แสดงตัวอย่างคำสั่งดำเนินการแบบสองทางเลือก 28
2.18 แสดงโครงสร้างการดำเนินการของคำสั่งดำเนินการแบบหลายทางเลือก 29
2.19 แสดงตัวอย่างคำสั่งดำเนินการแบบหลายทางเลือก 30
2.20 แสดงตัวอย่างคำสั่งดำเนินการแบบหลายทางเลือกด้วย case statement 31
2.21 แสดงโครงสร้างการดำเนินการของคำสั่งวนซ้ำแบบ while 32
2.22 แสดงตัวอย่างการใช้คำสั่งวนซ้ำแบบ while 33
2.23 แสดงโครงสร้างการดำเนินการของคำสั่งวนซ้ำแบบ do-while 34

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

2.24 แสดงตัวอย่างการใช้คำสั่งวนซ้ำแบบ do-while....34
2.25 แสดงโครงสร้างการดำเนินการของคำสั่งวนซ้ำแบบ for....35
2.26 แสดงตัวอย่างการใช้คำสั่งวนซ้ำแบบ for....36
3.1 แสดงองค์ประกอบของการพัฒนาเครื่องมือสร้างกราฟควบคุมกระและข้อมูลทดสอบ....38
3.2 แสดงแผนภาพกระแสข้อมูลระดับ 0 ของเครื่องมือ....39
3.3 แสดงแผนภาพกระแสข้อมูลระดับ 1 ของส่วนติดต่อผู้ใช้....40
3.4 แสดงแผนภาพกระแสข้อมูลระดับ 1 ของส่วนกระจายนิพจน์....42
3.5 แสดงตัวอย่างซอร์สโค้ดและการกรองฟังก์ชัน....44
3.6 แสดงตัวอย่างซอร์สโค้ดและการแยกโหนด....45
3.7 แสดงการเก็บข้อมูลแต่ละโหนดและความเกี่ยวเนื่องกันของแต่ละโหนด....47
3.8 แสดงแผนภาพกระแสข้อมูลระดับ 1 ของส่วนสร้างกราฟควบคุมกระแส....49
3.9 แสดงการวาดโหนดโดยใช้ข้อมูลจากส่วนแยกโหนด....50
3.10 แสดงการวาดเส้นเชื่องระหว่างโหนดโดยใช้ข้อมูลจากส่วนแยกโหนด....51
3.11 แสดงแผนภาพกระแสข้อมูลระดับ 1 ของส่วนสร้างเส้นทางทดสอบ....52
3.12 แสดงเส้นทางทดสอบที่สร้างจากเครื่องมือ....54
3.13 แสดงแผนภาพกระแสข้อมูลระดับ 1 ของส่วนสร้างข้อมูลทดสอบ....55
3.14 แสดงกราฟควบคุมกระแสที่สร้างจากเครื่องมือ....58
3.15 แสดงเส้นทางทดสอบและข้อมูลทดสอบของแต่ละเส้นทาง....58
3.16 แสดงโครงสร้างการประมวลผลของเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบสำหรับภาษาซี....59
3.17 แสดงโครงสร้างส่วนประกอบของเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบสำหรับภาษาซี....60
3.18 แสดงตัวอย่างการแสดงผลฟังก์ชัน กราฟควบคุมกระแส เส้นทางทดสอบและข้อมูลทดสอบของแต่ละฟังก์ชัน....62
3.19 แผนภาพแสดงเอนทิตี้และความสัมพันธ์ของตารางในฐานข้อมูล....66

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

4.1 แสดงผลการทำงานเมื่อมีการนำเข้าซอร์สโค้ดภาษาซี 68
4.2 แสดงผลการทำงานเมื่อเครื่องมือแยกฟังก์ชัน 69
4.3 แสดงผลการสร้างกราฟและเส้นทางทดสอบของฟังก์ชันที่ 1 จากรูปที่ 4.2 69
4.4 แสดงผลการสร้างกราฟเส้นทางทดสอบของฟังก์ชันที่ 2 จากรูปที่ 4.2 70
4.5 แสดงผลการสร้างกราฟเส้นทางทดสอบของฟังก์ชันที่ 3 จากรูปที่ 4.2 70
4.6 แสดงข้อมูลทดสอบสำหรับเส้นทางทดสอบ จากรูปที่ 4.3 71
4.7 แสดงข้อมูลทดสอบสำหรับเส้นทางทดสอบ จากรูปที่ 4.4 71
4.8 แสดงข้อมูลทดสอบสำหรับเส้นทางทดสอบ จากรูปที่ 4.5 71
4.9 แสดงตัวอย่างซอร์สโค้ดที่นำเข้าสู่เครื่องมือ 72
4.10 แสดงซอร์สโค้ดจากรูปที่ 4.1 ที่ถูกแยกเป็นฟังก์ชันย่อยเพื่อแสดงในแต่ละหน้า 73
4.11 แสดงข้อมูลซอร์สโค้ดที่ถูกเก็บไว้หลังกรองฟังก์ชัน 74
4.12 แสดงข้อมูลโหนดแต่ละโหนดที่เก็บไว้หลังการแยกโหนด 75
4.13 แสดงตัวอย่างข้อมูลโทเคนแต่ละตัวที่จัดเก็บไว้หลังผ่านการแยกโหนด 76
4.14 แสดงตัวอย่างข้อมูลโทเคนแต่ละตัวที่จัดเก็บไว้หลังผ่านการแยกโหนด 76
4.15 แสดงตัวอย่างข้อมูลโทเคนแต่ละตัวที่จัดเก็บไว้หลังผ่านการแยกโหนด 76
4.16 แสดงซอร์สโค้ดที่มีคำสั่งควบคุมแบบหนึ่งทางเลือก และกราฟควบคุมกระแส 77
4.17 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมแบบสองทางเลือก 78
4.18 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมแบบหลายทางเลือก 79
4.19 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมแบบหลายทางเลือก case statement 80
4.20 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมวนซ้ำแบบ while 81
4.21 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมวนซ้ำแบบ do-while 82

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

4.22 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมวนซ้ำแบบ for 82
4.23 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมแบบผสม.....83
4.24 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 1.....84
4.25 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 2.....84
4.26 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 3.....85
4.27 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 4.....85
4.28 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 5.....86
4.29 แสดงตัวอย่างการสร้างข้อมูลทดสอบแบบที่ 1 86
4.30 แสดงตัวอย่างการสร้างข้อมูลทดสอบแบบที่ 2 87
4.31 แสดงตัวอย่างการสร้างข้อมูลทดสอบแบบที่ 3 87

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

ในยุคของข้อมูลข่าวสารอย่างปัจจุบัน ในประเทศที่พัฒนาแล้วทั้งหลาย วิศวกรรมซอฟต์แวร์ (software engineering) มีบทบาทสำคัญมากในด้านเศรษฐศาสตร์ เนื่องจากในองค์ประกอบหนึ่งของวิศวกรรมซอฟต์แวร์ มีซอฟต์แวร์ (software) ซึ่งจัดเป็นสินค้าที่นำรายได้มูลค่าสูงมาให้ นอกจากนี้ซอฟต์แวร์ยังมีส่วนช่วยงานในกิจกรรมต่าง ๆ มีบทบาทให้ธุรกิจดำเนินไปได้อย่างมีประสิทธิภาพ เกือบทุกหน่วยงานล้วนต้องนำซอฟต์แวร์มาใช้แทบทั้งสิ้น ไม่ว่าจะเป็นงานควบคุม งานวิเคราะห์ งานจัดเก็บข้อมูล หรืองานสื่อสารข้อมูลเป็นต้น ซึ่งส่งผลให้หลายหน่วยงานทั้งภาครัฐ และเอกชน รวมถึงผู้ประกอบการค้าซอฟต์แวร์ ให้ความสนใจที่จะพัฒนาซอฟต์แวร์ให้มีคุณภาพมากขึ้น ความรู้หรือทฤษฎีที่เกี่ยวข้องกับกระบวนการพัฒนาซอฟต์แวร์ (software development process) จึงเป็นเรื่องที่มีความจำเป็นและสำคัญ โดยกระบวนการพัฒนาซอฟต์แวร์นี้เองเป็นแนวทางที่มุ่งเน้นให้ได้ซอฟต์แวร์ที่มีคุณภาพมีความถูกต้อง และตรงกับความต้องการของผู้ใช้งาน และยังอยู่ภายใต้งบประมาณที่กำหนดให้มากที่สุด

ในกระบวนการการพัฒนาซอฟต์แวร์ มีองค์ประกอบหลายส่วนด้วยกัน และ องค์ประกอบหนึ่งที่ทำให้ซอฟต์แวร์ที่ได้ มีความถูกต้องและมีคุณภาพยิ่งขึ้น คือ การทดสอบ (validation) เพื่อหาข้อผิดพลาดและการตรวจสอบ (verification) เพื่อยืนยันความถูกต้องของซอฟต์แวร์ ดังที่กล่าวไว้ในตำราของ John Watkins (2001) ซึ่งซอร์สโค้ด (source code) ที่พัฒนาจากภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่งจำเป็นต้องได้รับการทดสอบ เพื่อให้มั่นใจว่าโปรแกรมที่ได้ผิดพลาดน้อยที่สุดและทำงานได้ตรงตามข้อกำหนดที่ตั้งไว้ ในการทดสอบซอฟต์แวร์แบ่งระยะของการทดสอบได้ดังนี้ ระยะการทดสอบระดับหน่วย (unit testing) ระยะการทดสอบการรวมกัน (integration testing) ของแต่ละหน่วยแต่ละโมดูล (module) หรือแต่ละระบบย่อย (sub-system testing) และระยะของการทดสอบการยอมรับระบบ (acceptance testing) ในระยะการทดสอบระดับหน่วยเป็นระยะที่ทดสอบการทำงานของแต่ละหน่วย (unit) ฟังก์ชัน (function) หรือโปรซีเจอร์ (procedure)ว่าทำงานได้ถูกต้องโดยให้มีข้อผิดพลาดน้อยที่สุด ในการทดสอบระดับหน่วยนี้ต้องอาศัยเทคนิคการทดสอบแบบไวท์บอกซ์ (white-box testing) ซึ่งเป็นการทดสอบที่ผู้ทำการทดสอบต้องรู้โครงสร้างของซอร์สโค้ด โดยกราฟควบคุมกระแส (control flow graph) ถือเป็นเครื่องมือที่แสดงผังการทำงาน

และเห็นเส้นทางการทำงานของซอร์สโค้ด ซึ่งจะทำให้สามารถนำไปสร้างเป็นข้อมูลทดสอบ (test data) ให้กับแต่ละเส้นทางได้ ถ้ามีการทดสอบตามเส้นทางที่ได้จนครบจะทำให้เรามั่นใจได้ว่าในซอฟต์แวร์ที่พัฒนานั้นได้ทดสอบทุกเส้นทางแล้วอย่างน้อย 1 ครั้ง ซึ่งจะช่วยสร้างความน่าเชื่อถือให้กับตัวซอฟต์แวร์เพิ่มขึ้น ฉะนั้นเครื่องมือที่มีความสามารถสร้างกราฟควบคุมกระแสจากซอร์สโค้ด แสดงเส้นทางการทดสอบซอฟต์แวร์และสร้างเป็นข้อมูลทดสอบได้ จึงเป็นเครื่องมือที่มีประโยชน์สำหรับกระบวนการทดสอบซอฟต์แวร์อย่างยิ่ง

อนึ่ง ภาษาซีเป็นภาษาที่ถูกนำมาใช้พัฒนาในหลาย ๆ ด้าน เป็นพื้นฐานให้กับภาษาอื่น ๆ หลายภาษา เนื่องจากภาษาซีมีลักษณะ เป็นภาษาที่ไม่เข้มงวดกับรูปแบบการใช้ตัวแปรและคำสั่ง ซึ่งเป็นประโยชน์สำหรับผู้เขียนโปรแกรมที่ต้องการเขียนโปรแกรมให้สั้นลงแต่ควรต้องระมัดระวังความผิดพลาดที่อาจเกิดขึ้น ภาษาซีสามารถจัดการกับข้อมูลได้ถึงระดับบิตและไบต์ สามารถทำงานในระดับฮาร์ดแวร์ได้อย่างมีประสิทธิภาพ นอกจากนี้ภาษาซียังมีตัวแปรประเภทพอยต์เตอร์(pointer) ที่สามารถจัดการกับหน่วยความจำของคอมพิวเตอร์ได้โดยตรงเป็นต้น (ซิลด์, เฮอร์เบิร์ต, ศิวัฒน์ ศิวะบวร, พรชัย จักรธำรง, จิรศักดิ์ ชัยวิริยะกุล, 2536) ภาษาซีจึงเป็นภาษาที่ควรมีเครื่องมือสำหรับช่วยในการทดสอบโปรแกรมที่เขียนขึ้น เพื่อให้การพัฒนาสะดวกและรวดเร็ว

ดังนั้นงานวิจัยนี้มุ่งเน้นเพื่อศึกษาและพัฒนาเครื่องมือสร้างกราฟควบคุมกระแสที่สามารถแสดงผังควบคุมการทำงานของซอร์สโค้ดภาษาซี โดยจะนำเอากราฟที่ได้นั้นมาสร้างเส้นทางสำหรับทดสอบซอฟต์แวร์ให้ครอบคลุมทุกเส้นทางที่เป็นไปได้และสร้างข้อมูลทดสอบเพื่อใช้สำหรับทดสอบตามเส้นทางทดสอบที่สร้างขึ้นให้ครบทุกเส้นทาง เพื่อสร้างความมั่นใจในตัวซอร์สโค้ดว่าได้ผ่านการทดสอบทุกเส้นทางแล้วอย่างน้อย 1 ครั้ง โดยซอร์สโค้ดที่จะนำเข้าสู่เครื่องมือนี้เป็นซอร์สโค้ดที่พัฒนาแค่ในระดับพื้นฐานเท่านั้น ยังไม่สนับสนุนซอร์สโค้ดที่พัฒนาแบบขั้นสูง นอกจากนี้ซอร์สโค้ดนั้นจะต้องผ่านการคอมไพล์ หรือตรวจสอบหลักการเขียนและแก้ไขให้ถูกต้องก่อน เพราะถ้าไม่เป็นไปตามข้อกำหนดนี้กราฟควบคุมกระแส เส้นทางสำหรับทดสอบ และข้อมูลทดสอบที่ได้อาจจะมีความผิดพลาด หรือไม่ครอบคลุมได้

1.2 วัตถุประสงค์การวิจัย

1.2.1 เพื่อพัฒนากระบวนการทดสอบซอฟต์แวร์ในระยะการทดสอบระดับหน่วย โดยอาศัยกราฟ ควบคุมกระแส เส้นทางทดสอบ ที่เป็นเส้นทางอิสระของโปรแกรม (cyclomatic complexity) และข้อมูลทดสอบ ให้สำหรับเส้นทางทดสอบให้ครบทุกเส้นทาง

1.2.2 เพื่อพัฒนาเครื่องมือที่สามารถช่วยในกระบวนการทดสอบซอฟต์แวร์ในระยะของการทดสอบระดับหน่วย โดยเครื่องมือสามารถสร้างกราฟควบคุมกระแส เส้นทางทดสอบ และข้อมูลทดสอบสำหรับซอร์สโค้ดที่พัฒนาด้วยโปรแกรมภาษาซี

1.2.3 เพื่อให้ผู้ทดสอบซอฟต์แวร์สามารถ ดำเนินการทดสอบระดับหน่วยด้วยเทคนิคแบบไวท์บอกซ์ ได้สะดวกรวดเร็วและมีคุณภาพยิ่งขึ้น

1.3 ข้อตกลงเบื้องต้น

1.3.1 ซอร์สโค้ด ที่นำเข้าสู่เครื่องมือนี้ต้องมีชนิด (extension) เป็น C เท่านั้น

1.3.2 ซอร์สโค้ดที่จะนำเข้าสู่เครื่องมือนี้ต้องผ่านการคอมไพล์ (compile) หรือตรวจสอบหลักการเขียน และแก้ไขให้ถูกต้องก่อน

1.3.3 เครื่องมือที่พัฒนานี้จะแสดงผลได้ถูกต้อง และแม่นยำเฉพาะกับซอร์สโค้ดที่ใช้คำสั่งหรือการใช้ตัวแปรชนิดต่าง ๆ แค่ระดับพื้นฐานเท่านั้น ไม่เช่นนั้นการแสดงผลอาจอยู่ในลักษณะหยาบ ๆ ไม่ชัดเจน ไม่ครอบคลุม หรืออาจผิดพลาดได้

1.4 ขอบเขตของการวิจัย

1.4.1 เครื่องมือที่พัฒนานี้สามารถรองรับซอร์สโค้ดได้ครั้งละ 1 ไฟล์และแต่ละไฟล์ต้องมีฟังก์ชัน (function) ภายในไฟล์ไม่เกิน 24 ฟังก์ชัน

1.4.2 เครื่องมือที่พัฒนานี้จะสร้างข้อมูลทดสอบให้เฉพาะตัวแปรชนิดข้อมูลที่เป็นพื้นฐาน คือ int, char, float และ double เท่านั้น

1.4.3 เครื่องมือที่พัฒนานี้จะสร้างข้อมูลทดสอบให้เฉพาะการตรวจสอบเงื่อนไขที่มีการใช้ เครื่องหมายคำนวณและเปรียบเทียบที่เป็นพื้นฐานเท่านั้น ไม่รวมปฏิบัติการบนบิต

1.4.4 เครื่องมือที่พัฒนานี้สามารถรองรับซอร์สโค้ดที่พัฒนาโดยใช้คำสั่งควบคุม (control statement) แบบทางเลือก (branching) แบบวนซ้ำ (looping)

1.4.5 ซอร์สโค้ดที่นำเข้าสู่เครื่องมือนี้จะต้องมีคำสั่งควบคุมทั้ง แบบทางเลือก และแบบวนซ้ำ ที่ซ้อนกันไม่เกิน 10 ชั้น ถ้ามากกว่านั้นการแสดงผลอาจจะแสดงผลได้ไม่สวยงามหรือครบถ้วนภายในความกว้างของหน้าจอ

1.4.6 งานวิจัยนี้พัฒนาด้วยโปรแกรมวิชวลเบสิก 6.0 (Microsoft Visual Basic 6.0) บนระบบปฏิบัติการ Microsoft Windows XP Professional Version 2002 Service Pack 2

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 ช่วยให้กระบวนทดสอบซอฟต์แวร์ ในระยะการทดสอบระดับหน่วยด้วยเทคนิคแบบไวท์บอกซ์สามารถทำได้สะดวกรวดเร็วและมีคุณภาพมากขึ้น

1.5.2 ได้เครื่องมือสร้างกราฟควบคุมกระแส เส้นทางทดสอบและข้อมูลทดสอบสำหรับโปรแกรมที่พัฒนาด้วยภาษาซี ช่วยลดเวลาที่ใช้ในการทดสอบให้น้อยลงและช่วยเพิ่มความมั่นใจในการทดสอบซอฟต์แวร์ยิ่งขึ้น

1.5.3 ช่วยลดภาระผู้ทดสอบ เนื่องจากเครื่องมือที่พัฒนาแสดงให้เห็นเส้นทางอิสระของโปรแกรมได้ครบทุกเส้นทางและสร้างข้อมูลทดสอบไว้รองรับแต่ละเส้นทางด้วย

บทที่ 2
ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

2.1 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่จะกล่าวถึงต่อไปนี้เป็นงานที่เกี่ยวข้องกับการพัฒนาเครื่องมือเพื่อให้สำหรับกระบวนการทดสอบซอฟต์แวร์ทั้งสิ้น โดยมีงานวิจัยหลายชิ้นด้วยกันดังนี้

งานวิจัยแรก เป็นงานวิจัยที่เกี่ยวข้องกับการพัฒนาเครื่องมือซอฟต์แวร์สำหรับสร้างข้อมูลทดสอบ โดย ภพพงศ์ สกุลพิพัฒน์ศิลป์ (2544) เป็นการพัฒนาเครื่องมือเพื่อใช้สำหรับสร้างข้อมูลทดสอบเพื่อเตรียมให้กับผู้ทดสอบซอฟต์แวร์ โดยเครื่องมือนี้พัฒนาด้วยโปรแกรมวิชวลเบสิก (Visual Basic) เครื่องมือนี้สามารถสร้างข้อมูลที่อยู่ในรูปแบบแฟ้มข้อมูลประเภทข้อความ และแฟ้มข้อมูลประเภทไมโครซอฟต์แอคเซต ซึ่งเครื่องมือนี้จัดเป็นเครื่องมือที่สนับสนุนการทดสอบซอฟต์แวร์ โดยใช้เทคนิคแบบแบล็กบอกซ์คือการทดสอบที่มุ่งเน้นที่จะตรวจสอบการว่ามีการนำข้อมูลอะไรเข้าสู่ระบบและจะได้ผลลัพธ์อะไรออกมาจากระบบ ซึ่งหน้าที่ของเครื่องมือนี้คือจะสร้างข้อมูลทดสอบที่ใกล้เคียงกับข้อมูลที่จะใช้จริงเตรียมไว้ให้กับผู้ทดสอบ

ในช่วงเวลาเดียวกัน และหน่วยงานเดียวกันได้มีผู้วิจัยเกี่ยวกับการพัฒนาระบบจัดการกรณีทดสอบซอฟต์แวร์ โดยธนพล สิชฌนกฤษฎ์ (2544) งานวิจัยเป็นการพัฒนาเครื่องมือเพื่อใช้สำหรับสร้างกรณีทดสอบเข้าสู่ระบบ ช่วยเก็บบันทึกรวบรวมข้อมูลกรณีทดสอบ และผลการทดสอบแล้วนำข้อมูลไปสรุปเป็นรายงาน รายงานแสดงความก้าวหน้าในการทดสอบการและการแก้ไขข้อผิดพลาดของโปรแกรม กรณีทดสอบที่งานวิจัยนี้พัฒนาขึ้นและรายงานในระบบมีรูปแบบที่ปรับปรุงจากมาตรฐานของ IEEE Std 829-1998 ซึ่งเครื่องมือนี้จัดเป็นเครื่องมือที่สนับสนุนการทดสอบซอฟต์แวร์ โดยใช้เทคนิคแบบแบล็กบอกซ์เช่นกัน

งานวิจัยสองชิ้นที่กล่าวไปข้างต้น เป็นงานที่วิจัยโดยคนไทย ส่วนงานวิจัยที่จะกล่าวถึงต่อไปได้จากการงาน Conference จาก IEEE ซึ่งล้วนแล้วแต่เกี่ยวข้องกับกระบวนการทดสอบซอฟต์แวร์ทั้งสิ้น และยังเกี่ยวข้องกับการทดสอบบด้วยเทคนิคแบบไวท์บอกซ์ อีกด้วยโดยงานแรกเป็นงานวิจัยที่ศึกษาถึงการสร้างข้อมูลทดสอบที่ให้สามารถทดสอบกับ โปรแกรมให้ครบทุกเส้นทางในโปรแกรม (Neelam Gupta, Aditya P. Mathur, Mary Lou Softa, 2000) โดยเป็นการตรวจสอบโปรแกรมให้คลอบคลุมทุกเส้นทาง ถือเป็นสิ่งสำคัญสำหรับใช้ในระหว่างการตรวจสอบโครงสร้างของโปรแกรม โดยในงานวิจัยจะพูดถึงพื้นฐานการทำงานของโปรแกรมใหม่ในเรื่องของ

การสร้างข้อมูลนำเข้า (generate input data) เพื่อใช้ฝึกฝนการเลือกเส้นทางในโปรแกรมในการสร้างส่วนสร้างข้อมูลทดสอบนั้นโปรแกรมจะทำการกำหนดเส้นทางขึ้นมาให้เลือกด้วยตัวโปรแกรมเองจากเส้นทางหลัก ๆ ที่จะมีได้ โดยข้อมูลนำเข้า ตัวใหม่แต่ละตัวจะได้จากการกำหนด ในการทดลองเพื่อหาประสิทธิภาพในการทำงานให้ผ่านพ้นไปได้ตลอดเส้นทางที่ถูกเลือก วิธีการแบบ dynamically จะเป็นลักษณะสับเปลี่ยนระหว่างเส้นทางที่สามารถไปได้ โดยจะพิจารณาจากข้อมูลนำเข้าที่มีการใช้เทคนิคแบบนับจำนวนรอบของส่วนวนซ้ำ (iterative) ที่เกิดขึ้นในการทดลอง โดยที่ dynamically จะพิจารณาเลือกเส้นทางที่มีแรงต้านน้อย ๆ หมายถึงเส้นทางที่ไปได้ ง่ายกว่าซึ่งจุดหลัก ๆ ที่งานวิจัยนี้กล่าวถึงก็คือ เทคนิคที่ใช้และ experimental result ที่ได้จากความสามารถของบางโปรแกรม ซึ่งผลลัพธ์ที่ได้จะเป็นตัวแสดงให้เห็นความเป็นไปได้และการทดลอง ของวิธีการ

ในภาพรวมของงานวิจัยเขา ได้จำลองการทำงานการสร้างข้อมูลทดสอบ และเส้นทางต่าง ๆ แสดงออกเป็นกราฟ โดยในกราฟจำแนกข้อมูลได้เป็น search predicate, critical predicates, path selection สำหรับ algorithm ที่ใช้ในการสร้างข้อมูลทดสอบเพื่อใช้ฝึกฝนเส้นทางเป็นข้อมูลจำพวก array, assignments, conditionals และ loops ในการทำงานจะผสม integer, real ให้เป็นข้อมูลที่มีขนาดเดียวกันให้กับการคำนวณ ด้วยวิธี linear constraints โดยวิธีนี้สามารถใช้ได้กับข้อมูลทั้งที่ใช่และไม่ใช่ numeric ได้ นอกจากความสามารถที่ได้กล่าวมาแล้วข้างต้นวิธีการนี้ยังเป็นลักษณะแบบ nonlinear และยังสามารถนำไปใช้ได้กลับโปรแกรมที่มีภาษาที่แตกต่างได้ เหมาะกับการทำงานที่เป็นแบบอัตโนมัติ

งานวิจัยต่อมาเป็นงานเกี่ยวกับการทดสอบสร้างข้อมูลทดสอบโดยอัตโนมัติเช่นเดียวกัน แต่ในงานวิจัยนี้มุ่งความสนใจไปที่ชนิดของข้อมูลด้วยกฎเกณฑ์โปรแกรมที่ประกอบด้วยตัวแปร ชนิด integer, boolean และ float (Nguyen Tran Sy, Yves Deville, 2001) ซึ่งใช้เทคนิคที่สอดคล้องกับการรวมกันของตัวแปร integer และ float มีการจัดการกับแต่ละเส้นทางและ path ครอบคลุมเงื่อนไขเป้าหมายเพื่อสร้างข้อมูลทดสอบอัตโนมัติซึ่งจะเป็นตัวที่ทำให้โปรแกรมทำงานตามคำสั่งในการข้ามเส้นทางหรือข้าม path ที่ระบุเพื่อให้ครอบคลุม path การระบุ path ที่จะให้ทำงานตามข้อกำหนดของ path นั้นทำได้โดยอาศัยพื้นฐานช่วงเวลาที่จำกัดหา algorithm ที่จัดการกับตัวแปร integer, boolean และ real การทดสอบที่ถูกต้องนั้นข้อมูลนำเข้าจะมีการขยายออกจากช่วงของการแก้ปัญหาสำหรับคำสั่งที่ครอบคลุม path จะมีคำสั่งเฉพาะหรือโครงสร้างแบบ dynamic สำหรับ algorithm ที่ใช้ได้สำหรับ path ครอบคลุมเมื่อมีการประยุกต์ใช้การค้นหา path ที่เหมาะสมและหาทางแก้ไขข้อจำกัดของ path ทำอย่างกว้างขวางด้วยความสอดคล้องของเทคนิค มีวัตถุประสงค์เพื่อหาแนวคิดที่จะสร้างความสอดคล้องกันอย่างง่าย ๆ เรียกว่า eBox consistency โดยทั่วไป box สอดคล้องกับ integer และ float eBox consistency เป็นสิ่งที่ตอบจุดประสงค์ได้เพียงพอในต้นแบบการพัฒนาและ experimetal results จะแสดงให้เห็นถึงความเป็นไปได้ในแนวคิดนี้และงานนี้มีการขยายผลต่อสำหรับตัวแปร

แบบ float และ boolean สำหรับงานนี้โปรแกรมอยู่ภายใต้การวิเคราะห์ตั้งแต่การแปลครั้งแรกในรูปแบบ Static Single Assignment (SSA) สำหรับครอบคลุม path มีข้อจำกัดของ path กับชนิดตัวแปรที่เป็น input ซึ่งเป็นสิ่งที่ได้มาจากการระบุ path และ program ในรูปแบบ SSA ซึ่งในงานนี้จะแสดงให้เห็นว่าจะมีการสร้างเมื่อมีข้อจำกัดอย่างไร

ท้ายสุดเป็นงานวิจัยที่กล่าวถึงยุทธศาสตร์การทดสอบเป็นการทดสอบแบบแยกกลุ่มตามเกณฑ์ซึ่งในระบบต้องการให้มีคู่ของ parameter ที่เป็นข้อมูลเข้าทุก ๆ การรวมกันของค่าที่ถูกต้องของ 2 parameter ที่ครอบคลุมโดยที่มันจะเป็นการทดสอบเล็ก ๆ ตัวหนึ่งผู้วิจัยได้มีเป้าหมายคือหายุทธศาสตร์ในการทดสอบตัวใหม่สำหรับการทดสอบแบบ (Kuo-chung Tai and Yu Lei, 2002) เป็นการนำเสนอ IPO ซึ่งเป็นยุทธศาสตร์ในการทดสอบสำหรับการทดสอบแบบ Pairwise ผลการทดลองจะเป็นตัวยืนยันยุทธศาสตร์ IPO ว่าสามารถทำงานได้ดีเข้ากันได้กับขนาดของการสร้าง test set และจำนวนของเวลาที่ใช้ไปในการ test นอกจากนี้ IPO ยังสามารถประยุกต์ใช้นำกลับมาใช้ใหม่ได้ง่ายเมื่อระบบมีการปรับปรุงในส่วนของการเปลี่ยน input parameter หรือค่าของมัน Pairwise testing หรือเรียกอีกอย่างว่า 2-way testing เป็นกรณีพิเศษของ n-way testing ซึ่งมีความต้องการของแต่ละ set ของ n-input parameter ของระบบทุก ๆ การรวมกันของค่าที่ถูกต้องของ n parameter เป็นการครอบคลุมถึงส่วนย่อย ๆ ของ test case หนึ่ง ๆ ปัญหาหนึ่งที่พบใน Pairwise testing คือถ้าตัวหลักของ input parameter มีขนาดใหญ่จำนวนของการทดสอบจะใหญ่มากสำหรับระบบ ในแต่ละ parameter มีค่า d จำนวนของการทดสอบที่ต้องการสำหรับ Pairwise testing เป็นค่าที่ เล็ก ๆ ของ d2 ดังนั้นถ้าแต่ละ parameter มีค่า 1000 ถึง 1 ล้าน การทดสอบที่ต้องการสำหรับ Pairwise testing และการแบ่งเบาปัญหานี้ใช้วิธีหารแต่ละ input หลักภายในแต่ละส่วนนั้น ๆ เลือกมาหนึ่งค่าจากแต่ละส่วนและสร้างการทดสอบที่เข้ากันได้กับค่าของ input parameter โดยควบคุมจำนวนของส่วนย่อย ๆ ให้กับแต่ละ input parameter ซึ่งสามารถเลือกจำนวนของการทดสอบให้สำหรับ Pairwise testing ได้

2.2 การตรวจสอบและการยืนยันความถูกต้อง (Validation and Verification)

การตรวจสอบและการยืนยันความถูกต้อง เป็นขั้นตอนที่ผู้พัฒนาสร้างความมั่นใจว่าซอฟต์แวร์ที่พัฒนานั้นตรงกับข้อกำหนด (specification) ที่ตั้งไว้และตรงกับความต้องการของผู้ใช้ระบบหรือไม่สรุปคือเพื่อวัตถุประสงค์ 2 ข้อหลัก (Ian Sommerville, 2001) คือ

1. ค้นหาข้อบกพร่องในระบบ
2. การทำงานของระบบเป็นไปตามการใช้งานหรือไม่

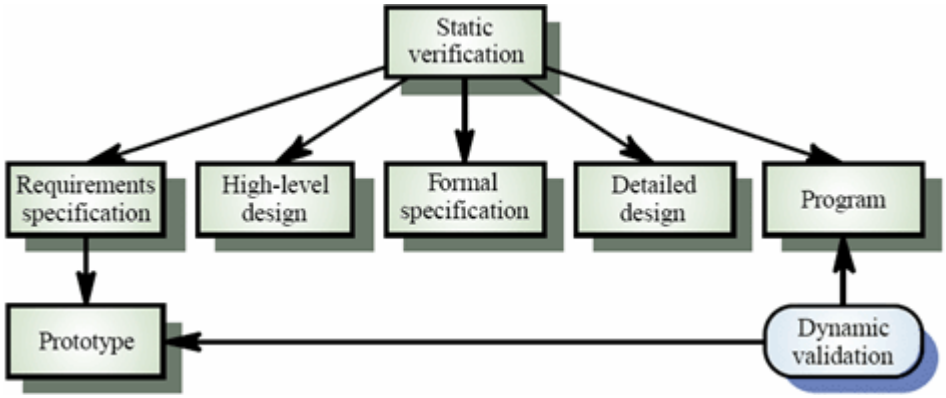
โดยเทคนิคในการวิเคราะห์และตรวจสอบซอฟต์แวร์ มี 2 วิธีคือ

2.2.1 เทคนิคแบบ Static (static verification)

เป็นการวิเคราะห์และตรวจสอบในซอฟต์แวร์เกี่ยวกับเอกสารไม่ว่าจะเป็นเอกสารข้อกำหนดเอกสารเกี่ยวกับการออกแบบรวมถึงเอกสารเกี่ยวกับตัวโปรแกรม ซึ่งเป็นเทคนิคที่สามารถประยุกต์ใช้ได้กับทุกขั้นตอนของกระบวนการตลอดการตรวจพิจารณามีลักษณะคงที่ไม่เปลี่ยนแปลง

2.2.2 เทคนิคแบบ Dynamic (dynamic verification)

เป็นการตรวจสอบในซอฟต์แวร์ขณะที่ซอฟต์แวร์กำลังทำงานมีลักษณะเปลี่ยนแปลงไปตามลำดับขั้นตอนต่าง ๆ ของโปรแกรมซึ่งจะใช้วิธีการทดสอบโปรแกรม (software testing)



รูปที่ 2.1 แสดงการใช้งาน static verification และ dynamic verification ในส่วนต่าง ๆ

2.3 การทดสอบโปรแกรม (program testing)

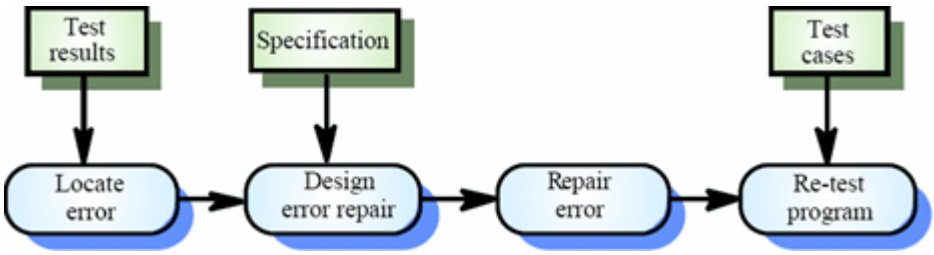
การทดสอบโปรแกรม เป็นการนำโปรแกรมไปทดสอบกับข้อมูลเหมือนเป็นข้อมูลจริง ๆ ซึ่งการทดสอบโปรแกรมแบ่งเป็น 2 ชนิดคือ

2.3.1 การทดสอบทางสถิติ (statistical testing)

เป็นการทดสอบการทำงานและความน่าเชื่อถือของโปรแกรม โดยทดสอบให้เห็นถึงความถี่ของการใช้งานในส่วนต่าง ๆ ของระบบและประเมินความน่าเชื่อถือในการทำงานของระบบ

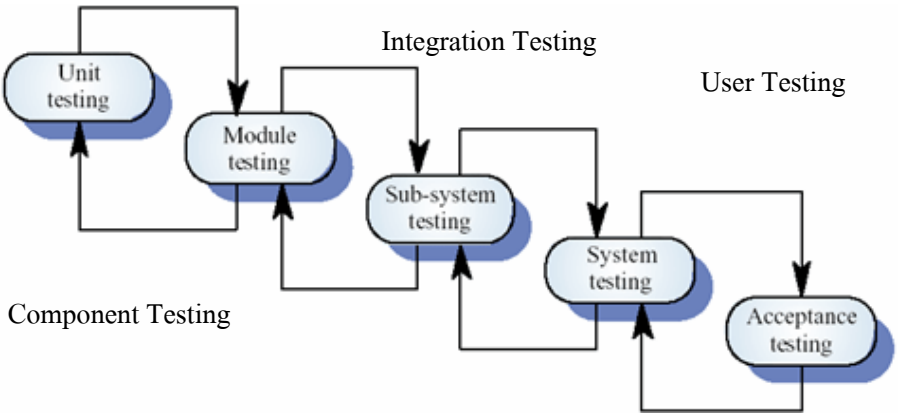
2.3.2 การทดสอบหาข้อบกพร่อง (defect testing)

เป็นการทดสอบเพื่อหาว่าจุดไหนของโปรแกรมที่ทำให้โปรแกรมไม่ตรงตามข้อกำหนด (specification) โดยทดสอบให้เห็นว่าจุดบกพร่องอยู่ที่ไหนเมื่อพบข้อบกพร่องจะได้ทำให้ข้อบกพร่องหมดไป (debugging)



รูปที่ 2.2 แสดงกระบวนการแก้ไขข้อผิดพลาด (debugging process)

กระบวนการทดสอบ (testing process) มีการแบ่งระยะ (phase) ของการทดสอบเป็น 3 ระยะใหญ่ ๆ คือการทดสอบระดับหน่วยหรือส่วนย่อย (unit/component testing) ต่าง ๆ การทดสอบการทำงานร่วมกัน (integration testing) และการทดสอบเพื่อการยอมรับของผู้ใช้ระบบ (acceptance/user testing) ในแต่ละขั้นตอนของการทดสอบเราจะดำเนินการแก้ไขข้อผิดพลาดที่เกิดขึ้น และหลังจากแก้ไขแล้วก็ต้องย้อนกลับมาทดสอบใหม่อีกครั้ง เป็นกระบวนการหมุนวนอยู่ภายในแต่ละขั้นตอน และในการแก้ไขแต่ละขั้นตอนอาจทำให้เกิดข้อผิดพลาดใหม่ ๆ ได้ทำให้ต้องย้อนกลับไปทดสอบในขั้นตอนที่ทดสอบผ่านมาแล้วก็ได้ ดังรูปที่ 2.3



รูปที่ 2.3 แสดงกระบวนการทดสอบโปรแกรม (testing process)

2.4 ระยะของการทดสอบโปรแกรม (testing phase)

กระบวนการทดสอบโปรแกรมสามารถแบ่งเป็นระยะของการทดสอบตามระดับของโปรแกรมออกเป็นระดับ (level) ต่าง ๆ ได้ดังนี้

2.4.1 การทดสอบระดับหน่วยหรือทดสอบส่วนย่อย (unit/component testing)

เป็นการทดสอบแต่ละหน่วยหรือส่วนย่อย ๆ ของโปรแกรมว่าทำงานได้ถูกต้องหรือไม่ โดยปกติแล้วมักจะถือว่าแต่ละส่วนนี้มีความเป็นอิสระสมบูรณ์ในตัวเองในขั้นนี้จึงไม่จำเป็นต้องสัมพันธ์กับหน่วยอื่น ๆ

2.4.2 การทดสอบระดับโมดูล (module testing)

โดยปกติแล้วโมดูล คือชุดของหน่วยย่อยต่าง ๆ ที่มีความเกี่ยวเนื่องกันอยู่ ดังนั้นในระยะนี้จึงเป็นการทดสอบการทำงานร่วมกันของแต่ละหน่วยย่อยในโมดูลเดียวกัน

2.4.3 การทดสอบระบบย่อย (sub-system testing)

เป็นการทดสอบการทำงานร่วมกันของโมดูลย่อย ๆ แต่ละระบบงานย่อยนี้อาจถูกพัฒนาขึ้นมาอย่างเป็นอิสระต่อกัน และอาจนำมาติดตั้งใช้งานโดยอิสระไม่เกี่ยวข้องกันก็ได้ ดังนี้ปัญหาที่พบในระบบงานขนาดใหญ่ คือการทำงานไม่สอดประสานกันระหว่างระบบงานย่อย ๆ ดังนั้นในระยะนี้จึงเป็นการทดสอบถึงปัญหาของการไม่สอดประสานกันของหน่วยย่อยต่าง ๆ

2.4.4 การทดสอบระบบ (system testing)

ระบบงานย่อย ๆ จะรวมกันทำให้เกิดเป็นระบบใหญ่ทั้งหมด การทดสอบการทำงานของระบบจึงเป็นการหาข้อผิดพลาดที่คาดไม่ถึงมาก่อน โดยที่เป็นข้อผิดพลาดซึ่งเกิดจากการขัดแย้งกันระหว่างระบบย่อยต่าง ๆ นอกจากนี้ยังเป็นการตรวจสอบว่าระบบทั้งหมดทำงานได้ตรงกับความต้องการของผู้ใช้อย่างแท้จริงหรือไม่

2.4.5 การทดสอบการยอมรับของผู้ใช้ระบบ (acceptance testing)

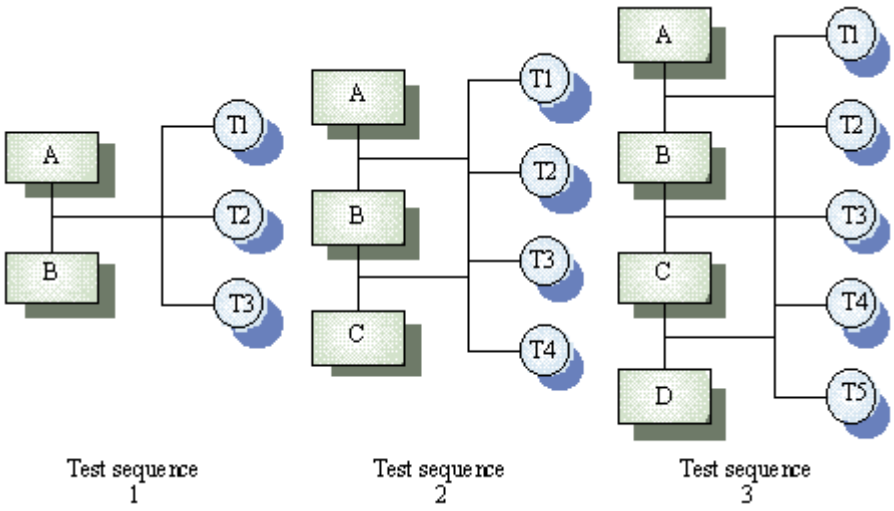
เป็นการทดสอบระยะสุดท้ายก่อนที่ระบบจะถูกยอมรับได้ว่าสามารถใช้งานได้จริง การทดสอบอาจกระทำได้โดยการใช้ข้อมูลจริง ๆ ป้อนเข้าสู่ระบบแทนที่จะใช้ข้อมูลตัวอย่าง การทดสอบเพื่อยอมรับมักจะช่วยทำให้เรามองเห็นว่าระบบทั้งหมดมีความผิดพลาดอย่างไร และยังไม่ได้ดำเนินการสิ่งใดบ้างที่ไม่ตรงกันกับข้อตกลง

2.5 กลยุทธ์ในการทดสอบโปรแกรม (testing strategies)

ในตำราของ John Watkins, 2001 กล่าวถึงกลยุทธ์ในการทดสอบโปรแกรมเอาไว้ในระบบงานใหญ่ ๆ โดยปกติจะทดสอบระบบ โดยการใช้เทคนิคในการทดสอบผสมกันไปมากกว่าที่จะใช้วิธีการเดียวเพราะเทคนิคหนึ่งอาจจำเป็นสำหรับส่วนของระบบส่วนหนึ่งและเทคนิคอื่นก็ใช้

ในส่วนอื่น ๆ ในการทดสอบโปรแกรม ถึงแม้เทคนิคการทดสอบจะถูกปรับปรุงเป็นหลายวิธี แต่มักจะปรับวิธีการโดยเริ่มจากทดสอบในส่วนย่อยแล้วเพิ่มส่วยย่อยอื่น ๆ เข้าไปทดสอบตาม ดังตัวอย่างในรูปที่ 2.4 ครั้งแรกทดสอบ T1, T2 และ T3 ซึ่งเป็นช่วงการทดสอบของการรวมโมดูล A และโมดูล B จากนั้นรวมโมดูล C เข้าไปทำการทดสอบ T1, T2 และ T3 ซ้ำเพื่อให้เกิดความมั่นใจว่าไม่เกิดข้อผิดพลาดกับ A และ B หลังจากเพิ่ม C เข้าไปแล้วทำการทดสอบ T4 ด้วย สุดท้ายเมื่อรวมโมดูล D เข้าไปก็ทำการทดสอบเช่นเดียวกันอีกทำอย่างนี้ไปเรื่อย ๆ จนกว่าทุกโมดูลจะถูกรวมเข้าไปในระบบ

ดังนั้นเทคนิคต่าง ๆ ที่ใช้ในการทดสอบโปรแกรมจึงมีทั้งที่ใช้สำหรับการทดสอบระดับหน่วย ระดับทดสอบการรวมระบบ และระดับทดสอบการยอมรับ ดังต่อไปนี้

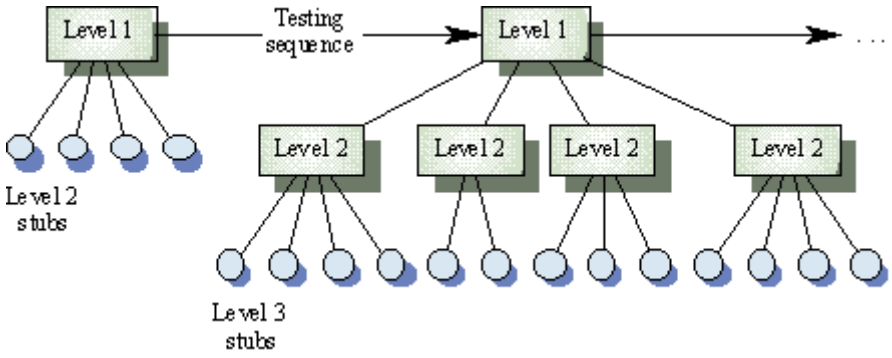


รูปที่ 2.4 แสดงการทดสอบโดยการรวมโมดูลเพิ่มในระบบทีละโมดูล

2.5.1 การทดสอบการรวมระบบจากบนลงล่าง (top-down testing)

เป็นการทดสอบในระดับบนสุดก่อนแล้วจึงทดสอบในรายละเอียดของส่วนย่อยของระบบ โดยตัวโปรแกรมถือเป็นส่วนย่อย (component) ที่มีส่วนย่อย ๆ อีกชั้นหนึ่ง (sub component) หลังจากที่ส่วนย่อยของระบบที่เป็น top-level ทดสอบแล้วส่วนย่อยลงไปอีกทีก็จะนำมาทดสอบในลักษณะเดียวกัน กระบวนการทดสอบดำเนินซ้ำเช่นนี้ไปเรื่อย ๆ จนกระทั่งส่วนย่อยที่เป็น bottom-level ถูกทดสอบเสร็จระบบทั้งหมดก็ถือว่าทดสอบอย่างสมบูรณ์

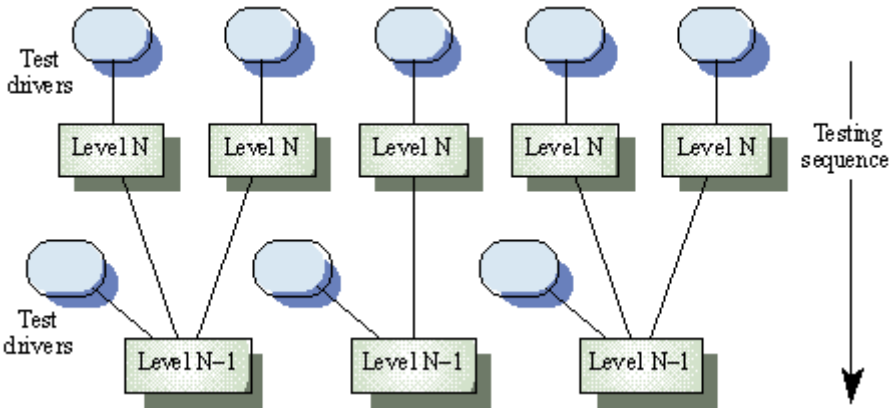
การทดสอบแบบ top-down ควรที่จะใช้กับการพัฒนาโปรแกรมแบบ top-down และในการทดสอบแบบนี้ข้อผิดพลาดที่เกิดขึ้นจะเป็นลักษณะความผิดพลาดของโครงสร้าง



รูปที่ 2.5 แสดงลักษณะการทดสอบโปรแกรมแบบบนลงล่าง (top-down testing)

2.5.2 การทดสอบการรวมระบบจากล่างขึ้นบน (bottom-up testing)

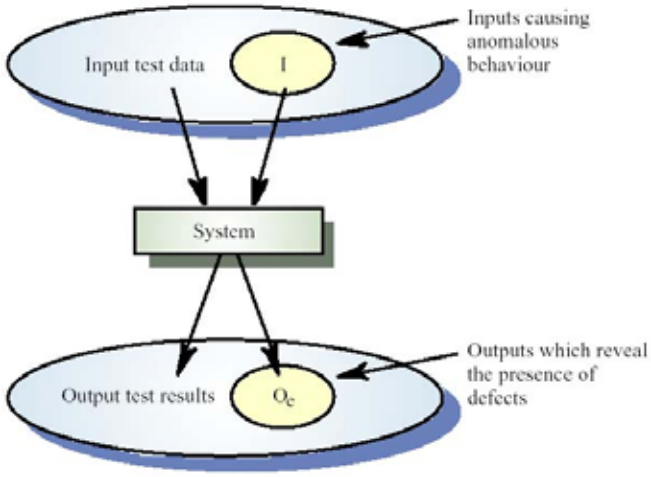
การทดสอบแบบนี้จะตรงข้ามกับการทดสอบแบบบนลงล่าง โดยจะมีการเริ่มต้นที่ระดับโปรแกรมล่างสุดก่อนแล้วทดสอบที่สูงขึ้นมาจนกระทั่งโมดูลสุดท้ายทดสอบเสร็จ การทดสอบแบบนี้เหมาะกับการพัฒนาระบบงานแบบเชิงวัตถุ (object-oriented) โดยที่วัตถุ (object) จะมีการทดสอบโดยการใช้ตัวขับ (test diver) ของตัวเองจากนั้นก็จะทดสอบกาทำงานประสานกันของวัตถุ เมื่อรวมกันโดยเน้นที่การติดต่อ ประสานงานกัน



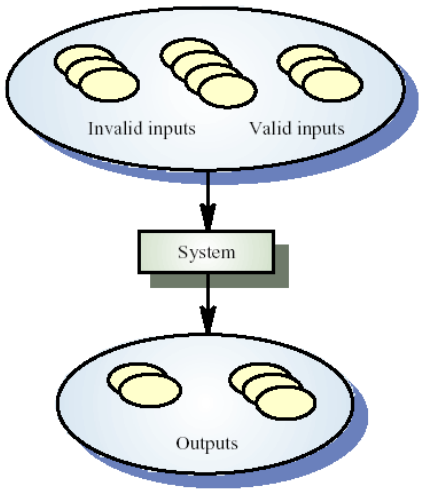
รูปที่ 2.6 แสดงลักษณะการทดสอบโปรแกรมแบบล่างขึ้นบน (bottom-up)

2.5.3 การทดสอบแบล็กบอกซ์ (black-box testing)

เป็นเทคนิคการทดสอบที่สนใจเฉพาะข้อมูล และผลลัพธ์ของโมดูลที่ต้องการทดสอบ เพื่อให้โปรแกรมที่ได้ตรงตามข้อกำหนดเท่านั้น ดังนั้นการทดสอบแบบนี้จึงตั้งอยู่บนการกำหนดรายละเอียด (specification) ว่าระบบต้องการข้อมูลเข้าคืออะไรและจะได้ผลลัพธ์อะไรออกมา รวมทั้งระบุด้วยว่าข้อมูลเข้าที่จะทำให้เกิดความผิดพลาดมีอะไรบ้าง และทำการทดสอบด้วยว่าเกิดความผิดพลาดตามที่คาดไว้หรือไม่



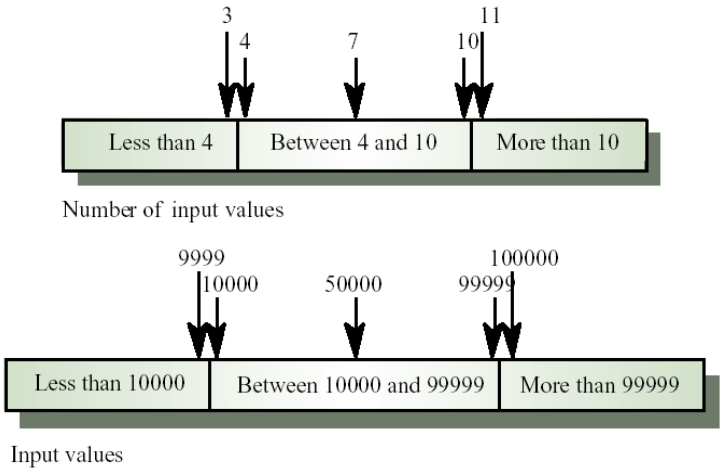
รูปที่ 2.7 แสดงแบบจำลองการทดสอบแบบแบล็กบอกซ์ (black-box testing)



รูปที่ 2.8 แสดงการแบ่งช่วงข้อมูลเข้าที่ปกติและที่ทำให้เกิดข้อผิดพลาด

ปัญหาที่สำคัญสำหรับการทดสอบแบบนี้คือการกำหนดข้อมูลเข้าหรือกรณีทดสอบ (test case) เพื่อให้มีความถูกต้องและเหมาะสม ซึ่งจะต้องอาศัยวิธีการในการกำหนดข้อมูลและสร้างกรณีทดสอบ ซึ่งแบ่งเป็น 2 ประเภทคือ

การแบ่งช่วงของข้อมูล (equivalence partitioning) เป็นเทคนิคเพื่อใช้ตัดสินว่าช่วงของข้อมูลเข้า (input data) ใดเป็นช่วงปกติและช่วงใดเป็นชั้นที่ทำให้เกิดข้อผิดพลาด ดังแสดงในรูปที่ 2.8 การแยกประเภทของช่วงข้อมูลได้โดยพิจารณาจากข้อกำหนด (program specification) และเงื่อนไขของข้อมูลเข้าจากเอสารของผู้ใช้งาน (user documentation) หรือจากประสบการณ์ของผู้ทดสอบเองว่าจะให้ช่วงใดเป็นข้อมูลที่ถูกต้องและช่วงใดเป็นข้อมูลที่ทำให้เกิดข้อผิดพลาดขึ้นในโปรแกรม ผลที่ได้จากขั้นตอนของการแยกช่วงของข้อมูลเข้า (input equivalence class) ช่วงของข้อมูลเข้าที่ถูกต้อง (valid class) และช่วงของข้อมูลเข้าที่ทำให้เกิดข้อผิดพลาด (invalid class) สำหรับเงื่อนไขของข้อมูลเข้านั้นตัวอย่างเช่น ในเอกสารข้อกำหนดระบุว่าโปรแกรมจะรับข้อมูลเข้าเป็นจำนวนเต็มขนาด 5 หลักได้ 4-8 ตัวและมีค่ามากกว่า 10000 ข้อมูลที่ได้จะเป็น 0, 9999, 10000, 10001, 99999, 100000 และลักษณะของการพิจารณาแบ่งช่วงข้อมูลก็จะได้ดังรูปที่ 2.9



รูปที่ 2.9 แสดงการแบ่งช่วงข้อมูลที่กำหนดไว้ และข้อมูลเข้าที่เป็นไปได้และใช้สำหรับการทดสอบ

ตารางที่ 2.1 แสดงตัวอย่างการกำหนดข้อมูลให้กับอาร์เรย์ และผลลัพธ์ที่จะเกิดขึ้นในการค้นหาข้อมูลที่อยู่ในอาร์เรย์

Array	Element
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

อีกตัวอย่างหนึ่งก็คือกำหนดข้อมูลเข้าให้กับการค้นหาตัวเลขในข้อมูลแบบอาร์เรย์ (array) ดังตารางที่ 2.1 และ ตารางที่ 2.2

ตารางที่ 2.2 แสดงตัวอย่างการกำหนดข้อมูล และผลลัพธ์ที่จะได้ในการค้นหาข้อมูล

Input sequence (T)	Key (key)	Output (found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41,18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

การวิเคราะห์ค่าขอบเขต (boundary value analysis) เนื่องจากข้อผิดพลาดที่เกิดขึ้นในโปรแกรมส่วนใหญ่จะเกิดในบริเวณที่เป็นค่าขอบเขตของข้อมูลทำให้เมื่อเราสร้างช่วงข้อมูลแล้วต้องมีการทำขั้นตอนของการวิเคราะห์ค่าขอบเขตเพื่อให้กรณีทดสอบที่สร้างขึ้นมีประสิทธิภาพมากที่สุดในการค้นหาข้อผิดพลาดรูปแบบของการวิเคราะห์ค่าขอบเขตสำหรับลักษณะเฉพาะหรือเงื่อนไขของข้อมูลเข้าโดยทั่วไปมีดังนี้

1. ถ้าข้อมูลเข้าระบุค่าอยู่ในช่วงของข้อมูล a และ b ข้อมูลสำหรับทดสอบค่าขอบเขตในกรณีทดสอบที่สร้างขึ้นควรมีค่าอยู่ เหนือและใต้ค่า a และ b เล็กน้อย

2. ถ้าข้อมูลเข้าระบุจำนวนมากที่สุด หรือน้อยที่สุดของข้อมูลเข้า กรณีทดสอบที่สร้างควรทดสอบค่ามากที่สุด น้อยที่สุด นั้น และทดสอบบริเวณที่เป็นขอบเขต คือเหนือและใต้จำนวนมากที่สุด น้อยที่สุดนั้นด้วย

3. ประยุกต์วิธีการตามข้อ 1 และ 2 สำหรับผลลัพธ์ที่ต้องการ

4. ถ้าโครงสร้างภายในของโปรแกรมมีการกำหนดขอบเขตไว้สร้างข้อมูลทดสอบเพื่อทดสอบโครงสร้างนั้น เช่น ถ้าโปรแกรมกำหนดข้อมูลแบบอาร์เรย์ขนาด100 ข้อมูลต้องสร้างข้อมูลทดสอบในบริเวณที่เป็นขอบ (99, 100, 101) เขตสำหรับทดสอบข้อมูลอาร์เรย์ขนาดนั้น

2.5.4 การทดสอบแบบไวท์บอกซ์ (white-box testing)

บางครั้งเรียกว่าการทดสอบโครงสร้างของโปรแกรม (structural testing) เป็นการทดสอบที่พิจารณาที่โครงสร้างของโปรแกรมโดยสร้างกรณีทดสอบที่ครอบคลุมทุกเส้นทาง (path) เพื่อให้แน่ใจว่าทุกคำสั่งในโปรแกรมได้รับการทดสอบอย่างน้อย 1 ครั้ง และทุกคำสั่งที่เป็นเงื่อนไขถูกทดสอบทั้งกรณีเงื่อนไขที่เป็นจริง และเป็นเท็จนอกจากนี้ยังสามารถค้นพบข้อผิดพลาดที่เกิดขึ้นจากข้อมูลภายในโมดูลได้อีกด้วย การทดสอบด้วยวิธีการของไวท์บอกซ์ผู้ทดสอบจะต้องสร้างกราฟการควบคุมกระแส จากโปรแกรมที่พัฒนาขึ้น และนำกราฟที่ได้มาสร้างกรณีทดสอบเพื่อให้ครอบคลุมทุกคำสั่งทุกเส้นทาง และทุกเงื่อนไขของโปรแกรม

กราฟที่แทนโครงสร้างควบคุมของโปรแกรม และโครงร่างของเส้นทางสำหรับทดสอบนั้นจะประกอบได้ด้วยเส้น (edges) ที่แทนเส้นทางการควบคุมของคำสั่ง และโหนด (nodes) ซึ่งอาจแทนด้วยคำสั่งควบคุมจำพวกทางเลือกหรือวนซ้ำ เช่น if-then-else, while, do-while, for หรือ case โดยไม่รวมคำสั่งประเภท goto

ในเส้นทางที่ได้จากกราฟนั้นจะเป็นเส้นทางอิสระของโปรแกรม (cyclomatic complexity : CC (G)) ซึ่งภายในแต่ละเส้นทางอิสระจะต้องมีอย่างน้อยเส้น (edge) ที่ไม่ซ้ำกับเส้นในเส้นทางอิสระเส้นทางอื่น ๆ ซึ่งสามารถคำนวณได้จากสูตร (Ian Sommerville, 2001)

CC(G) = จำนวนเส้น (edges) – จำนวนโหนด (nodes) + 2

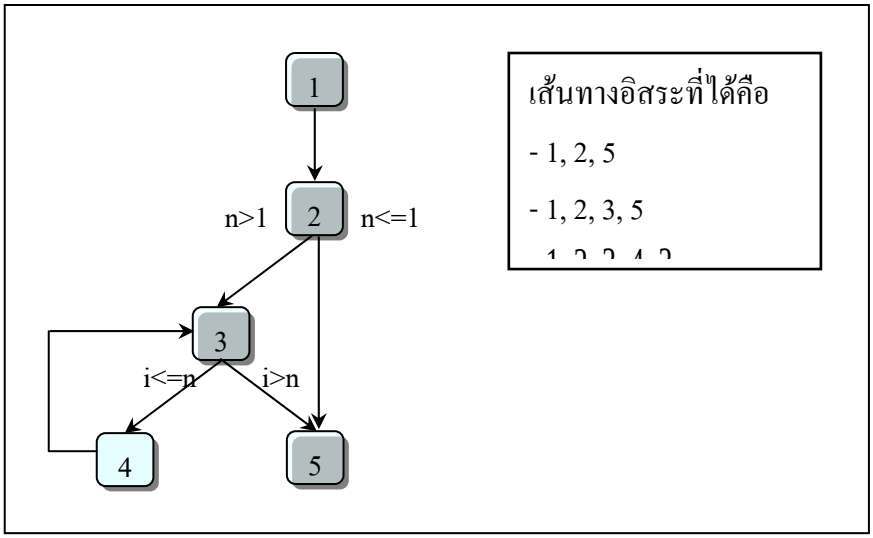
เส้นทางที่ได้ไม่ซ้ำกันแต่ต้องครอบคลุมทุกเส้นทางดังตัวอย่างโปรแกรมที่ทำหน้าที่คำนวณหาค่าแฟคทอเรียล n แสดงในรูป 2.10

หลังจากพบจำนวนของเส้นทางอิสระของโปรแกรมทั้งซอร์สโค้ด และขั้นตอนต่อไปคือการออกแบบกรณีทดสอบ (test case) เพื่อที่จะประมวลผลแต่ละเส้นทางนั้นจำนวนน้อย

ที่สุดของกรณีทดสอบที่จะถูกกำหนดในการทดสอบเส้นทางโปรแกรมจะเท่ากับจำนวนเส้นทางอิสระของโปรแกรม นั่นเอง

```
long int factorial (int n)
{
        int i;
        long int prod = 1;
        if (n>1)
                for (i=2;i<=n;++i)
                        prod *= i;
        return(prod);
}
```

รูปที่ 2.10 แสดงตัวอย่างโปรแกรมค้นหาข้อมูลเขียนด้วยภาษาซี



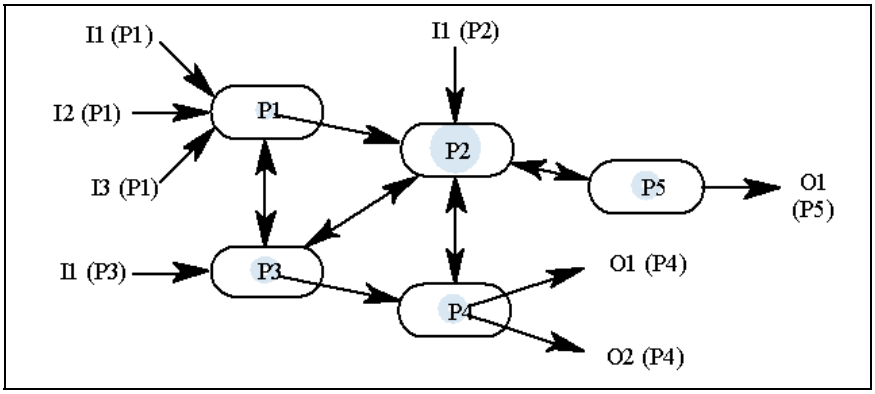
รูปที่ 2.11 แสดงตัวอย่างกราฟควบคุมกระแส และเส้นทางทดสอบ ที่สร้างจากซอร์สโค้ดในรูปที่ 2.10

2.5.5 การทดสอบความเกี่ยวเนื่องกัน (thread testing)

เป็นเทคนิคที่สามารถทดสอบระบบ Real-Time คือทดสอบโดยขึ้นกับเหตุการณ์ (event) ที่เกิดการกระทำในระบบ ดังนั้นการทดสอบนี้สามารถใช้กับระบบงานแบบเชิงวัตถุที่มีลักษณะแบบ Event-Driven จะใช้หลังจากที่วัตถุถูกทดสอบไปแล้ว ตัวอย่างดังรูปที่ 2.12 ที่มี 5 กระบวนการ (process) ที่ติดต่อกันบางกระบวนการรับข้อมูลจากสิ่งแวดล้อม เช่น รับข้อมูลจากตัวตรวจจับ (sensor) แป้นพิมพ์ (keyboard) หรือจากคอมพิวเตอร์อื่น ๆ แล้วสร้างข้อมูลออก (output) ส่งไปยังสิ่งแวดล้อมอื่น ๆ ซึ่งอาจเป็นหน้าจอหรือส่วนอื่น ๆ

กรณีรับข้อมูลเข้าจากสิ่งแวดล้อมให้ขึ้นต้นด้วย I ส่วนข้อมูลออกให้ขึ้นต้นด้วย O ลูกศรที่เชื่อมระหว่างกระบวนการ หมายถึงเหตุการณ์หนึ่งจะถูกสร้าง (generate) ได้ก็ต้องอาศัยข้อมูลเข้า ซึ่งเป็นข้อมูลออกจากเหตุการณ์หนึ่ง

ในส่วนของการทดสอบนั้นระบบจะทำการทดสอบเพื่อกำหนดสายโซ่ (thread) ต่าง ๆ ที่จะเกิดขึ้นที่เป็นไปได้ที่จะนำมาใช้ในการทดสอบ ซึ่งสายโซ่เหล่านี้ไม่ได้สัมพันธ์กับเหตุการณ์เพียงเหตุการณ์เดียวเท่านั้น แต่ยังรวมข้อมูลเข้ารวมเข้าไปด้วยซึ่งสามารเพิ่มได้



รูปที่ 2.12 แสดงกระบวนการที่รับส่งข้อมูลระหว่างกัน (interacting process)

2.5.6 การทดสอบการเชื่อมต่อของระบบ (interface testing)

การทดสอบการเชื่อมต่อของระบบนี้จะกระทำได้เมื่อมีการรวมระบบย่อย (sub ystem) เข้าด้วยกันแสดงได้ดังรูปที่ 2.13 โดยจะเป็นการทดสอบเพื่อหาข้อผิดพลาดที่อาจทำให้ระบบล้มเหลวที่เกิดจากจุดเชื่อมต่อหรือการติดต่อระหว่างระบบย่อย เช่น การส่งค่าระหว่างกันผิดรูปแบบหรือส่งไม่ครบ ข้อผิดพลาดที่อาจจะเกิดขึ้นมีดังนี้

1. ข้อมูลที่รับส่งระหว่างกัน (parameter interface)

2. หน่วยความจำที่ใช้ร่วมกัน (shared memory interface)

3. งานบางอย่างที่เรียกใช้ร่วมกัน (procedural interface)

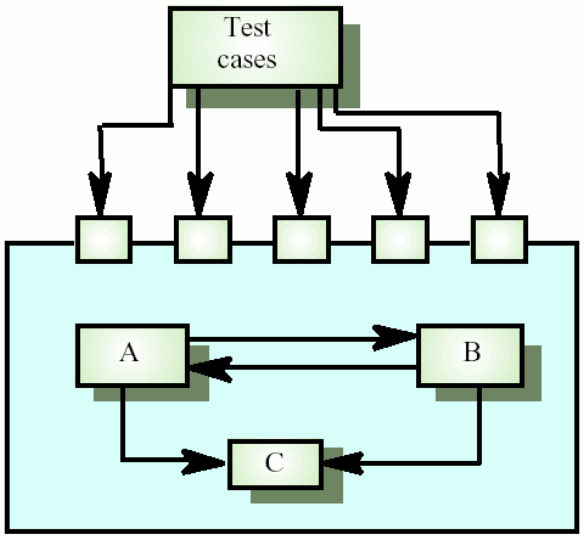
4. การติดต่อกันระหว่างโมดูล (message passing interface)

รูปแบบของข้อผิดพลาดจะอยู่ในลักษณะ 3 ลักษณะดังต่อไปนี้

1. การกำหนดค่าที่ใช้ส่งผ่านระหว่างระบบย่อยไม่ถูกต้องตามที่แต่ละระบบย่อยนั้น ๆ กำหนดไว้ (interface misuse)

2. การเข้าใจผิดเกี่ยวกับการติดต่อหรือรายละเอียดของการติดต่อระหว่าง ระบบย่อย (interface misunderstanding)

3. ความผิดพลาดที่เกิดจากการติดต่อกันระหว่างโมดูลมีความเร็วที่ต่างกัน (timing errors)



รูปที่ 2.13 แสดงภาพจำลองการติดต่อกันระหว่างระบบย่อย ภายในระบบ

2.5.7 การทดสอบความคงทนของระบบ (stress testing)

เป็นการทดสอบว่าระบบสามารถทำงานภายใต้สภาพการใช้งานที่สูงมาก ๆ ได้มากน้อยแค่ไหน เพื่อต้องการทราบความสามารถสูงสุดของระบบภายใต้ทรัพยากรที่กำหนด เช่น การทดสอบจำนวนรายการที่เกิดขึ้น (transaction) มากที่สุดที่ระบบสามารถรองรับได้ เป็นต้น การทดสอบแบบนี้มักจะนำไปใช้ในระบบแบบกระจาย (distributed system) ใช้ตรวจสอบความทนทานของระบบเครือข่ายของตัวประมวลผล เนื่องจากระบบแบบกระจายมักจะเกิดความผิดปกติเมื่อใช้งานอย่างหนัก

2.5.8 การทดสอบที่เสริมกัน (positive and negative testing)

เป็นการทดสอบเพื่อดูว่าระบบสามารถทำงานได้ตามที่กำหนดไว้หรือไม่ และระบบไม่ทำตามที่กำหนดไว้หรือไม่ กล่าวคือเป็นการทดสอบแบบเสริมกันทางบวก (positive) และทางลบ (negative) โดยกระบวนการของการทดสอบแบบเสริมกันทางบวกจะมุ่งเน้นที่จะตรวจสอบว่าระบบเป็นไปตามต้องการ หรือข้อกำหนดหรือไม่ ส่วนกระบวนการของการทดสอบแบเสริมกันทางลบจะมุ่งเน้นที่จะแสดงให้เห็นว่า ระบบไม่ทำในสิ่งที่ควรกระทำตามที่กำหนดไว้

2.5.9 การทดสอบแบบฝังคำสั่ง (intrusive testing)

เป็นการทดสอบที่มีการปรับเปลี่ยนอะไรบ้างอย่างลงไปในตัวโปรแกรม เพื่อให้แสดงอะไรบางอย่างเพื่อช่วยในการทดสอบ โดยผู้ทดสอบจะทำโดยเพิ่มคำสั่งอะไรบางอย่างเข้าไปในโปแกรมเพื่อแสดงผลตัวแปรหรือค่าที่ต้องการตรวจสอบ และเมื่อจะต้องส่งมอบโปรแกรมจะต้องนำคำสั่งที่เพิ่มออกไปด้วย

2.5.10 การทดสอบโครงแบบ (configuration/installation testing)

เป็นการทดสอบที่ตรวจสอบการติดตั้งและกำหนดค่าว่าถูกต้องหรือไม่ เพื่อต้องการทราบข้อกำหนดของฮาร์ดแวร์ (hardware) ที่เหมาะสม

2.5.11 การทดสอบการกู้ระบบ (Recovery testing)

เป็นการทดสอบเพื่อแสดงว่าระบบสามารถกู้ข้อมูลกลับคืนมาได้หากระบบเกิดความเสียหายขึ้น

2.5.12 การทดสอบประสิทธิภาพ (Performance testing)

เป็นการทดสอบเพื่อแสดงว่าระบบมีประสิทธิภาพตามที่ได้กำหนดไว้หรือไม่ และเพื่อปรับแต่งระบบให้มีประสิทธิภาพดีขึ้นโดยการตรวจสอบจากเวลาที่ตอบสนอง (response time) ปริมาณงาน (throughput) การใช้งานหน่วยประมวลผลกลาง (CPU utilization) เป็นต้น

2.5.13 การทดสอบความปลอดภัย (Security testing)

เป็นการทดสอบแสดงว่าระบบมีประสิทธิภาพเพียงพอในการรักษาความปลอดภัยและป้องกันระบบจากการลักลอบใช้งานของผู้ที่ไม่ได้รับอนุญาต กล่าวคือมีระบบรักษาความปลอดภัยอยู่ระดับใด เช่น การเก็บรักษาความลับ (confidentiality) สิทธิการเข้าถึง (availability) ความครบถ้วนของข้อมูล (integrity of the data and software)

2.6 การเขียนโปรแกรมด้วยภาษาซีเบื้องต้น (fundamental C programming language)

โปรแกรมภาษาซีจัดเป็นโปรแกรมภาษาระดับสูง (high level language) อยู่ในยุคที่ 2 ของโปรแกรมภาษาคอมพิวเตอร์ มีรูปแบบการพัฒนาแบบเชิงโครงสร้าง ถูกพัฒนาขึ้นในช่วงเวลาเดียวกันกับภาษาปาสคาล โดยนักวิจัยที่ห้องปฏิบัติการ AT&T Bell ซึ่งได้นำเอาจุดเด่นของภาษาบีซีพีแอล (BCPL) และภาษาบี (B) มาใช้และได้เพิ่มคุณลักษณะและชนิดข้อมูลอื่นเข้ามาด้วยได้รับการยอมรับและนำมาพัฒนาเป็นโปรแกรมระบบ (systems programming) เช่น ภาษาซีถือว่าเป็นภาษาคอมพิวเตอร์ที่สำคัญในการพัฒนาโปรแกรมบนระบบปฏิบัติการยูนิกส์ ซึ่งเป็นภาษาคอมพิวเตอร์ที่สามารถสร้างโปรแกรมประยุกต์ที่ทำงานได้รวดเร็วมาก เมื่อเทียบกับภาษาคอมพิวเตอร์อื่น ๆ

จุดเด่นที่สำคัญอีกอย่างของภาษาซีคือการมีโครงสร้างที่ดีทั้งในแง่ของโครงสร้างภาษาโครงสร้างชนิดข้อมูลความสะดวกสบายในการเขียนและไม่ขึ้นกับระบบเครื่อง การใช้งานภาษาซีตามตำราของ Byron S. Gottfried, Ph.D (1996) มีรายละเอียดและหลักการใช้ดังต่อไปนี้

2.6.1 ชนิดข้อมูลและค่าคงที่ (data type and constants)

ตัวแปรทุกตัวต้องมีการประกาศตัวแปรด้วยชนิดข้อมูลต่าง ๆ โดยข้อมูลอาจเป็นค่าคงที่ ตัวแปรหรือตัวแปรโครงสร้างที่ประกอบจำนวนเลข (integer และ real) อักขระ (text) หรือเลขที่อยู่ (address) ของตัวแปรก็ได้

ชนิดของข้อมูล (data type) ข้อมูลพื้นฐานที่ใช้ในภาษาซีได้แก่

ตารางที่ 2.3 แสดงชนิดข้อมูลพื้นฐาน

ชนิดตัวแปร	คำอธิบาย	หน่วยความจำที่ใช้
int	เลขจำนวนเต็มฐานสิบ	1 ไบต์ (8 บิต)
char	ตัวอักษร	1 ไบต์ (8 บิต)
float	เลขทศนิยม ที่ไม่มีเลขยกกำลัง	4 ไบต์ (32 บิต)
double	เลขทศนิยม ที่มีเลขชี้กำลัง	8 ไบต์ (64 บิต)

นอกจากนี้ชนิดของข้อมูลยังสามารถมีตัวขยาย (qualifier) ซึ่งใช้ขยายข้อมูลพื้นฐานข้างต้นเช่น short กับ long ซึ่งทำให้ชนิดของข้อมูลเหมือนกันแต่มีขนาดเพิ่มขึ้น

ตารางที่ 2.4 แสดงตัวขยาย (Qualifier) ของชนิดข้อมูล

ชนิดตัวแปร	คำอธิบาย	หน่วยความจำที่ใช้
short int	เลขจำนวนเต็มฐานสิบแบบสั้น	2 ไบต์ (16 บิต)
long int	เลขจำนวนเต็มฐานสิบแบบยาว	4 ไบต์ (32 บิต)
long double	เลขทศนิยม ที่มีเลขชี้กำลัง มีขนาดใหญ่กว่า double	10 ไบต์ หรือมากกว่า

ตัวขยาย signed (เครื่องหมายบวกหรือลบ) และ unsigned (ไม่มีเครื่องหมายมีค่าเป็นบวก) สามารถใช้ขยาย char หรือ int ตัวเลขแบบ unsigned จะมีค่าเป็นบวกหรือศูนย์เท่านั้น เช่น signed char มีค่าระหว่าง -128 ถึง127 ส่วนตัวเลขแบบ unsigned char มีค่าระหว่าง 0 ถึง 255 และถ้าเป็น signed int มีค่าระหว่าง -32,768 ถึง 32,767 ส่วนตัวเลขแบบ unsigned int มีค่าระหว่าง 0 ถึง 65,535

ค่าคงที่ (constants) คือค่าของสิ่งใดสิ่งหนึ่งที่กำหนดไว้ในโปรแกรมให้มีค่าคงตัวจนกว่าจะมีคำสั่งเปลี่ยนแปลงเป็นอย่างอื่นเขียนได้ 4 ลักษณะคือค่าคงที่จำนวนเต็ม (integer constants) เขียนอยู่ในรูปตัวอักษรอาจมีเครื่องหมายลบนำหน้า ค่าคงที่จำนวนจริง (floating constants) เขียนในรูปตัวเลขมีทศนิยม ค่าคงที่รหัสตัวอักษร (character constants) ตัวอักษรถูกจำในรูปแบบตัวเลขตามรหัส ASCII ค่าคงที่สายอักษร (string constants) การตั้งค่าคงที่สามารถทำได้ โดยมีกฎว่าห้ามใช้เครื่องหมายจุลภาค (,) และช่องว่าง สามารถใช้เครื่องหมายลบ (-) ได้ การตั้งค่าคงที่ไม่สามารถกำหนดค่าสูงสุดหรือต่ำสุดได้ขึ้นอยู่กับชนิดของข้อมูล ซึ่งต้องทำความเข้าใจทีละชนิดข้อมูล

- **ค่าคงที่จำนวนเต็ม** (integer constant) ค่าคงที่จำนวนเต็มสามารถระบุได้ด้วยเลขฐานสิบ ฐานแปดและฐานสิบหก ถ้านำหน้าด้วยเลข 0 (ศูนย์) จะหมายถึงเลขฐานแปด ถ้านำหน้าด้วย 0x หรือ 0X จะหมายถึงเลขฐาน 16 เช่น เลขฐานสิบ 31 เขียนเป็นเลขฐานแปด 037 ส่วนเขียนเป็นเลขฐานสิบหก 0xlf หรือ 0XlF โดยเราใช้ตัวแปร int ในการระบุค่าซึ่งจะมีช่วงตัวเลข อยู่ระหว่าง -32768 ถึง 32767 คือจะสามารถเป็นได้ทั้งค่าบวกและค่าลบ ซึ่งจะมีค่าขนาด 16 บิต ใช้เนื้อที่ในการเก็บ 2 ไบต์ ตัวอย่างตัวแปรชนิดนี้ เช่น 5, -10, 2534

- **ค่าคงที่แบบ** unsigned (unsigned constants) แสดงว่าเป็นตัวแปรที่เก็บค่าเป็นจำนวนเต็มแบบไม่คิดเครื่องหมาย (เป็นบวกเท่านั้น) มักจะใช้เป็นคำนำหน้าตัวแปร ตัวอย่างเช่น unsigned int ซึ่งค่าคงที่แบบ unsigned สามารถระบุได้โดยกำกับท้ายด้วยตัวอักษร u หรือ U

- **ค่าคงที่จำนวนเต็มแบบยาว** (long integer constants) เป็นตัวแปรที่เก็บค่าเป็นจำนวนเต็มที่มีจำนวนไบต์เป็น 2 เท่าของจำนวนเดิม (มักจะใช้เป็นคำนำหน้าตัวแปร เช่น long int)

ซึ่งสามารถระบุได้โดยค่าคงที่จำนวนเต็มแบบยาวต้องเขียนกำกับด้วยตัวอักษร l หรือ L ที่ข้างท้าย (ถ้าเป็นเครื่องหมาย ul หรือ UL กำกับข้างท้ายจะหมายถึง unsigned long)

- **ค่าคงที่จำนวนจริง** (floating constants) เป็นตัวแปรที่ใช้เก็บข้อมูลเลขทศนิยม ได้แก่ ข้อมูลชนิด float และ double โดยจะเก็บอยู่ในรูป a.b x 10e ใช้พื้นที่ในการเก็บ 4 ไบต์มีค่าระหว่าง 3.4E-38 ถึง 3.4E+38 หรือแสดงเป็นเลขทศนิยมไม่เกิน 6 ตำแหน่ง ตัวอย่างตัวแปรชนิดนี้ เช่น 10.625-6.67 (จำนวนทศนิยมในรูปเลขชี้กำลัง) เช่น 1.23456E+2 คือ $1.23456 \times 10^2 = 123.456$E ในที่นี้ใช้แทนความหมายว่า “คูณด้วย 10 ยกกำลัง”

- **ค่าคงที่รหัสตัวอักษร** (character constants) เป็นตัวแปรที่ใช้สำหรับเก็บข้อมูลที่เป็นตัวอักษรขนาด 1 ตัว โดยใช้เนื้อที่ในการเก็บ 1 ไบต์ ตัวอย่างตัวแปรชนิดนี้ เช่น ‘A’, ‘b’, ‘1’, ‘?’ การเก็บค่าของอักขระเหล่านี้ในหน่วยความจำจะเก็บเป็นรหัสตามแบบที่คอมพิวเตอร์นั้นใช้อยู่ เช่นเป็นรหัส EBCDIC, ASCII ในไมโครคอมพิวเตอร์ เก็บเป็นรหัส ASCII เช่น

‘A’ มีค่าเท่ากับ 65_{10}
‘a’ มีค่าเท่ากับ 97_{10}
‘5’ มีค่าเท่ากับ 53_{10}
‘\n’ มีค่าเท่ากับ 10_{10}

- **ค่าคงที่สายอักษร** (string constants) เป็นกลุ่มข้อมูลที่ประกอบไปด้วยอักขระที่เขียนเรียงกันไปใช้ตัวแปรชนิดอักขระในการเก็บข้อมูลต่อหนึ่งหน่วยตัวอักษร โดยจะมีเครื่องหมาย “ ” อัญประกาศ (double quote) ล้อมรอบอยู่เสมอ เช่น Hello World! ในภาษาซีต้องเขียนเป็น “Hello World!”

2.6.2 ตัวดำเนินการ (operator)

คือเครื่องหมายที่ใช้แทนการปฏิบัติการอย่างใดอย่างหนึ่งกับค่าข้อมูลหนึ่งหน่วยหรือมากกว่า ค่าข้อมูลนี้เรียกว่าตัวถูกดำเนินการ (operand) ซึ่งได้แก่ตัวแปร (variable) ค่าคงที่ (constant) นิพจน์ (expression) ตัวดำเนินการส่วนมากใช้ตัวถูกดำเนินการสองตัว แต่มีตัวดำเนินการบางชนิดที่กระทำกับข้อมูลหรือตัวถูกดำเนินการเพียงตัวเดียว ตัวดำเนินการในภาษาซีมีหลายประเภท ได้แก่ ตัวดำเนินการเลขคณิต (arithmetic operators) ตัวดำเนินการเอกภาพ (unary operators) ตัวดำเนินการสัมพันธ์และตรรกะ (relational and logical operators) ตัวดำเนินการกำหนดค่า (assignment operators) ตัวดำเนินการเงื่อนไข (conditional operators)

ตัวดำเนินการเลขคณิต (arithmetic operators) เป็นตัวดำเนินการเพื่อหาผลลัพธ์จากการคำนวณ ได้แก่ การบวก การลบ การคูณ การหาร และการหาร เอาผลลัพธ์เฉพาะเศษ สัญลักษณ์ตัวดำเนินการและตัวอย่างเมื่อ a และ b เป็นตัวถูกดำเนินการ แสดงได้ดังตารางที่ 2.5

ในภาษาซีไม่มีตัวดำเนินการแบบเลขยกกำลัง แต่เราสามารถเรียกใช้งานเลขยกกำลังผ่านไลบรารีฟังก์ชัน (pow) ได้ซึ่งจะได้กล่าวถึงต่อไป ตัวถูกดำเนินการที่ใช้กับตัวดำเนินการเลขคณิตนี้จะต้องมีค่าเป็นตัวเลข ได้แก่ ข้อมูลประเภทจำนวนเต็ม ทศนิยม (floating-point) และตัวอักขระ (character)

ตารางที่ 2.5 แสดงตัวดำเนินการเลขคณิต (arithmetic operators)

ตัวดำเนินการ	ความหมาย	ตัวอย่าง
+	การบวก (Addition)	a + b
-	การลบ (Subtraction)	a - b
*	การคูณ (Multiplication)	a * b
/	การหาร (Division)	a / b
%	การหาเศษจากการหาร (Modulus)	a % b

ตัวดำเนินการเชิงสัมพันธ์และตรรกะ (relational and logical operators)

- **ตัวดำเนินการเชิงสัมพันธ์** (relational operators) ตัวดำเนินการเชิงสัมพันธ์เป็นตัวดำเนินการที่ใช้แสดงความสัมพันธ์ระหว่างค่าของตัวแปร โดยถ้าความสัมพันธ์นั้นเป็นจริง (true) ผลที่ได้จะมีค่าเท่ากับ 1 แต่ถ้าความสัมพันธ์นั้นเป็นเท็จ (false) ผลที่ได้จะมีค่าเท่ากับ 0 ตัวดำเนินการเชิงสัมพันธ์มีทั้งหมด 4 ตัวได้แก่

< คือ น้อยกว่า

<= คือ น้อยกว่าหรือเท่ากับ

> คือ มากกว่า

>= คือ มากกว่าหรือเท่ากับ

ตัวดำเนินการทั้ง 4 ตัวนี้มีลำดับความสำคัญเท่ากันแต่ต่ำกว่าตัวดำเนินการเลขคณิต โดยมีลำดับการทำงานจากซ้ายไปขวา

- **ตัวดำเนินการเทียบเท่า** (equality operators) ตัวดำเนินการเทียบเท่าเป็นตัวดำเนินการที่ใกล้เคียงกับตัวดำเนินการเชิงสัมพันธ์ ตัวดำเนินการเทียบเท่ามี 2 ตัวได้แก่

= = เท่ากับ

!= ไม่เท่ากับ

ตัวดำเนินการเทียบเท่าทั้งสองตัวมีลำดับความสำคัญเท่ากัน แต่ต่ำกว่าตัวดำเนินการเชิงสัมพันธ์ และมีลำดับการทำงานจากซ้ายไปขวา ผลลัพธ์ที่ได้จากนิพจน์ถ้าเป็นจริงผลที่ได้มีค่าเท่ากับ 1 และถ้าเป็นเท็จ ผลที่ได้จะมีค่าเท่ากับ 0

- ตัวดำเนินการทางตรรกะ (logical operators) ตัวดำเนินการทางตรรกะเป็นตัวดำเนินการที่ใช้ในการเชื่อมนิพจน์ทางตรรกะ ให้มีความซับซ้อนขึ้นโดยผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ ถ้าเป็นจริงจะมีค่าเท่ากับ 1 และถ้าเป็นเท็จจะมีค่าเท่ากับ 0 ตัวดำเนินการทางตรรกะมีทั้งหมด 3 ตัวได้แก่

&& และ (and)

|| หรือ (or)

! ไม่ (not)

ผลลัพธ์ที่ได้จากการ “and” จะเป็นจริงในกรณีที่ตัวถูกดำเนินการทั้งสองเป็นจริงเท่านั้น ส่วนกรณีอื่น ๆ ผลลัพธ์ที่ได้จะเป็นเท็จ

ผลลัพธ์ที่ได้จากการ “or” จะเป็นจริง เมื่อตัวถูกดำเนินการตัวใดตัวหนึ่งหรือทั้งสองตัวเป็นจริง ส่วนกรณีอื่น ๆ ผลลัพธ์ที่ได้จะเป็นเท็จ

2.6.3 คำสั่งควบคุม (control statements)

ในการเขียนโปรแกรมภาษาซี แบ่งคำสั่งควบคุมออกเป็น 2 ประเภทใหญ่ ๆ คือ คำสั่งแบบทางเลือก และคำสั่งแบบวนซ้ำ มีรายละเอียดดังต่อไปนี้

คำสั่งดำเนินการแบบเลือก (Branching) คำสั่ง if เป็นคำสั่งดำเนินการที่ใช้ตรวจสอบผลลัพธ์ที่ได้จากนิพจน์ตรรกะ (boolean Expression) เพื่อเลือกทำคำสั่งหรือกลุ่มของคำสั่ง 3 รูปแบบดังนี้

1. คำสั่งดำเนินการแบบหนึ่งทางเลือก (if statement)
2. คำสั่งดำเนินการแบบสองทางเลือก (if-else statement)
3. คำสั่งดำเนินการแบบหลายทางเลือก (nested-if statement)

- คำสั่งดำเนินการแบบหนึ่งทางเลือก (if statement) เป็นคำสั่งที่ใช้เลือกทำคำสั่งหรือกลุ่มคำสั่ง เมื่อผลลัพธ์ที่ได้จากนิพจน์ตรรกะเป็นจริง ซึ่งมีรูปแบบของคำสั่งและรูปแบบการใช้งานดังนี้

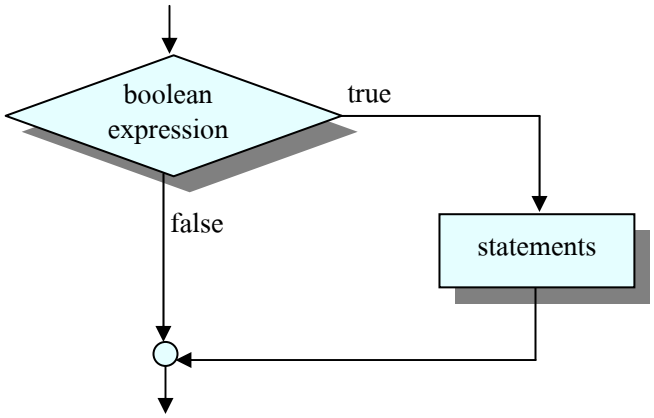
<u>รูปแบบ</u> if (boolean expression)

statement

<u>ตัวอย่าง</u> if (score >= 80)

ch = ‘A’;

โดยมีแผนผังโครงสร้างการดำเนินการดังรูปที่ 2.14



รูปที่ 2.14 แสดงโครงสร้างการดำเนินการของคำสั่งดำเนินการแบบหนึ่งทางเลือก

ขั้นตอนการทำงานของคำสั่งเริ่มจากการคำนวณผลลัพธ์ที่ได้จากนิพจน์ตรรกะ (boolean expression) กรณีผลลัพธ์ที่ได้เป็นจริง ก็จะดำเนินการคำสั่งหรือกลุ่มคำสั่งที่อยู่ถัดไป

```
1    #include <stdio.h>
2    void main() {
3            unsigned int your_number;
4            printf("Enter a whole number: ");
5            scanf("%d",&your_number);
6            if (your_number%2 == 0)
7                    printf("Your number is even\n");
8            if (your_number%2 != 0) {
9                    printf("Your number is odd.\n");
10           }
11           printf("That's all,folks!\n);
12   }
```

รูปที่ 2.15 แสดงตัวอย่างคำสั่งดำเนินการแบบหนึ่งทางเลือก

- **คำสั่งดำเนินการแบบสองทางเลือก** (if-else statement) เป็นคำสั่งที่ใช้เลือกทำคำสั่งหรือกลุ่มคำสั่งหนึ่ง เมื่อผลลัพธ์ที่ได้จากนิพจน์ตรรกะเป็นจริงหรือเลือกทำคำสั่งหรือกลุ่มของคำสั่งอีกคำสั่งหนึ่ง เมื่อเงื่อนไขเป็นเท็จ ซึ่งมีรูปแบบของคำสั่งและมีรูปแบบการใช้งานดังนี้

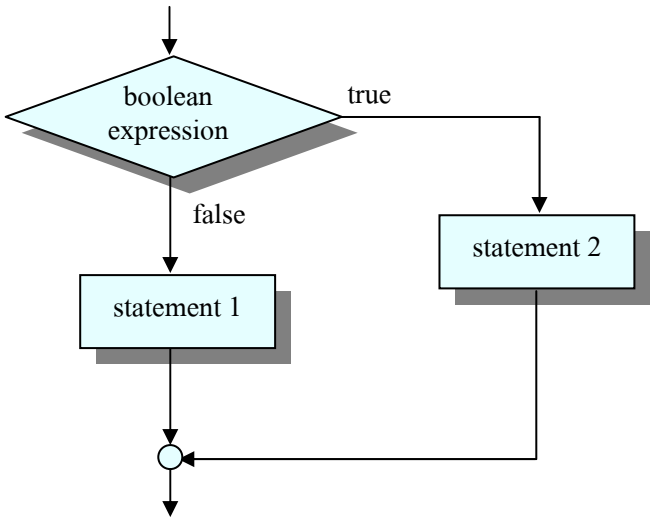
<u>รูปแบบ</u>

```
if (boolean expression)
        statement 1
else
        statement 2
```

<u>ตัวอย่าง</u>

```
if (score >= 50)
        ch = 'P';
else
        ch = 'F';
```

โดยมีแผนผังโครงสร้างการดำเนินการดังรูปที่ 2.16



รูปที่ 2.16 แสดงโครงสร้างการดำเนินการของคำสั่งดำเนินการแบบสองทางเลือก

ขั้นตอนการทำงานของคำสั่งเริ่มจากการคำนวณผลลัพธ์ที่ได้จากนิพจน์ตรรกะกรณีผลลัพธ์ที่ได้เป็นจริง จะดำเนินการคำสั่งหรือกลุ่มคำสั่งที่ 1 (statement 1) หรือกรณีที่ผลลัพธ์ที่ได้เป็นเท็จ จะดำเนินการคำสั่งหรือกลุ่มคำสั่งที่ 2 (statement 2)

```
1    void main() {
2            unsigned int your_number;
3            scanf("%d",&your_number);
4            if (your_number%2 == 0)
5                    printf("Your number is even\n");
6            else {
7                    printf("Your number is odd.\n");
8            }
9            printf("That's all,folks!\n");
10   }
```

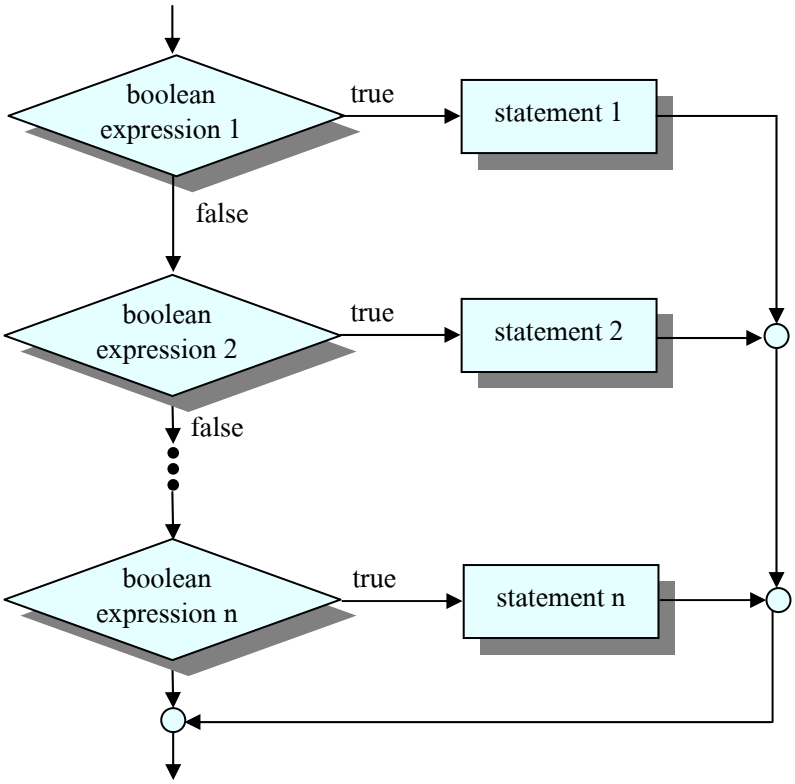
รูปที่ 2.17 แสดงตัวอย่างคำสั่งดำเนินการแบบสองทางเลือก

- **คำสั่งดำเนินการแบบหลายทางเลือก** (nested-if statement) เป็นการเขียนคำสั่งแบบทางเลือก ทั้งทางเลือกเดียวหรือสองทางเลือก เพื่อใช้เลือกดำเนินการกับคำสั่งใดคำสั่งหนึ่ง ซึ่งมีรูปแบบของคำสั่งดังนี้

<u>รูปแบบ</u>
```
if (boolean expression 1)
        statement 1
else if (boolean expression 2)
        statement 2
…
else
        statements n
```

<u>ตัวอย่าง</u>
```
if (x == 1)
        ch = 'A';
else if (x == 2)
        ch = 'B';
…
else
        ch = 'Z';
```

โดยมีแผนผังโครงสร้างการดำเนินการดังรูปที่ 2.18



รูปที่ 2.18 แสดงโครงสร้างการดำเนินการของคำสั่งดำเนินการแบบหลายทางเลือก

ลักษณะการทำงานของคำสั่งเป็นการเขียนคำสั่งแบบเลือกซ้อนกัน ซึ่งขั้นตอนการทำงานของคำสั่งเริ่มจากการคำนวณผลลัพธ์ที่ได้จากนิพจน์ตรรกะที่ 1 (boolean expression 1) กรณีผลลัพธ์ที่ได้เป็นจริง จะดำเนินการคำสั่งหรือกลุ่มคำสั่งที่ 1 (statement 1) หรือกรณีผลลัพธ์ที่ได้เป็นเท็จ จะดำเนินการตรวจสอบผลลัพธ์ที่ได้จากนิพจน์ตรรกะที่ 2 (boolean expression 2) กรณีผลลัพธ์ที่ได้เป็นจริง จะดำเนินการกับคำสั่งหรือกลุ่มคำสั่งที่ 2 (statement 2) หรือกรณีผลลัพธ์เป็นเท็จ จะดำเนินการตรวจสอบผลลัพธ์ที่ได้จากนิพจน์ตรรกะที่ซ้อนภายในถัดไปเรื่อย ๆ และในกรณีที่ผลลัพธ์ที่ได้จากนิพจน์ตรรกะในลำดับถัดไปไม่มีคำตอบใดเป็นจริง ก็จะดำเนินการกับคำสั่งหรือกลุ่มคำสั่งที่ n (statement n)

```
1    void main() {
2            char cmd;
3            printf("Chart desired : Pie Bar Scatter Line Three-D\n");
4            printf("Press first letter of the Chart you want: ");
5            scanf("%c",&cmd);
6            if (cmd=='p')
7                    printf("Doing pie chart\n");
8            else if (cmd == 'b')
9                    printf("Doing bar chart\n");
10           else if (cmd == 's')
11                   printf("Doing scatter chart\n");
12           else if (cmd == 'l')
13                   printf("Doing line chart\n");
14           else if (cmd == 't')
15                   printf("Doing 3-D chart\n");
16           else printf("Invalid choice.\n");
17   }
```

รูปที่ 2.19 แสดงตัวอย่างคำสั่งดำเนินการแบบหลายทางเลือก

นอกจากคำสั่งเลือกแบ if statement แล้วยังมีคำสั่งดำเนินการแบบหลายทางเลือกอีกหนึ่งรูปแบบคือ case statement ซึ่งมีรูปแบบการทำงานดังนี้

<u>รูปแบบ</u> switch (expression) {

case label 1 : statement 1

break;

case label 2 : statement 2

break;

case label n : statement n

break;

ตัวอย่าง

```
switch (n) {
        case 1  : printf("one"); break;
        case 2  : printf("two"); break;
        default : printf("Other number");
    }
```

ขั้นตอนการทำงานจะเริ่มจากการคำนวณนิพจน์ (expression) และนำไปเปรียบเทียบกับ label ในลำดับต่าง ๆ กรณีที่ผลลัพธ์ของนิพจน์ไปมีค่าตรงกับ label ใดก็จะดำเนินการคำสั่งหรือกลุ่มคำสั่งที่อยู่หลังเครื่องหมาย ":" ของบรรทัดคำสั่งนั้นดังตัวอย่างถ้า n มีค่าเท่ากับ 1 ก็จะพิมพ์ข้อความ one บนจอภาพ ถ้าตรงกับ 2 ก็จะพิมพ์ข้อความ two หรือถ้าไม่ตรงกับค่าใดเลยก็จะพิมพ์ข้อความ Other number

```
1   #include <stdio.h>
2   #include <conio.h>
3   void main() {
4           char cmd;
5           printf("Chart desired : Pie bar scatter Line Three-D\n");
6           printf("Press first letter of the chart you want: ");
7           scanf("%c",&cmd);
8          switch (cmd) {
9                   case 'p' : printf("Doing pie chart\n"); break;
10                  case 'b' : printf("Doing bar chart\n"); break;
11                  case 's' : printf("Doing scatter chart\n"); break;
12                  case 'l' : printf("Doing line chart\n"); break;
13                  case 't' : printf("Doing 3-D chart\n"); break;
14                  default : printf("Invalid choice.\n");
15          }
16  }
```

รูปที่ 2.20 แสดงตัวอย่างคำสั่งดำเนินการแบบหลายทางเลือกด้วย case statement

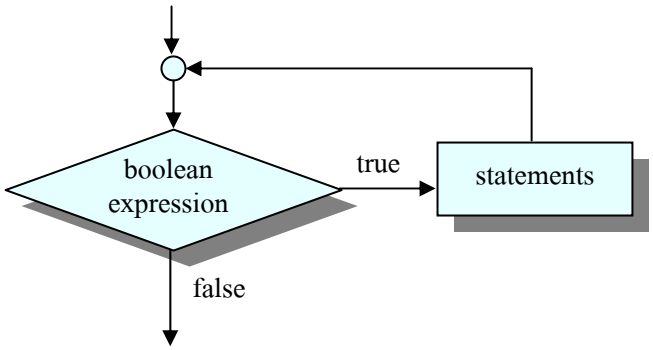
คำสั่งดำเนินการแบบวนซ้ำ (Looping) ในการเขียนโปรแกรมบางครั้งผู้เขียนอาจต้องแก้ปัญหาที่ซับซ้อนหรือต้องทำชุดคำสั่งที่เป็นกิจกรรมเดียวกันหลายครั้งซึ่งในภาษาโปรแกรมเราเรียกว่า ลูป (loop) ในภาษาซีมีชุดคำสั่งที่ใช้ในการควบคุมการทำงานลักษณะดังกล่าวดังนี้

- คำสั่งวนซ้ำแบบ while (while statement) คุณสมบัติของการวนซ้ำด้วยคำสั่ง "while" จะใช้เมื่อผู้เขียนโปรแกรมต้องการตรวจสอบเงื่อนไขหรือนิพจน์ (expression) ก่อนทำชุดคำสั่ง (statement) ในกรณีที่เงื่อนไขเป็นจริง (นิพจน์มีค่าไม่เท่ากับศูนย์) จึงทำชุดคำสั่งเมื่อทำชุดคำสั่งจบแล้วจะกลับไปตรวจสอบเงื่อนไขจนกว่าเงื่อนไขจะเป็นเท็จ จึงจะหยุดการทำงานในลูป ถ้าชุดคำสั่งมีมากกว่า 1 คำสั่งจะต้องเขียนชุดคำสั่งเหล่านั้นอยู่ภายในเครื่องหมาย { } มีรูปแบบการใช้งานดังนี้

<u>รูปแบบ</u> while (expression)
statement ;

<u>หรือ</u> while (expression)
{
statements ;
}

โดยมีแผนผังโครงสร้างการดำเนินการดังรูปที่ 2.21



รูปที่ 2.21 แสดงโครงสร้างการดำเนินการของคำสั่งวนซ้ำแบบ while

```
1   #include <stdio.h>
2   main() {
3           int digit = 0;
4           while (digit <= 9)
5                   printf("%d\n", digit++);
6   }
```

รูปที่ 2.22 แสดงตัวอย่างการใช้คำสั่งวนซ้ำแบบ while

- **คำสั่งวนซ้ำแบบ** do-while (do-while statement) การใช้งาน do-while จะใช้เมื่อผู้เขียนโปรแกรมต้องการทำชุดคำสั่งที่ต้องการทำซ้ำก่อนแล้วจึงตรวจสอบเงื่อนไข ถ้าเงื่อนไขเป็นจริงจะทำชุดคำสั่งที่อยู่หลังข้อความ do อีกครั้งและจบการทำชุดคำสั่งวนซ้ำเมื่อเงื่อนไขหลังข้อความ while เป็นเท็จ สังเกตว่าการใช้คำสั่งวนซ้ำแบบ do-while นั้นจะมีการทำคำสั่ง (statement) ภายในลูปอย่างน้อย 1 ครั้ง กรณีชุดคำสั่งมีมากกว่า 1 ประโยคผู้เขียนจะต้องเขียนอยู่ภายในเครื่องหมาย { } มีรูปแบบการใช้งานดังนี้

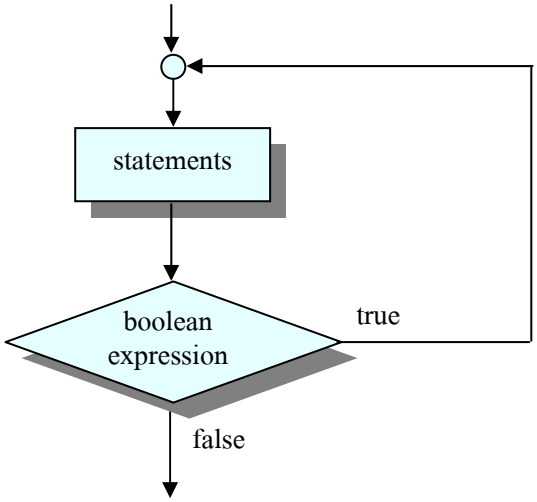
<u>รูปแบบ</u>

```
do
    statement;
while (expression );
```

<u>หรือ</u>

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

โดยมีแผนผังโครงสร้างการดำเนินการดังรูปที่ 2.23



รูปที่ 2.23 แสดงโครงสร้างการดำเนินการของคำสั่งวนซ้ำแบบ do-while

```
1   #include <stdio.h>
2   main()
3   {
4           int digit = 0;
5           do
6                   printf("%d\n", digit++);
7   while (digit <= 9)
8   }
```

รูปที่ 2.24 แสดงตัวอย่างการใช้คำสั่งวนซ้ำแบบ do-while

- **คำสั่งวนซ้ำแบบ** for (for statement) การวนซ้ำด้วยชุดคำสั่ง for รูปแบบจะคล้ายกับ while แต่ for มักใช้การวนซ้ำกับเงื่อนไขที่เกิดจากการเปลี่ยนแปลงค่าของตัวแปรที่กำหนดไว้หรือกำหนดจำนวนรอบในการทำซ้ำด้วยการเปลี่ยนค่าตัวแปรที่ตั้งไว้ มีรูปแบบดังนี้

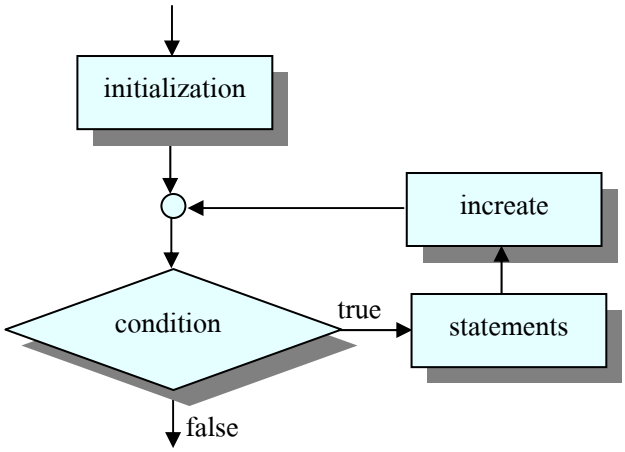
<u>รูปแบบ</u>

```
for(initialization ; condition ; increate/decrease)
    statement ;
```

หรือ

```
for(initialization; condition; increate/decrease)
 {
      statements;
 }
```

โดยมีแผนผังโครงสร้างการดำเนินการดังรูปที่ 2.25



รูปที่ 2.25 แสดงโครงสร้างการดำเนินการของคำสั่งวนซ้ำแบบ for

รูปแบบจะเขียนชุดคำสั่งตรวจสอบการวนซ้ำไว้ภายในเครื่องหมาย ( ) หลังข้อความ for การเขียนชุดคำสั่งตรวจสอบภายในวงเล็บจะแบ่งเป็น 3 ส่วนโดยแต่ละส่วนจะคั่นด้วยเครื่องหมาย semicolon (;) แต่ละส่วนมีความหมายดังนี้

initialization ใช้กำหนดค่าเริ่มต้นให้กับตัวแปรที่ใช้ควบคุมการวนรอบ เช่น i = 1; การทำงานจะทำส่วนนี้เพียงครั้งแรกเพียงครั้งเดียวเท่านั้น กรณีกำหนดค่าตัวแปรมากกว่า 1 ตัวแปรต้องใช้เครื่องหมาย comma (,) คั่น เช่น a = 2, b = 5;

condition เงื่อนไขตรวจสอบการเปลี่ยนแปลงค่าของตัวแปรที่อยู่ใน initialization เช่น i <= 5; หรือ (a<10) and (b < 5); กรณีเงื่อนไขได้ผลลัพธ์เป็นจริงจะทำชุดคำสั่งซึ่งอยู่ภายในเครื่องหมาย { } ทั้งหมด

increate/decrease ชุดคำสั่งเปลี่ยนแปลงค่าของตัวแปรในส่วน initialization เช่น i++ หรือ a++, b++ เมื่อทำชุดคำสั่งนี้เสร็จแล้วจะส่งขั้นตอนการทำงานไปที่ส่วน condition อีกครั้ง ถ้าเงื่อนไขเป็นเท็จ จะทำชุดคำสั่งหลังเครื่องหมาย } ต่อไป

โดยทั่วไปการใช้งานคำสั่งวนรอบแบบ for จะเหมาะกับการเขียนโปรแกรมที่ทราบจำนวนรอบที่ต้องการทำคำสั่งซ้ำที่แน่นอน

```
1    #include <stdio.h>
2    main()
3    {
4            int digit;
5            for (digit = 0; digit <= 9; ++digit)
6                    printf("%d\n", digit);
7    }
```

รูปที่ 2.26 แสดงตัวอย่างการใช้คำสั่งวนซ้ำแบบ for

บทที่ 3
การออกแบบและพัฒนาเครื่องมือสร้างกราฟควบคุมกระแส และข้อมูลทดสอบ สำหรับภาษาซี

งานวิจัยนี้มุ่งเน้นที่จะพัฒนาเครื่องมือเพื่อสนับสนุนการทดสอบซอฟต์แวร์ด้วยเทคนิคแบบไวท์บอกซ์ เพื่อเป็นการอำนวยความสะดวกให้กับผู้ทดสอบให้สามารถทำการทดสอบซอฟต์แวร์ได้สะดวกยิ่งขึ้น กล่าวคืองานวิจัยนี้จะพัฒนาเครื่องมือที่สามารถรับซอร์สโค้ดที่พัฒนาด้วยภาษาซีเข้าสู่เครื่องมือแล้วเครื่องมือจะนำซอร์สโค้ดนั้นมาแยกส่วนพิจารณาเพื่อให้สามารถนำเสนอออกมาในรูปแบบของกราฟควบคุมกระแสที่สามารถแสดงให้เห็นผังการทำงานของซอร์สโค้ด จากนั้นเครื่องมือต้องสามารถแสดงเส้นทางอิสระของโปรแกรมเพื่อให้เห็นเส้นทางสำหรับทดสอบและเครื่องมือยังต้องสามารถสร้างข้อมูลทดสอบที่สอดคล้องกับเส้นทางเพื่อเตรียมไว้ให้สำหรับผู้ทดสอบจะได้นำไปใช้ในการทดสอบซอร์สโค้ดนั้นต่อไป โดยเครื่องมือที่พัฒนาจะมีส่วนหน้าจอที่ผู้ใช้สามารถใช้งานได้สะดวกและนำเสนอข้อมูลต่าง ๆ ออกมาในรูปแบบที่น่าสนใจและในการประมวลผลจะอาศัยฐานข้อมูล (database) ที่เกี่ยวข้องเกี่ยวกับโปรแกรมภาษาซีที่ต้องการจะทดสอบ ฉะนั้นจึงต้องมีส่วนที่ทำหน้าที่ในการจัดเก็บข้อมูลเหล่านี้ไว้ด้วย

3.1 องค์ประกอบหลักของเครื่องมือ

แนวคิดในการสร้างเครื่องมือเพื่อทดสอบโปรแกรมที่พัฒนาด้วยภาษาซีได้แบ่งส่วนของงานวิจัยออกเป็น 2 ส่วนใหญ่ ๆ คือส่วนของการพัฒนาเครื่องมือ และส่วนของการจัดเก็บข้อมูลที่เกี่ยวข้องสำหรับการประมวลผล ซึ่งแต่ละส่วนมีรายละเอียดดังต่อไปนี้

3.1.1 ส่วนของการพัฒนาเครื่องมือ

ผู้วิจัยได้ทำการพัฒนาเครื่องมือนี้ด้วยโปรแกรมวิชวลเบสิก 6.0 (Microsoft Visual Basic 6.0) เนื่องจากเป็นภาษาคอมพิวเตอร์ที่สามารถแสดงคำสั่งและนำเสนอในรูปแบบรูปภาพ (graphic user interface: GUI) ที่น่าสนใจได้ อีกทั้งยังมีรูปแบบและหลักการใช้งานไม่ยุ่งยาก สำหรับแนวทางในการพัฒนาเครื่องมือนี้ผู้วิจัยได้แบ่งโครงสร้างของโปรแกรมเป็นส่วนย่อย ๆ ออกเป็น 5 ส่วนดังนี้

1. ส่วนนำเข้าซอร์สโค้ดและใช้ติดต่อกับผู้ใช้ (User Interface)
2. ส่วนกระจายนิพจน์ (Parser Section)

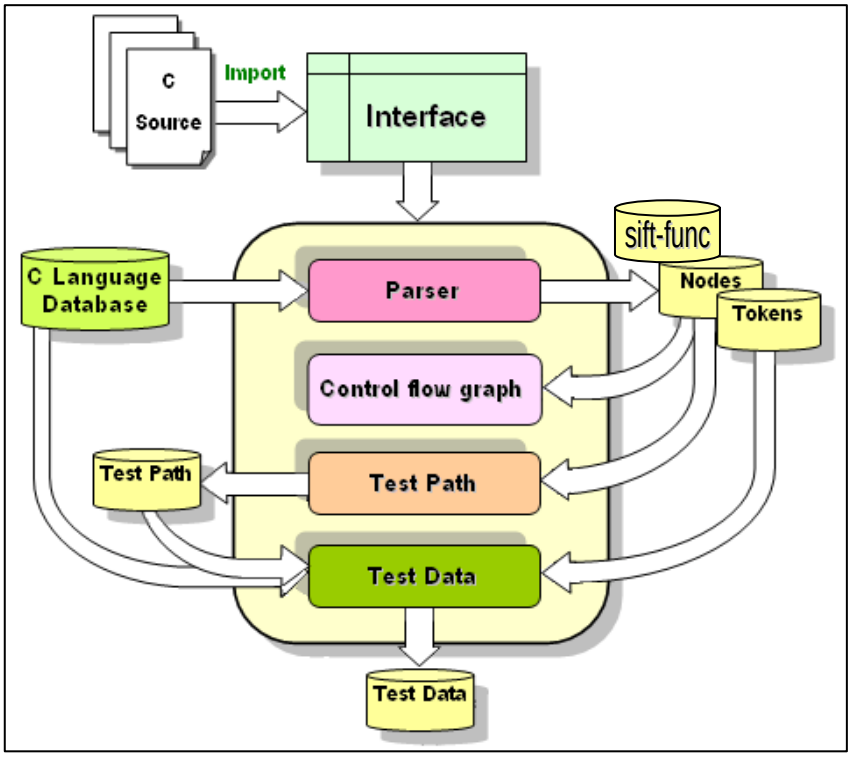
3. ส่วนสร้างกราฟควบคุมกระแส (Control flow graph Section)
4. ส่วนสร้างเส้นทางทดสอบ (Test path Section)
5. ส่วนสร้างข้อมูลทดสอบ (Test data Section)

3.1.2 ส่วนการจัดเก็บข้อมูล

ในการประมวลผล เพื่อให้แต่ละส่วนได้ผลลัพธ์ถูกต้องอย่างที่ต้องการ จำเป็นต้องอาศัยข้อมูลบางส่วนเพื่อช่วยในการตัดสินใจ และนอกจากนี้ยังต้องเก็บข้อมูลที่ได้จากการประมวลผลบางส่วนไว้เพื่อการประมวลผลในส่วนอื่น ๆ ดังนั้นจึงแบ่งส่วนของการจัดเก็บข้อมูลเป็น 2 ส่วนย่อย ๆ ดังต่อไปนี้

ฐานข้อมูล เก็บคำสั่งและเครื่องหมายต่าง ๆ ในภาษาซีจัดเก็บ โดยใช้ระบบจัดการฐานข้อมูลไมโครซอฟต์แอคเซส (Microsoft Access)

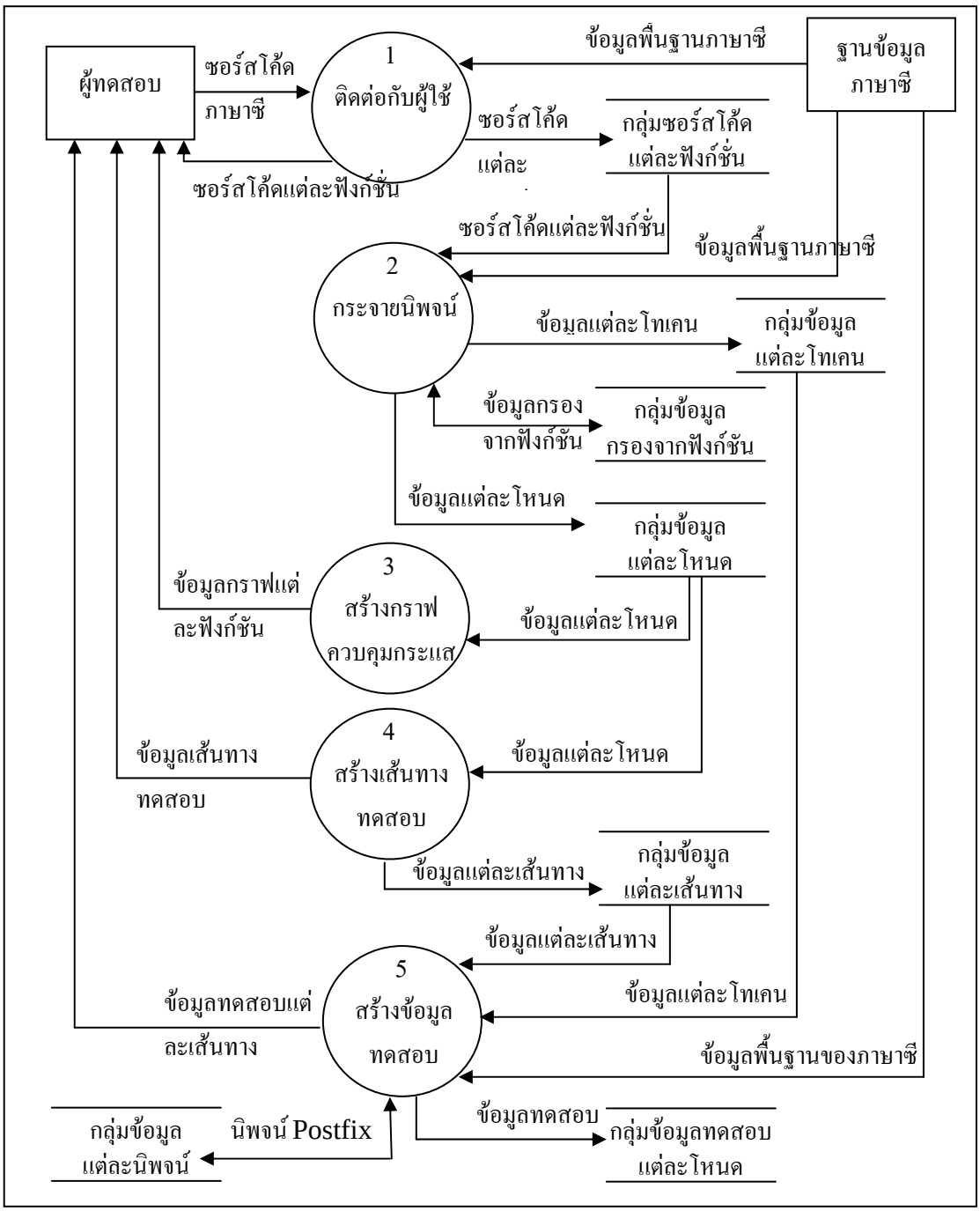
ส่วนเก็บข้อมูลระหว่างประมวลผล คือส่วนที่ใช้เก็บข้อมูลขณะประมวลผล ซึ่งมีการจัดเก็บข้อมูลแบบแถวลำดับ (array) ซึ่งมีการจัดเก็บโหนด (nodes) โทเคน (tokens) เส้นทางทดสอบ (test path) และ ข้อมูลทดสอบ (test data)



รูปที่ 3.1 แสดงองค์ประกอบของการพัฒนาเครื่องมือ

3.2 รายละเอียดของขั้นตอนการทำงานของเครื่องมือ

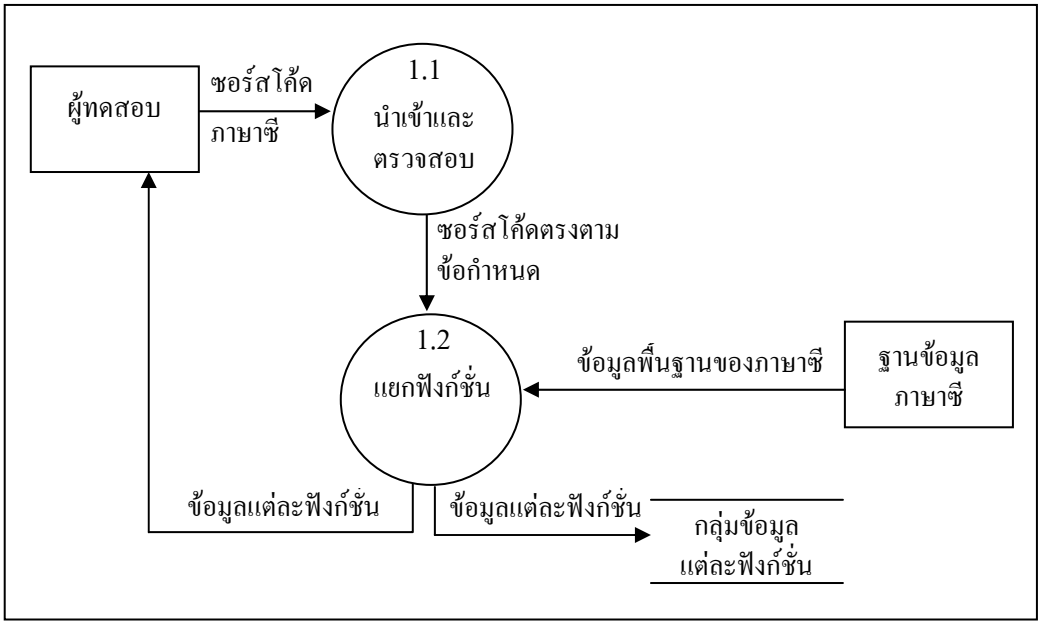
แนวคิดในการพัฒนาเครื่องมือได้มีการแบ่งโครงสร้างของเครื่องมือออกเป็น 5 ส่วนย่อย ๆ ดังรูปที่ 3.1 แต่ละส่วนมีการทำงานและการไหลของข้อมูลระหว่างกัน แสดงได้ดังรูปที่ 3.2



รูปที่ 3.2 แสดงแผนภาพกระแสข้อมูลระดับ 0 ของเครื่องมือ

3.2.1 ส่วนติดต่อกับผู้ใช้ (User Interface)

เป็นส่วนที่ใช้สำหรับติดต่อกับผู้ใช้ในการเรียกใช้งานส่วนต่าง ๆ ของเครื่องมือ รวมถึงหน้าที่หลักคือนำเข้าซอร์สโค้ดที่พัฒนาด้วยภาษาซีพื้นฐาน โดยกำหนดให้สามารถนำเข้าได้เฉพาะซอร์สโค้ดที่มีนามสกุลหรือประเภทของไฟล์ (extension) เป็น C และต้องมีฟังก์ชันย่อย ๆ (user-defined function) ที่สร้าง ภายในไม่เกิน 24 ฟังก์ชันเท่านั้น และนอกจากนี้ซอร์สโค้ดที่นำเข้าต้องผ่านการตรวจสอบการใช้ไวยากรณ์ (syntax) และหลักการใช้คำสั่งต่าง ๆ หรือผ่านการตรวจสอบด้วยตัวแปลภาษา (compiler) ให้มีความถูกต้องก่อน เนื่องจากเครื่องมือนี้จะไม่ตรวจสอบในลักษณะนี้ซ้ำอีกครั้งแต่จะเน้นในเรื่องความสัมพันธ์กันของคำสั่งที่มีผลต่อเส้นทางการทำงานของโปรแกรม ถ้าซอร์สโค้ดที่นำเข้าไม่เป็นไปตามข้อกำหนดผลลัพธ์ที่ได้อาจมีความคลาดเคลื่อนจากรูปแบบที่ควรจะเป็นได้ สำหรับการทำงานของส่วนนำเข้าซอร์สโค้ดมีรายละเอียดการทำงานดังรูปที่ 3.3



รูปที่ 3.3 แสดงแผนภาพกระแสข้อมูลระดับ 1 ของส่วนติดต่อผู้ใช้

นำเข้าและตรวจสอบซอร์สโค้ด เป็นส่วนที่ทำหน้าที่ในการรับซอร์สโค้ดที่ต้องการตรวจสอบโดยผ่านเมนูนำเข้า (import) และเลือกหาซอร์สโค้ดที่ต้องการและเมื่อเลือกซอร์สโค้ดแล้วซอร์สโค้ดที่เลือกจะแสดงที่หน้าแรกของเครื่องมือ ซึ่งขณะที่ยังไม่มีซอร์สโค้ดนำเข้าส่วนอื่น ๆ ก็จะยังใช้งานไม่ได้และเมื่อนำเข้าซอร์สโค้ดแล้ว ส่วนแยกฟังก์ชันจะทำงานต่อโดยอัตโนมัติ

แยกฟังก์ชัน เมื่อซอร์สโค้ดถูกนำเข้าและแสดงที่หน้าแรกแล้วการทำงานต่อมาคือเครื่องมือจะทำการตรวจสอบและทำการแยกซอร์สโค้ดออกเป็นฟังก์ชันย่อย ๆ ตามที่มีในซอร์สโค้ด ซึ่งจะต้องไม่เกินกว่า 24 ฟังก์ชันและในการแยกฟังก์ชันนี้จะมีการดึงข้อมูลพื้นฐานที่เกี่ยวข้องกับภาษาซีมาใช้ในการตรวจสอบเงื่อนไขเพื่อประมวลผลแยกฟังก์ชัน เมื่อแยกฟังก์ชันได้สำเร็จแล้วจะเก็บข้อมูลแต่ละฟังก์ชันไว้ในส่วนเก็บข้อมูลแบบแถวลำดับและขณะเดียวกันก็นำซอร์สโค้ดแต่ละฟังก์ชันที่แยกไปแสดงในส่วนแสดงผลแต่ละฟังก์ชันเพื่อให้ผู้ทดสอบได้เห็นผลลัพธ์ที่ได้

ในระหว่างประมวลผลส่วนนี้มีการจัดเก็บข้อมูลแต่ละฟังก์ชันแบบแถวลำดับชนิดข้อความ (string) มีการออกแบบโครงสร้างเพื่อจัดเก็บส่วนแยกฟังก์ชันโดยมีการเก็บส่วนหัว (header) และเก็บส่วนตัวของฟังก์ชัน (body)

- **เก็บส่วนหัว** คือส่วนที่ใช้เก็บซอร์สโค้ดที่ไม่อยู่ในขอบเขตของฟังก์ชัน โดยพิจารณาเฉพาะที่อยู่ก่อนหน้าแต่ละฟังก์ชันมีการเก็บเป็นแถวลำดับ 1 มิติดังนี้

StrHeader(index-func)

- **เก็บส่วนตัวของฟังก์ชัน** คือส่วนที่ใช้เก็บซอร์สโค้ดแต่ละฟังก์ชัน แบ่งออกเป็นฟังก์ชันแต่ละฟังก์ชัน ซึ่งเก็บเป็นแถวลำดับ 1 มิติเช่นกัน ดังนี้

StrFunc(index-func)

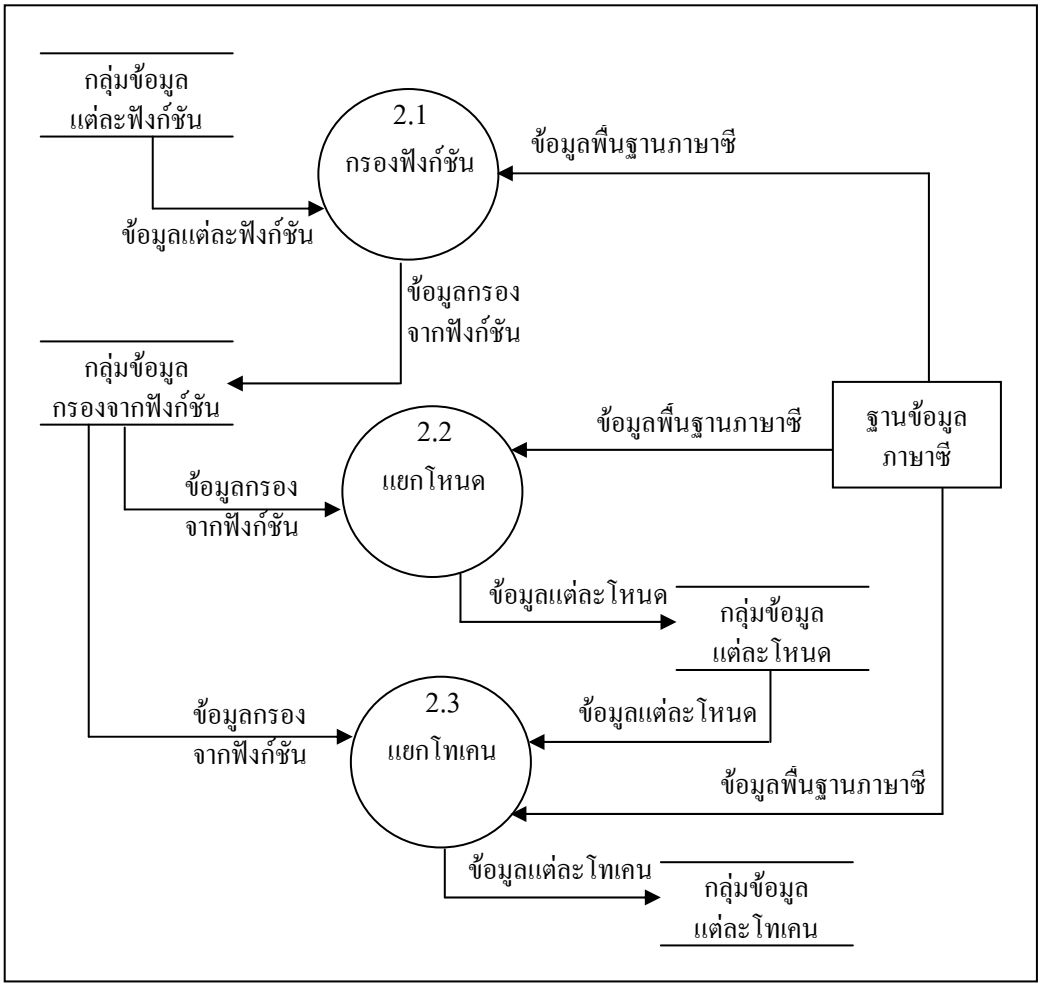
- **index-func** คือดัชนีหรือลำดับของฟังก์ชันที่มีการแยกออกได้ซึ่งกำหนดให้มีค่าตั้งแต่ 1-24 ข้อมูลนี้จะถูกเก็บไว้จนกว่าผู้ทดสอบ จะนำเข้าซอร์สโค้ดตัวใหม่

สำหรับส่วนที่ใช้แสดงผลซอร์สโค้ดแต่ละฟังก์ชันที่แยกออกมานั้นมีการแสดงผลด้วยวัตถุที่มีชนิดเป็น “RichTextBox” โดย “RichTextBox” นี้กำหนดให้มีลักษณะเป็นแถวลำดับเช่นเดียวกันและอาศัย index-func ค่าเดียวกันในการอ้างอิงข้อมูลเพื่อแสดงผล ในการกำหนดค่าทำได้ดังนี้

txt_func(index-func).text = StrFunc(index-func)

3.2.2 ส่วนกระจายนิพจน์ (Parser)

ซอร์สโค้ดที่นำเข้าเมื่อถูกแยกออกเป็นฟังก์ชันย่อยแต่ละฟังก์ชันแล้ว จากนั้นแต่ละฟังก์ชันจะถูกนำไปวิเคราะห์แยกเป็นส่วนย่อย ๆ ด้วยส่วนที่ทำหน้าที่กระจายนิพจน์ โดยจะพิจารณาเฉพาะส่วนที่เกี่ยวข้องกับสิ่งที่เครื่องมือจะสร้าง นั่นคือส่วนที่เป็นโหนดแต่ละโหนดที่จะสร้างเป็นกราฟควบคุมกระแสและส่วนย่อยที่สุดหรือโทเคนแต่ละตัวที่เกี่ยวข้องกับการสร้างข้อมูลทดสอบ ซึ่งอาจต้องตรวจสอบจากโหนดหรือส่วนอื่น ๆ ในฟังก์ชันนั้น ๆ เพื่อให้ส่วนต่าง ๆ ที่กล่าวมาข้างต้นทำงานง่ายขึ้น จึงออกแบบให้แยกส่วนในฟังก์ชัน หรือเรียกว่าส่วนกรอง (filter) ฟังก์ชันเพิ่มเติมเพื่อกรองเอาบางส่วนที่ไม่เกี่ยวข้องออกไป เหลือไว้แต่สิ่งที่จำเป็นต่อการประมวลผล แยกโหนด แยกโทเคนและสร้างข้อมูลทดสอบ ดังนั้นส่วนกระจายนิพจน์นี้มีหน้าที่หลักและขั้นตอนการทำงาน ดังรูปที่ 3.4



รูปที่ 3.4 แสดงแผนภาพกระแสข้อมูลระดับ 1 ของส่วนกระจายนิพจน์

กรองฟังก์ชัน ก่อนที่ซอร์สโค้ดจะถูกแยกโหนดเพื่อให้การทำงานขั้นอื่น ๆ ง่ายขึ้น จึงออกแบบให้มีส่วนกรองฟังก์ชัน เพื่อกรองเอาบางส่วนที่ไม่เกี่ยวข้องออกไป เหลือไว้แต่สิ่งที่จำเป็นต่อการประมวลผลแยกโหนด แยกโทเคนและสร้างข้อมูลทดสอบ ซึ่งส่วนสำคัญที่กล่าวมานั้นประกอบไปด้วย

- ชื่อฟังก์ชัน
- เครื่องหมายกำหนดขอบเขต (block) ของฟังก์ชันหรือขอบเขตของส่วนอื่น ๆ ซึ่งในที่นี้หมายถึง “{” และ “}”
- การประกาศตัวแปรที่เป็นพื้นฐานนั่นคือ int, char, float, double ซึ่งครอบคลุมถึงเครื่องหมายของชนิดเหล่านี้นั่นคือ unsigned และ signed
- คำสั่งควบคุมต่าง ๆ นั่นคือ if, for, while, do-while, switch-case
- คำสั่งที่อยู่ภายใต้เงื่อนไขของคำสั่งควบคุม ทั้งกรณีจริง และ เท็จ

องค์ประกอบต่าง ๆ นี้จะถูกออกแบบเก็บไว้ในส่วนเก็บข้อมูลแบบแถวลำดับ ซึ่งจะมีการจัดเก็บโดยนับเป็นจำนวนบรรทัด (lines) และกำหนดชนิดของแต่ละบรรทัดเพื่อให้นำไปใช้ได้ง่ายขึ้น ดังนี้

- **เก็บซอร์สโค้ด** ของแต่ละบรรทัดใช้สำหรับเก็บเนื้อหาของซอร์สโค้ดในบรรทัดลำดับที่กำลังประมวลผล ลำดับที่บรรทัดในที่นี้จะเป็นลำดับของเนื้อหาที่เก็บไม่ใช่ลำดับของบรรทัดจริงที่อยู่ในซอร์สโค้ด มีการจัดเก็บโครงสร้างเป็นแถวลำดับ 2 มิติ ดังนี้

Line2Node(index-func , index-line)

- **เก็บลำดับขอบเขต** ของแต่ละบรรทัดใช้เก็บลำดับบล็อก ที่บรรทัดนั้น ๆ อยู่ในภาษาซีมีการกำหนดขอบเขตของการใช้ตัวแปร หมายถึงตัวแปรที่ประกาศสามารถถูกนำไปใช้ในแต่ละส่วนของโปรแกรมได้ไม่เหมือนกัน จึงต้องเก็บข้อมูลนี้ไว้ ซึ่งมีโครงสร้างเป็นแถวลำดับ 2 มิติ ดังนี้

BofLine(index-func , index-line)

- **เก็บชนิดของแต่ละบรรทัด** เพื่อให้แยกแต่ละบรรทัดที่เก็บให้มีความแตกต่างกัน เพื่อจะง่ายต่อการนำมาใช้จะมีการเก็บชนิดของบรรทัดโดยแยกเป็น ชื่อฟังก์ชัน คำสั่งควบคุม เปิดบล็อก (“{”) ปิดบล็อก (“}”) หรืออื่น ๆ ที่เกี่ยวข้องเท่านั้น

TypeLine(index-func , index-line)

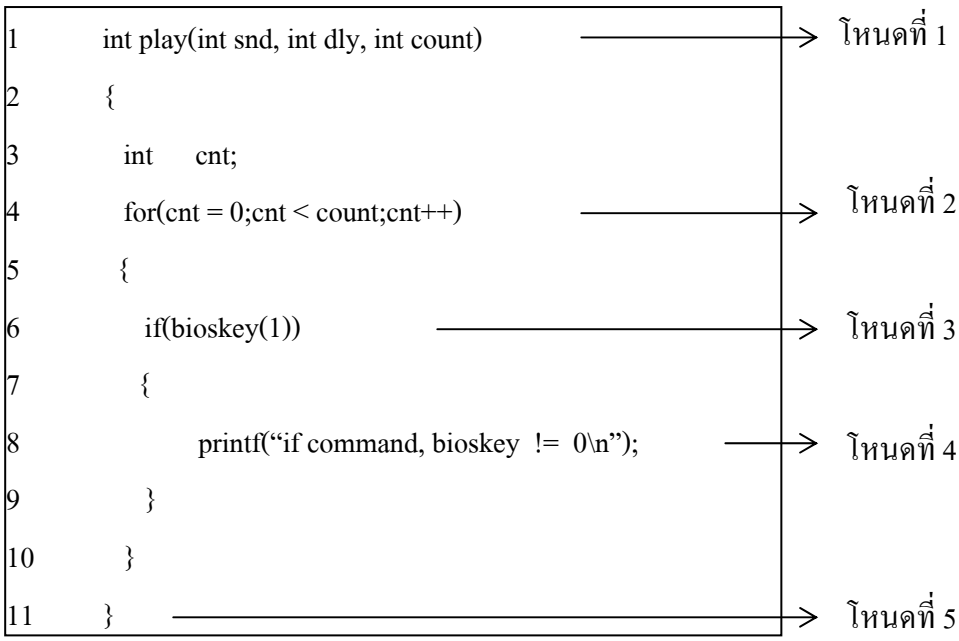
- **index-func** คือดัชนีหรือลำดับของฟังก์ชันที่ดำเนินการกรองฟังก์ชัน โดยได้กำหนดให้มีค่าตั้งแต่ 1-24 ของแต่ละซอร์สโค้ด

- **index-line** คือดัชนีหรือลำดับของบรรทัดที่กรองออกมาได้ โดยกำหนดให้มีค่าตั้งแต่ 1-100 ของแต่ละฟังก์ชัน

รายละเอียดต่าง ๆ แสดงดังรูปที่ 3.5 ซึ่งมีการจัดเก็บดังนี้ ซอร์สโค้ดบรรทัดที่ 1 แสดงชื่อฟังก์ชัน ซอร์สโค้ดบรรทัดที่ 2, 5 และ 8 แสดงเปิดบล็อก ซอร์สโค้ดบรรทัดที่ 11, 12 และ 15 แสดงเครื่องหมายปิดบล็อก ซอร์สโค้ดบรรทัดที่ 3 แสดงการประกาศตัวแปร ซอร์สโค้ดบรรทัดที่ 4 และ 7 แสดงคำสั่งควบคุมและบรรทัดที่ 9 แสดงสิ่งอื่น ๆ ที่เกี่ยวข้อง ซึ่งอาจเป็นคำสั่งหรือการกำหนดค่าหรือประโยคอื่น ๆ

```
1     int play(int snd, int dly, int count)
2     {
3       int    cnt;
4       for(cnt = 0;cnt < count;cnt++)
5       {
6         printf("for command, cnt =  %d\n",cnt);
7         if(bioskey(1))
8         {
9           printf("if command, bioskey  !=  0\n");
10          return 0;
11        }
12      }
13      printf("Exit for command, cnt = %d\n", cnt);
14      return 1;
15    }
```

รูปที่ 3.5 แสดงตัวอย่างซอร์สโค้ดและการกรองฟังก์ชัน



รูปที่ 3.6 แสดงตัวอย่างซอร์สโค้ดและการแยกโหนด

แยกโหนด ซอร์สโค้ดที่ถูกแยกเป็นฟังก์ชันแต่ละฟังก์ชันแล้วจะถูกนำมาแยกส่วนที่เกี่ยวข้องกับการสร้างกราฟควบคุมกระแส นั่นก็คือโหนด (nodes) และเส้น (edges) ที่เชื่อมระหว่างโหนดแต่ละโหนด โดยเส้นที่เชื่อมจะบ่งบอกถึงกระแสควบคุม (control flow) นอกจากนี้ ยังต้องหาความสัมพันธ์ของแต่ละโหนดด้วยว่าแต่ละโหนดมีโหนดก่อนหน้าเป็นโหนดใด และโหนดถัดไปคือโหนดใด ซึ่งข้อมูลนี้จะถูกจัดเก็บไว้เพื่อใช้สำหรับการสร้างกราฟควบคุมกระแสต่อไป

จากรูปที่ 3.6 ในขั้นตอนการทำงานของฟังก์ชัน “main()” จะถูกแยกโหนดและเก็บความสัมพันธ์แต่ละโหนด ซึ่งมีการเก็บข้อมูลแบบแถวลำดับ โดยแบ่งเป็น 3 ส่วนด้วยกันดังนี้

- **เก็บข้อความของโหนด** คือส่วนที่ใช้เก็บข้อความที่จะแสดงในโหนดนั้น ๆ ซึ่งอาจเป็นชื่อฟังก์ชันหรือเงื่อนไข ซึ่งกรณีที่โหนดนั้นไม่ใช่บรรทัดแรกของฟังก์ชันหรือเป็นคำสั่งที่ไม่ใช่คำสั่งควบคุม ส่วนนี้จะเป็นค่าว่าง

StrNodes(index-func , index-node)

- **เก็บโหนดทางซ้าย** ส่วนนี้จะมีความหมาย 2 ทางคือ

1. กรณีโหนดที่มี *index-node* ไม่ใช่คำสั่งควบคุม ส่วนนี้จะเก็บโหนดถัดไป หรือถ้าโหนดที่มี *index-node* เป็นโหนดสุดท้าย ส่วนนี้จะเก็บ 0

2. กรณีโหนดที่มี *index-node* เป็นคำสั่งควบคุม ส่วนนี้จะเก็บโหนดที่ต้องดำเนินการต่อไปกรณีเงื่อนไขเป็นจริง

LNodes(index-func , index-node)

- **เก็บโหนดทางขวา** คือส่วนที่ใช้เก็บโหนดที่ต้องดำเนินการต่อไปกรณีเงื่อนไขเป็นเท็จ ของโหนดที่มี *index-node* เป็นคำสั่งควบคุม ซึ่งถ้าโหนดที่มี *index-node* ไม่ใช่คำสั่งควบคุม ส่วนนี้จะเก็บ 0

RNodes(index-func , index-node)

- **index-func** คือดัชนีหรือลำดับของฟังก์ชันที่ดำเนินการแยกโหนด โดยได้กำหนดให้มีค่าตั้งแต่ 1-24 ของแต่ละซอร์สโค้ด
- **index-node** คือดัชนีหรือลำดับของโหนดที่แยกออกมาได้ ซึ่งกำหนดให้มีค่าตั้งแต่ 1-50 ของแต่ละฟังก์ชัน

ดังนั้นจากตัวอย่างเมื่อดำเนินการแยกโหนดจนเสร็จสิ้นจะมีข้อมูลที่จัดเก็บไว้คือ มี 1 ฟังก์ชัน 5 โหนดและแต่ละโหนดมีความสัมพันธ์กัน ดังนี้

โหนดที่ 1 เป็นจุดเริ่มต้นของฟังก์ชันมีโหนดซ้ายคือโหนดที่ 2 ไม่มีโหนดทางขวา ซึ่งข้อความของโหนดจะเก็บชื่อและรายละเอียดของฟังก์ชัน

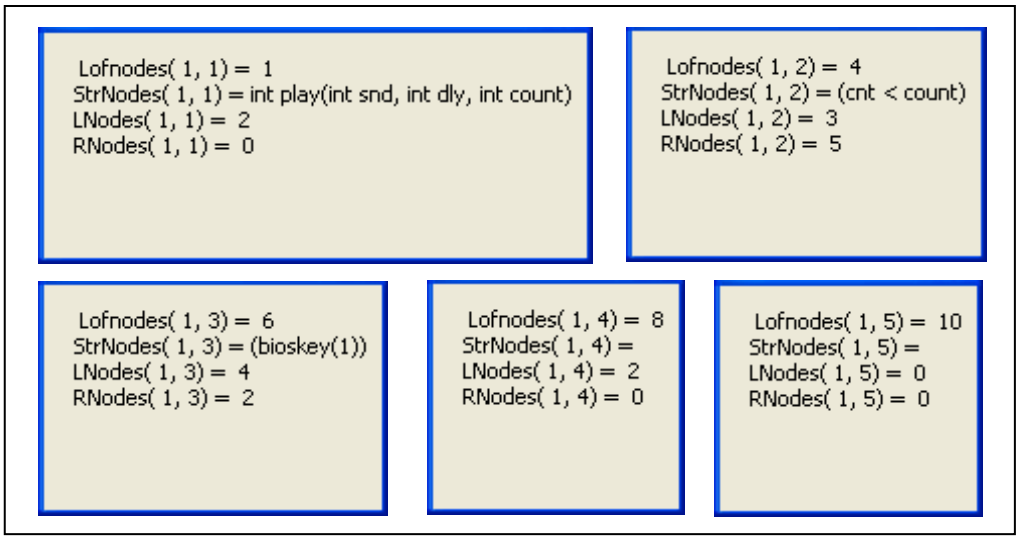
โหนดที่ 2 เป็นคำสั่งควบคุมที่มีโหนดซ้ายคือโหนดที่ 3 และโหนดทางขวาคือโหนดที่ 5 ซึ่งข้อความของโหนดจะเก็บเงื่อนไขของคำสั่งควบคุม

โหนดที่ 3 เป็นคำสั่งควบคุมเช่นกันมีโหนดทางซ้ายคือโหนดที่ 4 และโหนดทางขวาคือโหนดที่ 2 ซึ่งแสดงว่ามีการวนไปทำซ้ำที่โหนดที่ 2 ซึ่งข้อความของโหนดจะเก็บเงื่อนไข

โหนดที่ 4 เก็บการทำงานอื่นที่อยู่ในกรณีเงื่อนไขเป็นจริงของโหนดที่ 3 และไม่มีโหนดทางขวา ส่วนโหนดทางซ้ายก็วนไปทำซ้ำที่โหนดที่ 2 เช่นกัน และข้อความของโหนดจะว่าง

โหนดที่ 5 เป็นโหนดสุดท้ายสังเกตได้จากไม่มีโหนดซ้าย (โหนดถัดไปตามปกติ) และไม่มีโหนดขวา ซึ่งโหนดที่ 5 นี้จริงแล้วก็คือจุดสิ้นสุดของฟังก์ชันและข้อความของโหนดจะว่าง

รายละเอียดทั้งหมดที่กล่าวมาข้างต้น แสดงได้ดังรูปที่ 3.7 ต่อไปนี้



รูปที่ 3.7 แสดงการเก็บข้อมูลแต่ละโหนดและความเกี่ยวเนื่องกันของแต่ละโหนด

แยกโทเคน โทเคนที่กล่าวถึงในงานวิจัยนี้หมายถึง ส่วนย่อยสุดที่เครื่องมือจะสนใจที่จะแยกออกเป็นส่วน ๆ เพื่อนำไปใช้สำหรับการสร้างข้อมูลทดสอบต่อไป ซึ่งได้แบ่งชนิดของโทเคนออกเป็น 4 ประเภท ตามการพิจารณาดังนี้

- ตัวดำเนินการ (operator) หมายถึง เครื่องหมายที่ใช้แทนการปฏิบัติการ ซึ่งตัวดำเนินการที่ใช้มี 3 แบบ คือตัวดำเนินการความสัมพันธ์ (relational operator) ซึ่งมี “<”, “<=”, “>”, “>=” ตัวดำเนินการเทียบเท่า (equality operators) ได้แก่ “= =”, “!=” และตัวดำเนินการทางตรรกะ (logical operators) ซึ่งก็ได้แก่ “&&”, “||”, “!” นอกจากนี้ยังมีตัวดำเนินการที่เป็นวงเล็บเปิด “(“ และวงเล็บปิด “)” อีกด้วย

- ตัวแปร (variable) ใช้กับพวกตัวแปรชนิดต่าง ๆ ซึ่งตัวแปรที่ให้ความสนใจคือตัวแปรที่มีการประกาศขึ้นใช้งานในซอร์สโค้ดที่นำเข้าสู่เครื่องมือ เท่านั้น

- นิพจน์ (expression) ในเงื่อนไขของแต่ละโหนดอาจประกอบไปด้วย นิพจน์การคำนวณ หรือ การเรียกใช้ฟังก์ชันบางตัว ซึ่งในเครื่องมือที่พัฒนานี้จะไม่หาผลลัพธ์ของแต่ละนิพจน์ หรือ ค่าที่ได้จากการเรียกใช้ฟังก์ชัน ฉะนั้นจึงถือว่าส่วนที่เป็นนิพจน์นี้เป็นโทเคนตัวหนึ่ง

- ค่าคงที่ (constant) ที่สนใจมี 2 ลักษณะคือ ค่าคงที่แบบตัวเลข (numeric) ค่าคงที่ตัวอักษร (character)

โทเคนแต่ละประเภท เมื่อถูกแยกแล้วจะมีการเก็บข้อมูลส่วนนี้ไว้เพื่อใช้สำหรับการส่วนสร้างข้อมูลทดสอบ ซึ่งในการจัดเก็บได้ออกแบบส่วนจัดเก็บข้อมูลแต่ละโทเคนเป็นแบบแถวลำดับ โดยแบ่งเป็น 2 ส่วนดังนี้

- **เก็บข้อความของโทเคน** คือส่วนที่ใช้เก็บข้อความที่คัดแยกออกเป็นส่วน ๆ ของแต่ละโทเคน ซึ่งอาจจะเป็นโทเคนประเภทต่าง ๆ เช่น เป็นตัวดำเนินการ นิพจน์หรือค่าคงที่ ในส่วนของโครงสร้างส่วนจัดเก็บข้อมูล ออกแบบเป็นแถวลำดับแบบ 3 มิติ ดังนี้

Tokens(index-func , index-node, index-token)

- **เก็บประเภทของโทเคน** คือส่วนที่ใช้เก็บว่าโทเคนแต่ละตัวที่แยกได้จัดอยู่ในกลุ่มใด ซึ่งออกแบบให้จัดเก็บแบบแถวลำดับแบบ 3 มิติ เช่นกัน ดังนี้

TofToken(index-func , index-node, index-token)

- **index-func** คือดัชนีหรือลำดับของฟังก์ชันที่ดำเนินการแยกโทเคน ซึ่งกำหนดให้มีค่าตั้งแต่ 1-24 แต่ละซอร์สโค้ด
- **index-node** คือดัชนีหรือลำดับของโหนดที่กำลังพิจารณาแยกโทเคน โดยจะพิจารณาเฉพาะโหนดที่เป็นเงื่อนไข ซึ่งกำหนดให้มีค่าตั้งแต่ 1-24 ของแต่ละฟังก์ชัน
- **index-token** คือดัชนีหรือลำดับของโทเคนแยกออกได้ ซึ่งกำหนดให้มีค่าตั้งแต่ 1-150 ของแต่ละโหนด

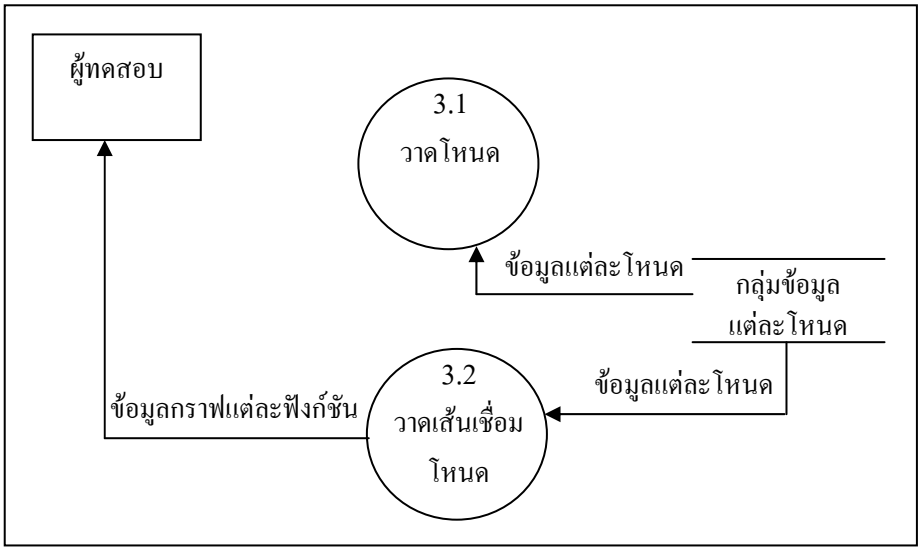
ตัวอย่าง เงื่อนไขในโหนดเงื่อนไขบางโหนดเป็น “((value>0)&& (value<30000))” และเมื่อผ่านการแยกโทเคนแล้วจะได้โทเคนทั้งหมด ดังนี้ “(”, “(”, “value”, “>”, “0”, “)”, “&&”, “(”, “value”, “<”, “30000”, “)”, “)” พร้อมกันนั้นโทเคนแต่ละตัวก็จะมีการจัดเก็บประเภทไว้ด้วย

หรือจะเป็นกรณีที่โทเคนที่แยกจะเป็นเพียงนิพจน์ทางคณิตศาสตร์โดยทั่วไปเมื่อจะใช้ในเครื่องมือนี้จะแยกเฉพาะที่สนใจ เช่น เงื่อนไข “((x=getch()) == ‘3’)” และเมื่อผ่านการแยกโทเคนแล้วจะได้โทเคนทั้งหมดดังนี้ “(”, “(”, “x=getch()”, “==”, “‘3’”, “)”

หรือเงื่อนไขเป็น “((eggx >= (x-1)*9+1) && (eggx <= x*9-1))” เมื่อผ่านการแยกโทเคนแล้วจะได้โทเคนทั้งหมด ดังนี้ “(”, “(”, “eggx”, “>=”, “(x-1)*9+1”, “)”, “&&”, “(“, “eggx”, “<=”, “x*9-1”, “)”, “)”

3.2.3 ส่วนสร้างกราฟควบคุมกระแส (Control flow graph)

หน้าที่หลักของส่วนนี้คือสร้างกราฟควบคุมกระแสให้สัมพันธ์ตามโครงสร้างของซอร์สโค้ดที่นำเข้า โดยนำข้อมูลของแต่ละโหนดที่เก็บไว้จากส่วนกระจายนิพจน์ ซึ่งจะระบุโหนดและโหนดถัดไปทั้งโหนดซ้ายและโหนดขวา มีขั้นตอนการทำงานดังรูปที่ 3.8



รูปที่ 3.8 แสดงแผนภาพกระแสข้อมูลระดับที่ 2 ของส่วนสร้างกราฟควบคุมกระแส

วาดโหนด ข้อมูลแต่ละโหนดที่เก็บไว้จะถูกดึงมาเพื่อวาดแต่ละโหนดตั้งแต่โหนดแรกจนถึงโหนดสุดท้าย โดยใช้คำสั่งทำงานแบบวนซ้ำ ซึ่งในการวนแต่ละรอบทำการวาดโหนดทางซ้าย และทางขวาของโหนดปัจจุบัน ซึ่งใช้การตรวจสอบว่ามีโหนดทางซ้ายหรือโหนดทางขวาหรือไม่ถ้ามีโหนดทางซ้ายจะทำการวาดต่ำกว่าลงไปด้านล่างส่วนโหนดทางขวาก็จะวาดถัดไปทางขวา ซึ่งการวาดโหนดทางซ้ายและขวายังต้องมีการพิจารณาปัจจัยอื่นด้วย เช่น โหนดนั้นมีความเกี่ยวข้องระดับใด ซึ่งมีผลให้กำหนดตำแหน่งของโหนดที่จะวาด เพื่อไม่ให้โหนดที่วาดทับซ้อนกัน ยกเว้นว่าโหนดนั้นเคยวาดไปแล้วก็จะข้ามการทำงานไปที่โหนดถัดไป

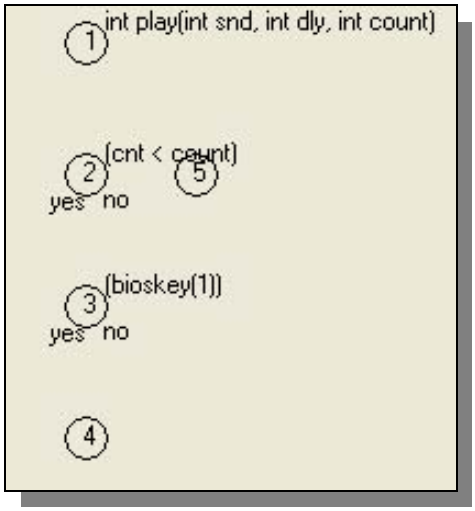
ข้อความ “yes” กับ “no” จะแสดงกรณีที่โหนดนั้นเป็นคำสั่งควบคุม และมีการตรวจสอบเงื่อนไข จะเกิดเป็น 2 กรณีคือ กรณีเงื่อนไขเป็นจริง ซึ่งในที่นี้ใช้ “yes” กำหนดให้ทำงานในโหนดทางซ้าย ส่วนกรณีที่เงื่อนไขเป็นเท็จ แทนด้วย “no” กำหนดให้ทำงานในโหนดทางขวา แต่ถ้ากรณีที่โหดนนั้นไม่มีการตรวจสอบเงื่อนไข “yes” กับ “no” ก็จะไม่แสดงให้เห็นทางหน้าจอ

สำหรับการวาดโหนดให้เป็นรูปวงกลมของแต่ละฟังก์ชันนั้น มีการแสดงผลด้วยวัตถุชนิด “Shape” โดย “Shape” จะมีพื้นหลังเป็น “PictureBox” และมีลักษณะเป็นแถวลำดับ

เช่นเดียวกัน ซึ่งแต่ละฟังก์ชันจะกำหนดให้ “Shape” มีชื่อแตกต่างกัน และ “Shape” ของแต่ละฟังก์ชันจะอาศัย *index-node* เป็นตัวอ้างอิงเพื่อจัดการแต่ละโหนด ด้วยการสร้างวัตถุขึ้นมาใหม่ให้มีขนาดและรูปแบบเหมือนต้นฉบับ ให้ครบตามจำนวนโหนดที่มีในแต่ละฟังก์ชัน ดังนี้

Load Shape-func(index-node)

จากนั้นค่อยกำหนดตำแหน่งของวัตถุ ให้แสดงในตำแหน่งที่เหมาะสมตามความเกี่ยวข้องกันของแต่ละโหนดต่อไป ดังรูปที่ 3.9



รูปที่ 3.9 แสดงการวาดโหนดที่ใช้ข้อมูลจากส่วนแยกโหนด

วาดเส้นเชื่อมโหนด เมื่อวาดโหนดครบทุกโหนดแล้ว ขั้นตอนการทำงานต่อไปคือวาดเส้นเชื่อมระหว่างโหนด โดยใช้วิธีคล้ายกับการวาดโหนดแต่ละโหนดคือใช้คำสั่งวนซ้ำเพื่อวาดเส้นเชื่อมไปยังโหนดทางซ้ายและโหนดทางขวา ในการวาดเส้นต้องตรวจสอบด้วยว่ามีโหนดอื่นขวางเส้นทางอยู่หรือไม่ ถ้าขวางต้องวาดเส้นให้หลบโหนดเหล่านั้นด้วยเพื่อไม่ให้เส้นไปทับซ้อนกับโหนดหรือเส้นอื่น ๆ ได้

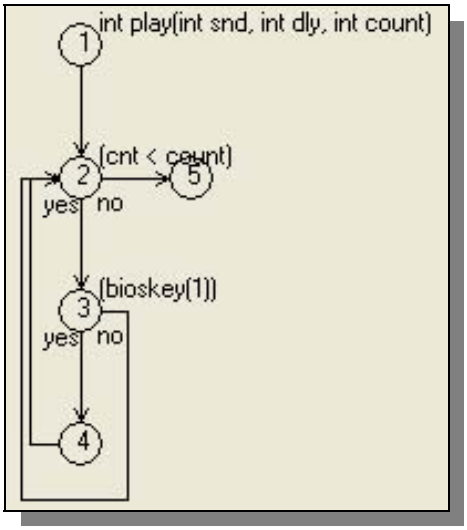
ในการวาดเส้นแสดงผลด้วยคำสั่งในการวาดเส้น “Line” ซึ่งเป็นคำสั่งในกลุ่มของคำสั่งที่ใช้เกี่ยวกับการวาดรูปทรงต่าง ๆ ในโปรแกรมวิชวลเบสิก ซึ่งการใช้ “Line” วาดลงบนวัตถุชนิด “PictureBox” ซึ่ง “PictureBox” จะมีลักษณะแบบแถวลำดับอีกเช่นกัน อ้างอิงด้วย *index-func* โดยการใช้ “Line” เพื่อวาดเส้นจะต้องมีการกำหนดจุดพิกัดแนวนอน “x” และแนวตั้ง “y” ให้เป็น

จุดเริ่มต้นและจุดสิ้นสุด โดยที่พิกัดที่ว่านี้ต้องกำหนดให้เหมาะสมกับโหนดแต่ละโหนดที่เป็นโหนดต้นทางและโหนดปลายทางด้วย ซึ่งมีรูปแบบการใช้ดังนี้

PictureBox(index-func).Line(x1,y1) – (x2,y2)

x1 และ y1 คือพิกัดแนวนอนและแนวตั้งที่เป็นจุดเริ่มต้นของเส้นที่จะวาด
x2 และ y2 คือพิกัดแนวนอนและแนวตั้งที่เป็นจุดสิ้นสุดของเส้น

ซึ่งเมื่อวาดเส้นเชื่อมตามรูปแบบที่ออกแบบไว้เสร็จสิ้นก็จะได้กราฟที่มีลักษณะดังรูปที่ 3.10



รูปที่ 3.10 แสดงการวาดเส้นเชื่อมระหว่างโหนดที่ใช้ข้อมูลจากส่วนแยกโหนด

3.2.4 ส่วนสร้างเส้นทางสำหรับทดสอบ (Test Path)

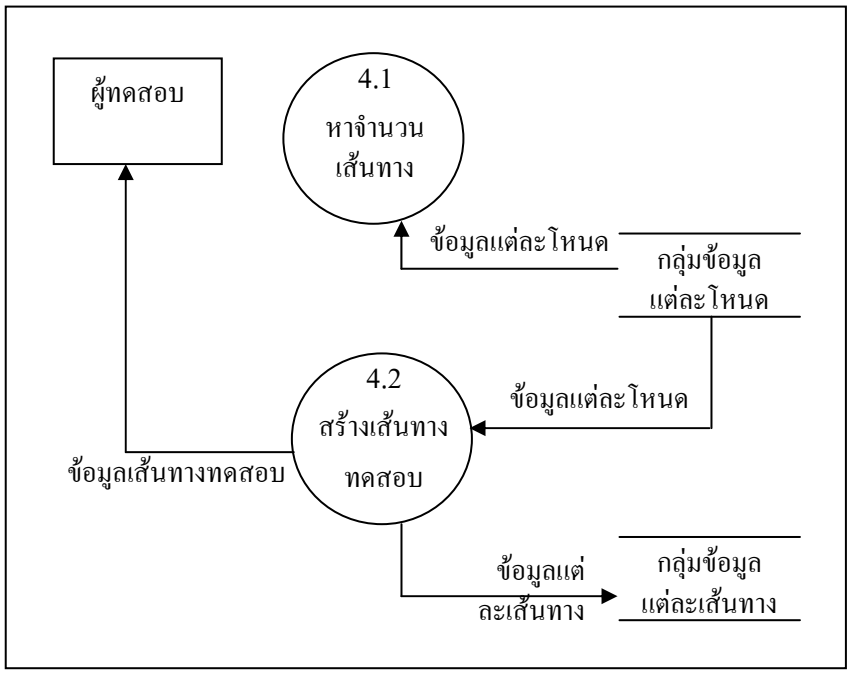
เพื่อให้ผู้ทดสอบสามารถแยกแยะเส้นทางที่จะต้องทดสอบได้ชัดเจนขึ้น เครื่องมือจะสร้างเส้นทางสำหรับการทดสอบ อำนวยความสะดวกให้แก่ผู้ทดสอบ โดยเส้นทางเหล่านี้ต้องครอบคลุม สำหรับการทดสอบ และจำนวนเส้นทางทดสอบที่สร้างจากเครื่องมือนี้จะเท่ากับจำนวนเส้นทางอิสระของโปรแกรม ซึ่งสูตรที่ใช้สำหรับคำนวณ (Ian Sommerville, 2001) ดังสูตรที่แสดง

CC(G) = จำนวนเส้น (edges) – จำนวนโหนด (nodes) + 2

โดยที่สามารถคำนวณหาค่านี้ได้จากอีกสูตรหนึ่ง นั่นคือ

CC(G) = จำนวนโหนดเงื่อนไข (Condition nodes) + 1

ซึ่งทั้งสองสูตรนี้ให้ค่าจำนวนเส้นทางอิสระของโปรแกรมออกมาเท่ากัน (Ian Sommerville, 2001) ภายใต้เงื่อนไขที่ว่าซอร์สโค้ดจะต้องไม่มีคำสั่งจำพวก “goto” ในส่วนสร้างเส้นทางสำหรับทดสอบนี้มีรายละเอียดการทำงานดังรูปที่ 3.11



รูปที่ 3.11 แสดงแผนภาพกระแสข้อมูลระดับที่ 2 ของส่วนสร้างเส้นทางสำหรับทดสอบ

หาจำนวนเส้นทางทดสอบ ส่วนนี้คือส่วนที่เครื่องมือจะทำการประมวลผลเพื่อหาจำนวนเส้นทางทดสอบที่จะสร้างก่อนเพื่อใช้เป็นตัวกำหนดจำนวนเส้นทางที่จะต้องสร้าง และเพื่อจะได้มั่นใจว่าเส้นทางที่ได้ไม่มีจำนวนจนมากเกินความจำเป็น หรือไม่จำนวนน้อยจนเกินไปไม่ครอบคลุมสำหรับการทดสอบ โดยจะอาศัยการตรวจสอบจำนวนโหนดที่เป็นเงื่อนไขแล้วบวกด้วยหนึ่ง ซึ่งตรงกับสูตรที่สองที่อธิบายไปในข้างต้น

ในการประมวลผลจะมีการตรวจสอบจากข้อมูลแต่ละโหนดที่มีการจัดเก็บไว้จากส่วนแยกโหนด นอกจากจะตรวจสอบเพื่อหาจำนวนเส้นทางทดสอบแล้ว ยังต้องทราบว่าโหนดใดเป็นโหนดเงื่อนไขด้วยเพราะมีความจำเป็นสำหรับการสร้างเส้นทางทดสอบ

สร้างเส้นทางทดสอบ สิ่งหนึ่งที่ต้องระวังในการสร้างเส้นทางทดสอบคือจำนวนเส้นทาง และการซ้ำซ้อนของเส้นทาง ซึ่งปัญหาแรกถูกแก้ไขด้วยการหาจำนวนเส้นทางไว้แล้ว ส่วนขั้นตอนที่สองนี้ต้องตรวจสอบระหว่างที่มีการสร้างเส้นทางทดสอบ เพื่อให้การประมวลผลถูกต้องตามต้องการมีการออกส่วนเก็บข้อมูลขึ้นมาจัดเก็บข้อมูลระหว่างการประมวลผลแบ่งเป็น 2 ส่วนคือ เก็บเส้นทางทดสอบที่สร้างและเก็บความเกี่ยวเนื่องของโหนดในแต่ละเส้นทางที่สร้างขึ้น ซึ่งส่วนที่สองนี้จะเป็นการเตรียมข้อมูลเพื่อการสร้างข้อมูลทดสอบ

- **เก็บเส้นทางทดสอบ** คือส่วนที่ต้องเก็บเส้นทางทดสอบที่สร้างแต่ละเส้นเพื่อจะได้ตรวจสอบไม่ให้เกิดการซ้ำซ้อนของเส้นทาง และเก็บไว้เพื่อให้การแสดงผลทำได้ง่ายขึ้น ซึ่งมีการออกแบบให้เก็บข้อมูลแบบแถวลำดับ 2 มิติดังนี้

StrPath(index-func, index-path)

- **index-path** คือดัชนีหรือลำดับของเส้นทางของแต่ละฟังก์ชัน ลักษณะของข้อมูลที่เก็บไว้ใน “StrPath(,,)” เช่น 1-2-3-2-5 เป็นต้น

- **เก็บความเกี่ยวเนื่องของโหนด** คือส่วนที่ทำหน้าที่เก็บความสัมพันธ์ของโหนดที่เกี่ยวข้องกันในแต่ละเส้นทางทดสอบ โดยข้อมูลนี้จะไม่ถูกแสดงในส่วนสร้างเส้นทางทดสอบ แต่จะถูกเก็บไว้เพื่อใช้ในส่วนต่อไปนั่นคือส่วนสร้างข้อมูลทดสอบ การเก็บความเกี่ยวเนื่องของโหนดแบ่งเป็นการเก็บโหนดแต่ละโหนดในเส้นทางและเก็บโหนดถัดไปที่ต่อเนื่องในเส้นทางทดสอบ ซึ่งมีการออกแบบให้เก็บข้อมูลแบบแถวลำดับ 3 มิติดังนี้

Tracer(index-func, index-path, trace)

และ

NextNode(index-func, index-path, trace)

- **trace** คือลำดับของโหนดที่อยู่ในเส้นทางทดสอบแต่ละเส้นทาง

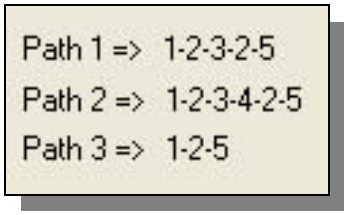
หลังจากสร้างเส้นทางทดสอบเสร็จเรียบร้อย ก็ขั้นตอนของการแสดงผลเส้นทางที่ได้ให้ผู้ทดสอบได้เห็น โดยในการแสดงผลมีการแสดงผลโดยใช้วัตถุชนิด “Label” โดยแต่ละฟังก์ชันจะกำหนด “Label” ที่ต่างกัน และ “Label” ที่ใช้แต่ละฟังก์ชันจะมีลักษณะแบบแถวลำดับอ้างอิงด้วย *index-path* ซึ่งมีรูปแบบการใช้ดังนี้

Load Label-func(index-path)

กำหนดให้ “Label” แสดงเส้นทางดังนี้

Label-func(index-path).caption = StrPath(index-func, index-path)

เมื่อการทำงานเสร็จสิ้นจะได้เส้นทางทดสอบดังรูปที่ 3.12

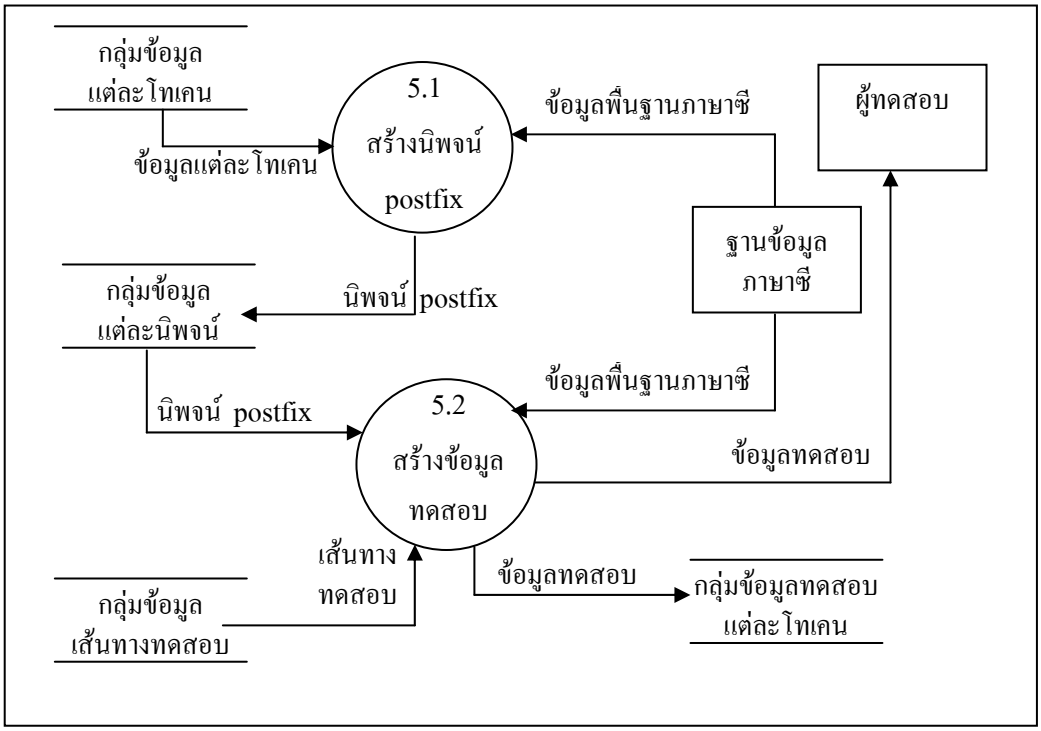


รูปที่ 3.12 แสดงเส้นทางทดสอบ ที่สร้างจากเครื่องมือ

3.2.5 ส่วนสร้างข้อมูลทดสอบ (Test Data)

สำหรับสร้างข้อมูลทดสอบเป็นส่วนสุดท้ายของเครื่องมือสร้างกราฟควบคุมกระแส และข้อมูลทดสอบสำหรับภาษาซี โดยส่วนนี้จะอาศัยข้อมูลการประมวลผลจากส่วนกระจายนิพจน์ และส่วนสร้างเส้นทางทดสอบ รวมถึงอาศัยหลักเกณฑ์ต่าง ๆ จากฐานข้อมูล ดังแสดงในรูปที่ 3.13

ในขั้นตอนการทำงานของส่วนสร้างเส้นทางทดสอบจะเริ่มจากการพิจารณาทีละโทเคนของโหนดที่เป็นเงื่อนไขในแต่ละฟังก์ชันที่ได้แยกไว้แล้วในส่วนกระจายนิพจน์ โดยนำโทเคนมาพิจารณาทีละโหนด โดยจัดรูปแบบของโทเคนที่ประกอบเป็นนิพจน์เงื่อนไขของโหนดใด ๆ ใหม่ให้อยู่ในรูปแบบนิพจน์แบบ postfix เพื่อจะได้รู้ถึงลำดับการพิจารณาตรวจสอบค่าความจริง-เท็จของเงื่อนไขนั้นเมื่อได้เป็นนิพจน์ postfix จะมีการเก็บข้อมูลไว้เพื่อนำไปใช้สำหรับสร้างเป็นข้อมูลทดสอบต่อไป



รูปที่ 3.13 แสดงแผนภาพกระแสข้อมูลระดับที่ 2 ของส่วนสร้างข้อมูลทดสอบ

สำหรับนิพจน์ postfix คือการแปลงรูปนิพจน์คณิตศาสตร์โดยทั่ว ๆ ไปซึ่งเรียกว่านิพจน์แบบ infix ให้อยู่ในรูปแบบที่ระบบคอมพิวเตอร์ใช้สำหรับหาผลลัพธ์ของนิพจน์คณิตศาสตร์ ตัวอย่างเช่น

- นิพจน์แบบ infix คือ (a+b) * (c-d)
- แปลงเป็นนิพจน์แบบ postfix ได้เป็น ab+cd-*

แต่สำหรับในเครื่องมือที่พัฒนานี้ นิพจน์แบบ infix จะไม่ใช่การคำนวณในลักษณะแบบตัวอย่างที่แสดงอยู่ด้านบน แต่จะอยู่ในรูปแบบของเงื่อนไข ตัวอย่างเช่น

- นิพจน์แบบ infix แบบปกติทั่วไปเป็น “((value>0) && (value<30000))”
- เมื่อแยกโทเคนจะได้ “(”, “(”, “value”, “>”, “0”, “)”, “&&”, “(”, “value”, “<”, “30000”, “)”, “)”
- เปลี่ยนเป็น postfix จะได้ “value”, “0”, “>”, “value”, “30000”, “<”, “&&”

กรณีที่โทเคนที่แยกเป็นนิพจน์ทางคณิตศาสตร์ที่ไม่ใช่แบบตรรกะเครื่องมือนี้จะแยกเฉพาะที่สนใจ เช่น

- นิพจน์แบบ infix คือ “((x=getch()) == ‘3’)”
- เมื่อแยกโทเคนจะได้ “(”, “(”, “x=getch()”, “==”, “‘3’”, “)”

- เปลี่ยนเป็น postfix จะได้ “x=getch()”, “‘3’”, “==”
- นิพจน์แบบ infix คือ “((eggx >= (x-1)*9+1) && (eggx <= x*9-1))”
- เมื่อแยกโทเคนจะได้ “(”, “(”, “eggx”, “>=”, “(x-1)*9+1”, “)”, “&&”, “(“, “eggx”, “<=”, “x*9-1”, “)”, “)”
- เปลี่ยนเป็น postfix จะได้ “eggx”, “(x-1)*9+1”, “>=”, “eggx”, “x*9-1”, “<=”, “&&”

เมื่อได้โทเคนแต่ละตัวอยู่ในรูปแบบ postfix แล้วจะทำให้ได้ลำดับการทำงานของเครื่องหมายต่าง ๆ เพื่อตรวจสอบหาผลลัพธ์ของนิพจน์นี้ด้วย ซึ่งส่งผลให้การสร้างข้อมูลทดสอบได้ข้อมูลที่สมเหตุสมผลและในการสร้างข้อมูลทดสอบจะอาศัยโทเคนแต่ละตัวที่อยู่ในรูปแบบนิพจน์ postfix โดยได้ออกแบบส่วนเก็บข้อมูลทดสอบเป็นแบบแถวลำดับแบบ 3 มิติเช่นเดียวกับส่วนเก็บโทเคน โดยออกแบบเพื่อเก็บข้อมูล 2 ส่วนดังนี้

- **เก็บข้อมูลทดสอบเพื่อให้เงื่อนไขเป็นจริง** คือส่วนที่ใช้สำหรับเก็บข้อมูลเตรียมไว้ให้กรณีเงื่อนไขเป็นจริง ดังนี้

DataYes(index-func , index-node,index-token)

- **เก็บข้อมูลทดสอบเพื่อให้เงื่อนไขเป็นเท็จ** คือส่วนที่ใช้สำหรับเก็บข้อมูลเตรียมไว้ให้กรณีเงื่อนไขเป็นเท็จ ดังนี้

DataNo(index-func , index-node,index-token)

- **index-func** คือดัชนีหรือลำดับของฟังก์ชันที่ดำเนินการแยกโทเคน กำหนดให้มีค่าตั้งแต่ 1-24
- **index-node** คือดัชนีหรือลำดับของโหนดที่กำลังพิจารณาแยกโทเคน โดยจะพิจารณาเฉพาะโหนดที่เป็นเงื่อนไข กำหนดให้มีค่าตั้งแต่ 1-24 ของแต่ละฟังก์ชัน
- **index-token** คือดัชนีหรือลำดับของโทเคนแยกออกได้ ซึ่งกำหนดให้มีค่าตั้งแต่ 150 ของแต่ละโหนด

ในแต่ละโหนดเงื่อนไขประกอบไปด้วยโทเคนหลายตัว ซึ่งบางตัวไม่จำเป็นต้องกำหนดข้อมูลทดสอบ อาจจะเนื่องมาจากโทเคนบางตัวเป็นค่าคงที่อยู่แล้ว ฉะนั้นโทเคนตัวไหนที่ไม่มีข้อมูลทั้งใน DataYes(,,) และ DataNo(,,) แสดงว่าโทเคนนั้นไม่เกี่ยวข้องหรือไม่จำเป็นต้องสร้างข้อมูลทดสอบให้ สำหรับส่วนของการแสดงผลข้อมูลทดสอบ กำหนดให้มีการแสดงผลโดยใช้วัตถุ

ชนิด “Label” โดยแต่ละเส้นทางของแต่ละฟังก์ชันจะกำหนด “Label” ต่างกันและ “Label” ที่ใช้แต่ละฟังก์ชันจะมีลักษณะแบบแถวลำดับ อ้างอิงด้วย index-path ซึ่งมีรูปแบบการใช้ดังนี้

Load Label-func(index-path)

กำหนดให้ “Label” แสดงเส้นทางดังนี้

LabelD-func(index-path).caption = DataYes(index-func, index-node, index-token)

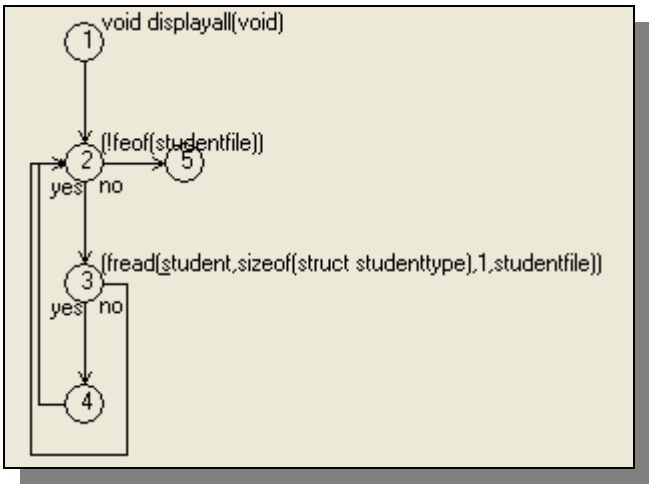
หรือ

Label-func(index-path).caption = DataNo(index-func, index-node, index-token)

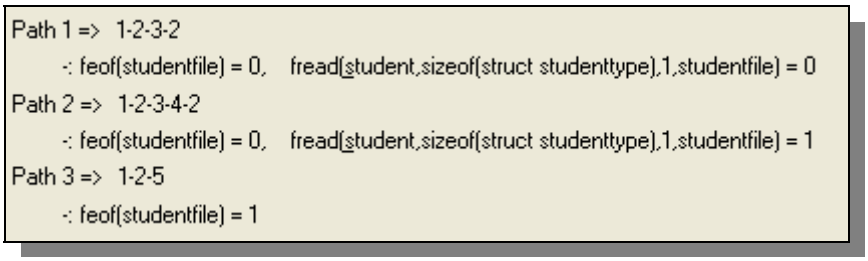
เมื่อการทำงานเสร็จสิ้น จะได้เส้นพร้อมแสดงข้อมูลทดสอบแต่ละโทเคนของแต่ละเส้นทางทดสอบ ส่วนขั้นตอนต่อไปจากรูปที่ 3.14 เป็นการแสดงกราฟควบคุมกระแสที่ได้จากเครื่องมือ จากนั้นเมื่อผ่านกระบวนการสร้างเส้นทางทดสอบและข้อมูลทดสอบแล้วทั้งเส้นทางทดสอบและข้อมูลทดสอบจะถูกแสดงออกมาพร้อมกันโดยแสดงเส้นทางและข้อมูลทดสอบที่ควรในแต่ละโหนดที่เป็นเงื่อนไขในแต่ละเส้นทาง

จากกราฟควบคุมกระแสที่แสดงในรูปที่ 3.14 มีเส้นทางทดสอบได้ 3 เส้นทาง ซึ่งในแต่ละเส้นทางสามารถใช้ข้อมูลที่กำหนดให้เพื่อให้การทำงานของโปรแกรมวิ่งไปผ่านตามเส้นทางทดสอบที่ได้อธิบายได้ดังนี้

เส้นทางที่ 1 (path 1) => การทำงานเริ่มที่โหนดที่ 1 แล้วไปที่โหนด 2 ในโหนด 2 นี้เป็นโหนดที่มีเงื่อนไขและสามารถแยกโทเคนได้เป็น “(“, “!”, “feof(studentfile)”, “)” และโทเคนที่ต้องการข้อมูลทดสอบเพื่อให้วิ่งไปตามเส้นทางทดสอบคือ “feof(studentfile)” โดยตามเส้นทางโหนดต่อไปคือโหนด 3 ซึ่งจะไปโหนด 3 ได้นั้นเงื่อนไขต้องเป็นจริง และเงื่อนจะเป็นจริงได้ “feof(studentfile)” = 0 เนื่องจากด้านหน้าของ “feof(studentfile)” มีตัวดำเนินการตรรกะ “!” จากนั้นโหนดต่อไปคือโหนด 3 ซึ่งเป็นโหนดเงื่อนไขเช่นเดียวกันแยกโทเคนได้เป็น “(“, “fread(&student,sizeof(struct studenttype),1,studentfile)”, “)” ซึ่งโทเคนที่ต้องการข้อมูลทดสอบคือ “fread(&student,sizeof(struct studenttype),1,studentfile)” จากเส้นทางทดสอบโหนดต่อไปคือโหนด 2 ซึ่งจะไปโหนด 2 ได้โหนด 3 ต้องเป็นเท็จซึ่งจะได้ “fread(&student,sizeof(struct studenttype),1,studentfile)” = 0 เงื่อนไขจึงจะเป็นเท็จ

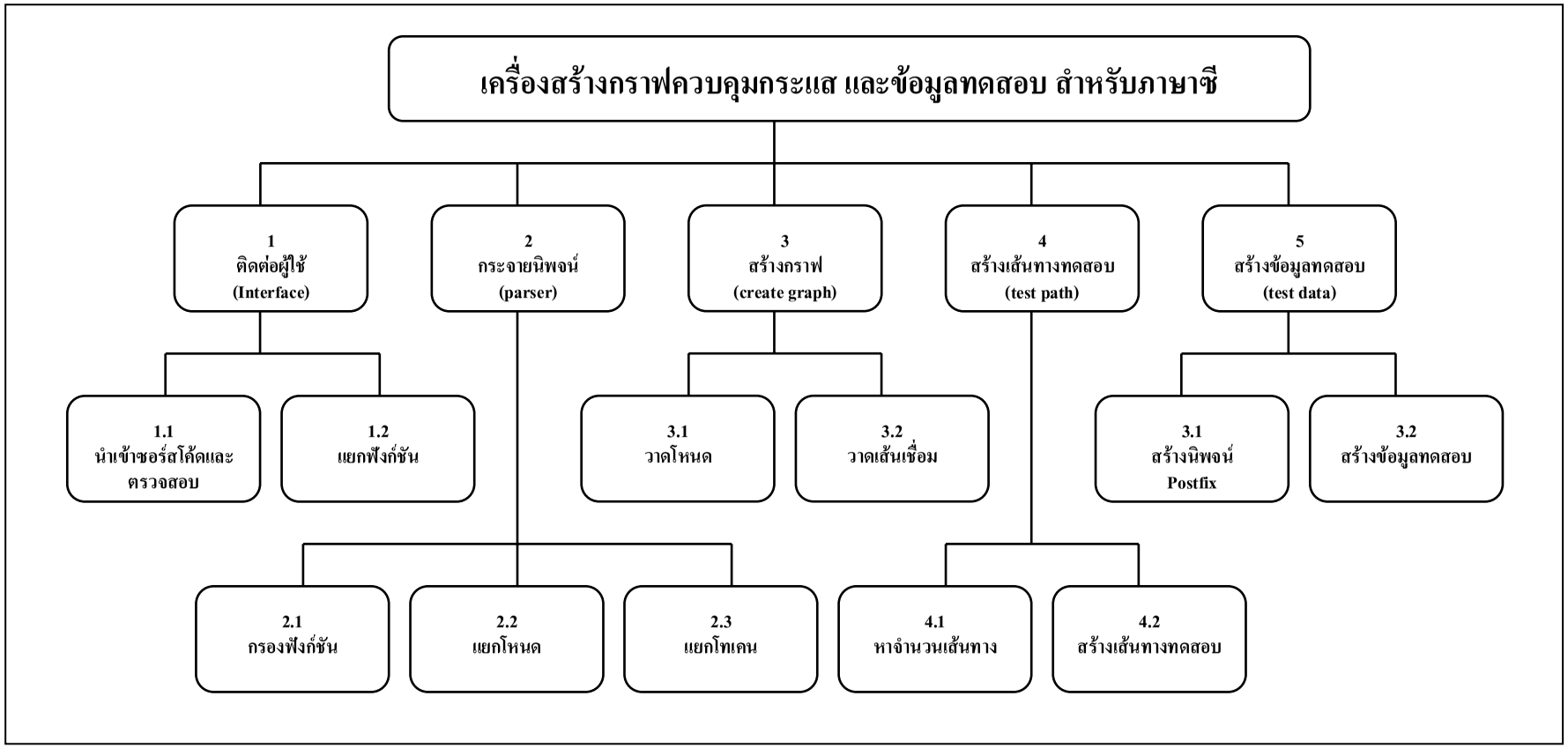


รูปที่ 3.14 แสดงกราฟควบคุมกระแสที่สร้างจากเครื่องมือ



รูปที่ 3.15 แสดงเส้นทางทดสอบและข้อมูลทดสอบของแต่ละเส้นทาง

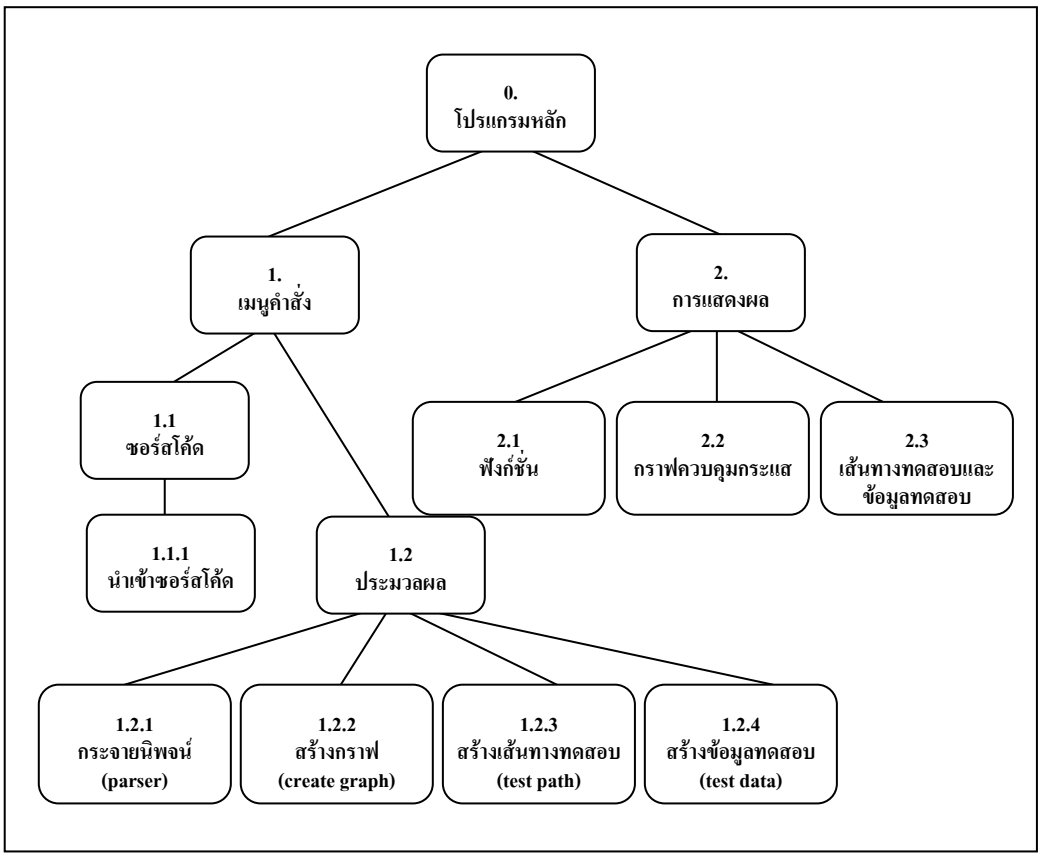
ส่วนเส้นทางที่เหลือแต่ละเส้นทางก็จะมีหลักการทำงานในลักษณะเช่นเดียวกันนี้ไปจนครบทุกฟังก์ชัน กล่าวโดยสรุปการทำงานทั้งหมดทุกส่วนที่กล่าวมาข้างต้นสามารถอธิบายโครงสร้างได้ดังรูปที่ 3.16



รูปที่ 3.16 แสดงโครงสร้างการประมวลผลของ เครื่องมือสร้างกราฟควบคุมกระแส และข้อมูลทดสอบ สำหรับภาษาซี

3.3 โครงสร้างของเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบ

การพัฒนาเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบ ได้แบ่งโครงสร้างส่วนการทำงานของเครื่องมือออกแบบส่วน ๆ ประกอบไปด้วยส่วนการทำงาน ดังแสดงในรูปที่ 3.17



รูปที่ 3.17 แสดงโครงสร้างส่วนประกอบของเครื่องมือสร้างกราฟควบคุมกระแส และข้อมูลทดสอบสำหรับภาษาซี

3.3.1 เมนูคำสั่ง

เป็นส่วนที่ใช้สำหรับรับคำสั่งจากผู้ใช้ เพื่อจะได้สั่งให้เครื่องมือทำงานตามขั้นตอนต่อไป โดยในส่วนของเมนูคำสั่งนี้ได้แบ่งโครงสร้างการทำงานเป็นส่วนย่อย ๆ ดังนี้

ซอร์สโค้ด เป็นส่วนที่เกี่ยวกับซอร์สโค้ดที่ต้องการจะนำเข้าทดสอบในเครื่องมือ ซึ่งมีโครงสร้างการทำงานดังนี้

นำเข้าซอร์สโค้ด เป็นส่วนที่ใช้รับซอร์สโค้ดภาษาซีที่ต้องการทดสอบ

ประมวลผล ใช้สำหรับเลือกคำสั่งเพื่อสั่งงานให้เครื่องมือทำงานตามลำดับขั้นตอน ประกอบด้วย การประมวลผล 4 ขั้นตอน ดังนี้

- กระจายนิพจน์ เป็นส่วนที่กระจายซอร์สโค้ดแต่ละฟังก์ชันออก เพื่อให้ได้ส่วนที่สร้างกราฟ และเตรียมไว้สำหรับสร้างข้อมูลทดสอบ
- สร้างกราฟ เป็นส่วนที่สั่งให้เครื่องมือสร้างกราฟควบคุมกระแส ของแต่ละฟังก์ชัน
- สร้างเส้นทางทดสอบ เป็นส่วนที่สร้างเส้นทางทดสอบโดยจะพิจารณาจากกราฟที่สร้างแต่ละฟังก์ชัน
- สร้างข้อมูลทดสอบ เป็นขั้นตอนสุดท้ายของเครื่องมือ คือสร้างข้อมูลทดสอบให้สำหรับแต่ละเส้นทางทดสอบที่เครื่องมือสร้างเตรียมไว้

3.3.2 การแสดงผล

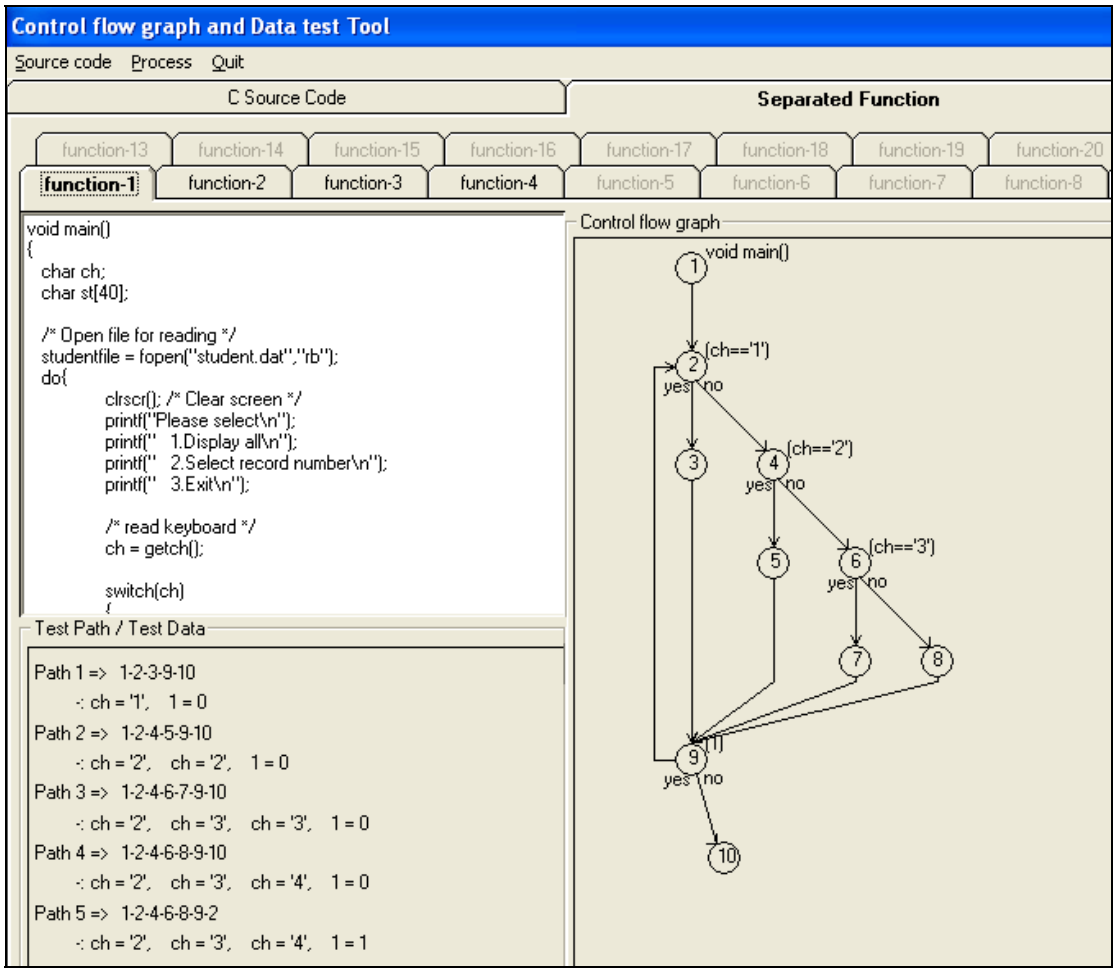
เป็นส่วนที่ใช้แสดงผลลัพธ์การทำงานของเครื่องมือ โดยกำหนดการแสดงผลให้มีการแยกส่วนของแต่ละฟังก์ชันโดยชัดเจนอาศัยวัตถุชนิด “SSTab” ซึ่งกำหนดให้สามารถแสดงสูงสุดได้ 24 แท็บ หรือ 24 ฟังก์ชัน ในการนำเข้าซอร์สโค้ด 1 ไฟล์ ซึ่งในแต่ละแท็บผลลัพธ์จะแสดงทางหน้าจอให้แก่ผู้ทดสอบ ประกอบด้วย 3 ส่วนด้วยกัน ดังนี้

ฟังก์ชัน เป็นส่วนที่แสดงซอร์สโค้ดแต่ละฟังก์ชันที่ถูกแยกออกมาจาก ซอร์สโค้ดที่นำเข้ากรณีที่มีหลาย ๆ ฟังก์ชันแต่ละฟังก์ชันก็จะกระจายไปแสดงผลที่แต่ละแท็บ

กราฟควบคุมกระแส คือกราฟควบคุมกระแสที่จะได้จากการสร้างของเครื่องมือ

เส้นทางทดสอบและข้อมูลทดสอบ คือส่วนที่จะแสดงเส้นทางทดสอบและข้อมูลทดสอบที่เครื่องมือสร้างออกมาได้

รายละเอียดโครงสร้างดังกล่าวแสดงได้ดังรูป 3.18



รูปที่ 3.18 แสดงตัวอย่างการแสดงผลฟังก์ชัน กราฟควบคุมกระแส เส้นทางทดสอบ และข้อมูลทดสอบของแต่ละฟังก์ชัน

3.4 รายละเอียดการออกแบบส่วนการจัดเก็บข้อมูลพื้นฐานภาษาซี

นอกจากข้อมูลที่มีการจัดเก็บระหว่างการประมวลผลแต่ละขั้นตอนแล้ว ยังมีข้อมูลอีกกลุ่มหนึ่งที่เป็นข้อมูลเกี่ยวกับคำสั่ง และเครื่องหมายต่าง ๆ ที่ใช้ในภาษาซี ซึ่งข้อมูลส่วนนี้จะมีการจัดเก็บไว้ล่วงหน้าด้วยวิธีการกรอกข้อมูลเข้าสู่ฐานข้อมูลโดยตรง และข้อมูลจะมีการจัดเก็บอย่างคงที่ การออกแบบตารางเพื่อจัดเก็บข้อมูลดังกล่าวมีทั้งหมด 5 ตาราง แต่ละตารางมีรายละเอียดดังนี้

1. ตารางกำหนดค่าตั้งต้นระบบ (Config Table) คือตารางที่ใช้กำหนดค่าตั้งต้นของระบบ เพื่อให้ระบบนำข้อมูลนี้ไปใช้ เพื่อเป็นค่าตั้งต้นระบบ หรือในการประมวลผลในส่วนต่าง ๆ ประกอบด้วยเขตข้อมูล (Field) ต่าง ๆ ดังตารางที่ 3.1

2. ตารางข้อกำหนดชนิดข้อมูล (DataRule Table) คือตารางที่ใช้เก็บขอบเขตของชนิดของข้อมูล แต่ละชนิด ซึ่งจะมีรายละเอียดเกี่ยวกับค่าต่ำสุด และสูงสุดของชนิดข้อมูล ประกอบด้วยเขตข้อมูลต่าง ๆ ดังตารางที่ 3.2

3. ตารางโทเคน (Token Table) คือตารางที่ใช้จัดเก็บส่วนย่อยสุดของซอร์สโค้ด ซึ่งประกอบด้วย คำ ที่เป็นคำสงวน คำสั่ง และเครื่องหมายต่าง ๆ ที่ถูกใช้ในหลักการพื้นฐานการเขียนโปรแกรมด้วยภาษาซี ประกอบด้วยเขตข้อมูลต่าง ๆ ดังตารางที่ 3.3

4. ตารางข้อกำหนดของคำสั่งควบคุม (CmdRule Table) คือส่วนที่ใช้เก็บหลักการของคำสั่งควบคุม ประกอบด้วยเขตข้อมูลต่างๆ ดังตารางที่ 3.4

5. ตารางตัวดำเนินการ (Operator Table) คือตารางเก็บตัวดำเนินการแบบต่าง ๆ พร้อมแยกประเภท จัดลำดับความสำคัญ ของตัวดำเนินการที่เครื่องมือที่พัฒนานี้จะต้องใช้ รายละเอียดดังแสดงในตารางที่ 3.5

ตารางที่ 3.1 แสดงโครงสร้างของตารางกำหนดค่าตั้งต้นระบบ

ลำดับ	ชื่อเขตข้อมูล	ชนิดของเขตข้อมูล	คำอธิบาย
1	TypeLG	Text	กำหนดภาษาที่ใช้เครื่องมือ หรือหลักการต่าง ๆ ที่กำหนดไว้ออกแบบเพื่อให้สามารถพัฒนาต่อเพื่อใช้ได้กับภาษาอื่น ๆ ด้วย
2	OpBlock	Text	กำหนดเครื่องหมายเริ่มต้นบล็อก เป็นการเก็บเครื่องหมายที่ใช้เป็นตัวเริ่มต้นบล็อก หรือตัวกำหนดขอบเขตของการใช้ตัวแปร

ตารางที่ 3.1 แสดงโครงสร้างของตารางกำหนดค่าตั้งต้นระบบ (ต่อ)

ลำดับ	ชื่อเขตข้อมูล	ชนิดของเขตข้อมูล	คำอธิบาย
3	ClBlock	Text	กำหนดเครื่องหมายจบบล็อก เป็นการเก็บเครื่องหมายที่ใช้เป็นตัวระบุการสิ้นสุดขอบเขตของบล็อก
4	ECommand	Text	กำหนดเครื่องหมายจบประโยค
5	FComment	Text	กำหนดเครื่องหมายกำหนดหมายเหตุ
6	NComment	Text	กำหนดเครื่องหมายกำหนดหมายเหตุ
7	FileType1	Text	กำหนดชนิดของไฟล์ที่นำเข้า
8	FileType2	Text	กำหนดชนิดของไฟล์ที่นำเข้า
9	cmd-if	Byte	กำหนดหมายเลขคำสั่งควบคุม ที่เป็นคำสั่ง if
10	cmd-else	Byte	กำหนดหมายเลขคำสั่งควบคุมที่เป็นคำสั่ง else
11	cmd-loop	Byte	กำหนดหมายเลขคำสั่งควบคุมที่เป็นคำสั่งวนซ้ำ
12	cmd-do	Byte	กำหนดหมายเลขคำสั่งควบคุมที่เป็นคำสั่งวนซ้ำแบบ do-while
13	cmd-switch	Byte	กำหนดหมายเลขคำสั่งควบคุมที่เป็นคำสั่ง switch
14	cmd-case	Byte	กำหนดหมายเลขคำสั่งควบคุมที่เป็นคำสั่ง case
15	cmd-default	Byte	กำหนดหมายเลขคำสั่งควบคุมที่เป็นคำสั่ง default
16	AOpr	Byte	กำหนดหมายเลขกำกับตัวดำเนินการคณิตศาสตร์
17	ROpr	Byte	กำหนดหมายเลขกำดับตัวดำเนินการแบบสัมพันธ์
18	LOpr	Byte	กำหนดหมายเลขกำดับตัวดำเนินการแบบตรรกะ

ตารางที่ 3.2 แสดงโครงสร้างของตารางข้อกำหนดชนิดข้อมูล

ลำดับ	ชื่อเขตข้อมูล	ชนิดของเขตข้อมูล	คำอธิบาย
1	TypeLG	Text	กำหนดภาษาที่ใช้หลักการของชนิดข้อมูลในระเบียนนี้
2	Token_id	Text	รหัสโทเคนที่เป็นชนิดของข้อมูล
3	Size	Byte	ขนาดของหน่วยความจำที่ใช้เก็บ
4	Description	Text	คำอธิบาย เกี่ยวกับชนิดข้อมูล

ตารางที่ 3.3 แสดงโครงสร้างของตาราง Token

ลำดับ	ชื่อเขตข้อมูล	ชนิดของเขตข้อมูล	คำอธิบาย
1	Token_id	Text	รหัสโทเคน หมายถึงคำเฉพาะ ตัวดำเนินการ คำสั่ง ชนิดของข้อมูล หรือส่วนสำคัญต่างของภาษาซี ที่เครื่องมือสร้างกราฟควบคุมกระแส และข้อมูลทดสอบนี้จะนำมาใช้
2	Command_f	Yes/No	สถานะของโทเคน เป็นว่าเป็นคำสั่ง
3	Identify_f	Yes/No	สถานะของโทเคน เป็นชนิดของข้อมูล
4	Opr_f	Yes/No	สถานะของโทเคน เป็นเครื่องหมาย
5	Description	Text	คำอธิบายของโทเคนนี้

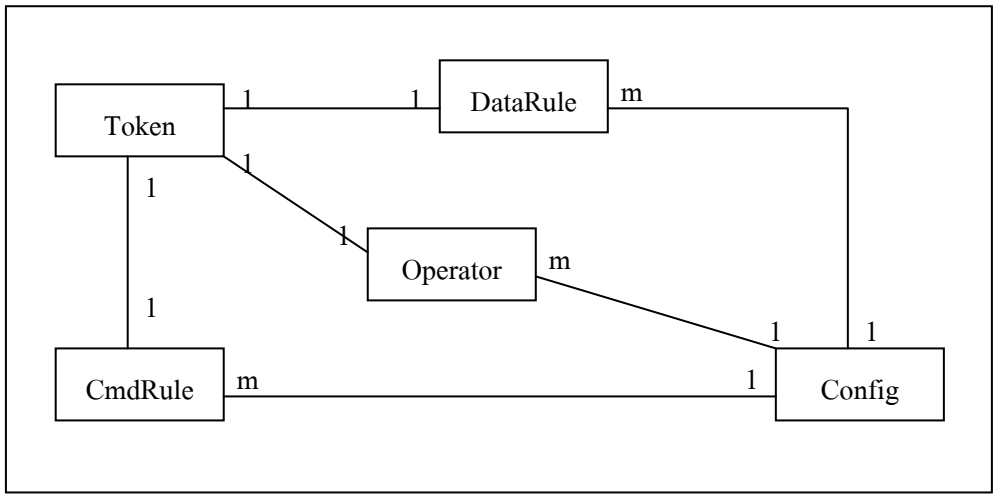
ตารางที่ 3.4 แสดงโครงสร้างของข้อกำหนดของคำสั่งควบคุม

ลำดับ	ชื่อเขตข้อมูล	ชนิดของเขตข้อมูล	คำอธิบาย
1	TypeLG	Text	กำหนดภาษาที่ใช้หลักการของคำสั่งในระเบียนนี้
2	Token_id	Text	รหัสโทเคน ที่เป็นคำสั่งควบคุมในภาษาซี
3	Token1	Text	รหัสโทเคน ที่ใช้ต่อจากคำสั่งตัวแรก
4	Token2	Text	รหัสโทเคน ที่ใช้ต่อจากคำสั่งตัวอื่น ๆ
5	TypeCmd	Byte	หมายเลขของคำสั่งควบคุม

ตารางที่ 3.5 แสดงโครงสร้างของตารางตัวดำเนินการ

ลำดับ	ชื่อเขตข้อมูล	ชนิดของเขตข้อมูล	คำอธิบาย
1	TypeLG	Text	กำหนดภาษาที่ใช้หลักการของชนิดข้อมูลในระเบียนนี้
2	Token_id	Text	รหัสโทเคนที่เป็นตัวดำเนินการ
3	TypeOpr	Byte	หมายเลขกำกับชนิดของตัวดำเนินการ

จากโครงสร้างส่วนจัดเก็บข้อมูลพื้นฐานต่าง ๆ ของภาษาซีที่เครื่องมือนี้จะนำไปใช้พิจารณาในแต่ละส่วนการประมวลผล สามารถนำมาแสดงแผนรูปที่ 3.19 ความสัมพันธ์ของตารางแต่ละตัวได้ดังนี้



รูปที่ 3.19 แผนภาพแสดงเอนทิตีและความสัมพันธ์ของตารางในฐานข้อมูล

บทที่ 4
ผลการดำเนินการวิจัยและการอภิปรายผล

งานวิจัยนี้มุ่งเน้นที่จะพัฒนาเครื่องมือที่สามารถช่วยอำนวยความสะดวกในกระบวนการทดสอบซอฟต์แวร์ให้สามารถทำการทดสอบซอฟต์แวร์ได้สะดวกขึ้น กล่าวคืองานวิจัยนี้จะพัฒนาเครื่องมือที่สามารถรับซอร์สโค้ดที่พัฒนาด้วยภาษาซีเข้าสู่เครื่องมือ แล้วเครื่องมือจะนำซอร์สโค้ดนั้นมาแยกส่วนพิจารณาเพื่อนำเสนอออกมาในรูปแบบของกราฟควบคุมกระแสที่สามารถแสดงให้เห็นผังการทำงานของซอร์สโค้ด จากนั้นเครื่องมือต้องแสดงเส้นทางอิสระของโปรแกรม เพื่อให้เห็นเส้นทางสำหรับทดสอบ และเครื่องมือยังต้องสามารถสร้างข้อมูลทดสอบที่สอดคล้องกับเส้นทาง เพื่อเตรียมไว้ให้สำหรับผู้ทดสอบจะได้นำไปใช้ในการทดสอบซอร์สโค้ดนั้นต่อไปได้

ฉะนั้นผลการดำเนินการของการวิจัยนี้ถูกแบ่งเป็น 5 ส่วนเช่นเดียวกัน นั่นคือ

4.1 ส่วนนำเข้าซอร์สโค้ดและใช้ติดต่อกับผู้ใช้

4.2 ส่วนกระจายนิพจน์

4.3 ส่วนสร้างกราฟควบคุมกระแส

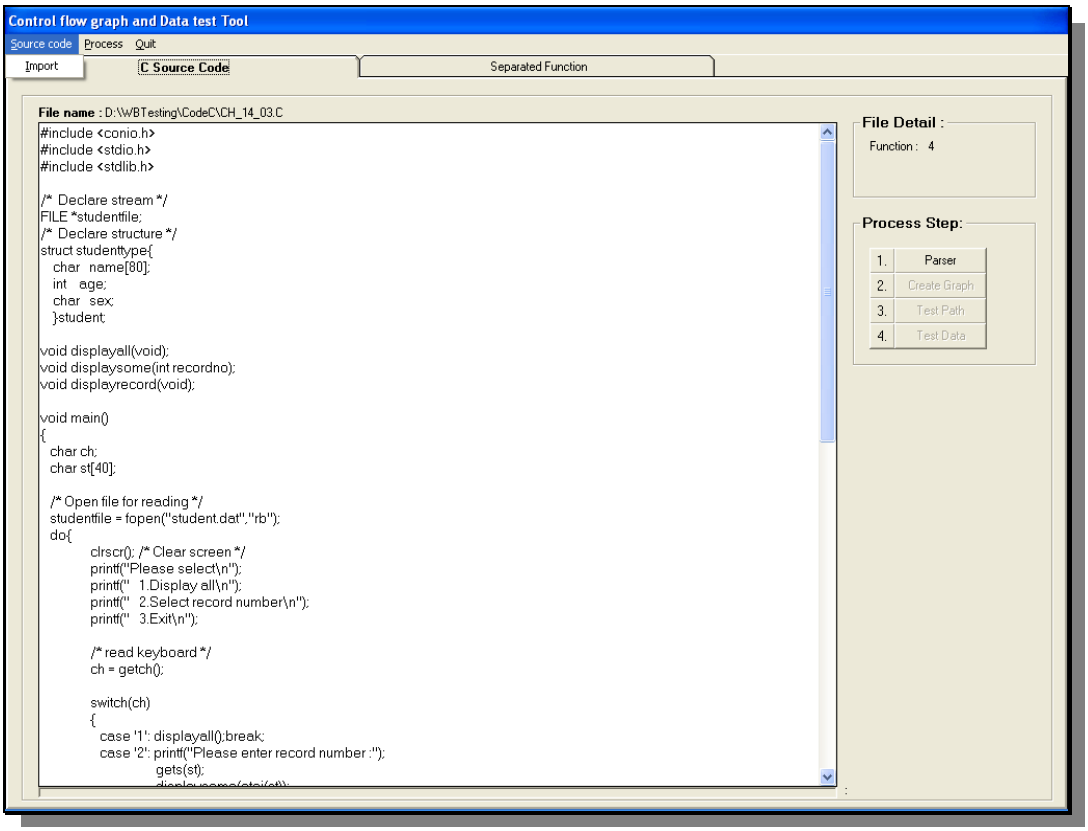
4.4 ส่วนสร้างเส้นทางทดสอบ

4.5 ส่วนสร้างข้อมูลทดสอบ

4.1 ส่วนนำเข้าซอร์สโค้ดและใช้ติดต่อกับผู้ใช้

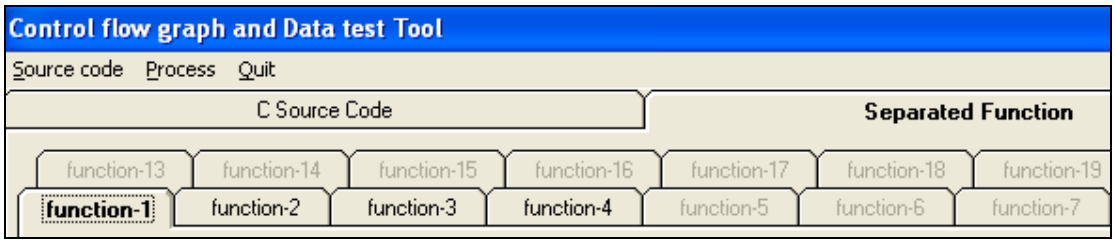
เครื่องมือส่วนนี้จะทำหน้าที่นำเข้าซอร์สโค้ดและแสดงผลการประมวลการทำงานต่อผู้ทดสอบโดยลักษณะการแสดงผลคือแสดงซอร์สโค้ดทั้งหมดที่นำเข้าในส่วนของหน้าจอ “C Source code” และเมื่อผ่านการประมวลแยกส่วนของซอร์สโค้ดจะแสดงรายละเอียดของโค้ดนั้นว่ามีกี่ฟังก์ชันย่อย ซอร์สโค้ดที่ถูกแยกฟังก์ชันออกเป็นส่วนย่อย ๆ จะถูกกระจายให้ไปแสดงในแต่ละหน้าย่อย ๆ ที่อยู่ในส่วนหน้าจอ “Separated Function” เพื่อให้การนำเสนอผลลัพธ์ที่ได้ต่อ ผู้ทดสอบให้เห็นภาพได้คลอบคลุมทุกส่วนของผลลัพธ์ที่เครื่องมือนี้จะสร้างของแต่ละฟังก์ชันย่อย ๆ (กรณีที่ในซอร์สโค้ดมีฟังก์ชันย่อย ๆ ภายใน (user-defined) ที่ถูกสร้างขึ้น) จึงมีการนำเสนอทั้งกราฟควบคุมกระแส เส้นทางทดสอบและข้อมูลทดสอบให้อยู่ภายในหน้าจอเดียวกัน เนื่องจากพื้นที่ของหน้าจอมีจำกัดจึงไม่สามารถแสดงแต่ละส่วนได้เต็มความสูงหรือกว้างได้

จากรูปที่ 4.1 เป็นการแสดงผลลัพธ์ของการทำงานเมื่อมีการนำซอร์สโค้ด เข้าสู่เครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบ โดยเมื่อซอร์สโค้ดที่นำเข้าถูกต้องตามที่เครื่องมือกำหนด เครื่องมือจะแสดงว่าซอร์สโค้ดนี้ประกอบด้วยฟังก์ชันย่อย ๆ กี่ฟังก์ชันที่ส่วน “File Detail” จากนั้นเครื่องมือจะอนุญาตให้ผู้ทดสอบดำเนินการขั้นตอนต่อไปได้ตามลำดับ ซึ่งมีทั้งหมด 4 ขั้นตอนคือ “Parser”, “Create Graph”, “Test Path” และ “Test Data” ซึ่งปุ่มสั่งงานเหล่านี้จะแสดงเมื่อขั้นตอนก่อนหน้าประมวลผลเสร็จสิ้น



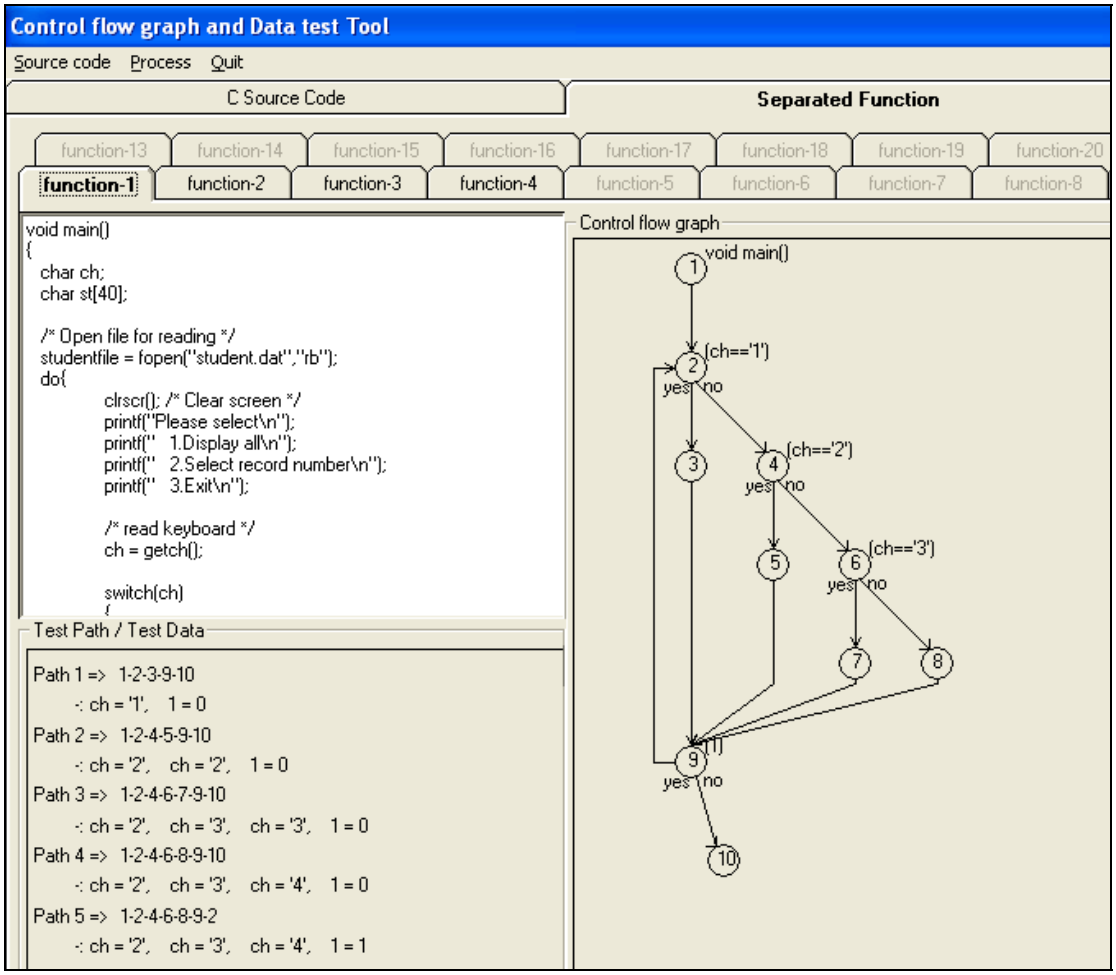
รูปที่ 4.1 แสดงผลการทำงานเมื่อมีการนำเข้าซอร์สโค้ดภาษาซี

จากรูปที่ 4.2 เป็นการแสดงผลการทำงานในขั้นตอน “Parser” เครื่องมือมีการประมวลผลแยกซอร์สโค้ดออกเป็นฟังก์ชันย่อย ๆ และกระจายให้ไปแสดงในแต่ละหน้าย่อย ๆ ที่อยู่ในส่วนหน้าจอ “Separated Function” โดยกำหนดให้สามารถทำงานได้เฉพาะหน้าจอที่ถูกแสดงฟังก์ชันย่อย ซึ่งจะเท่ากับจำนวนของฟังก์ชันย่อยที่มีในซอร์สโค้ดนั่นเอง

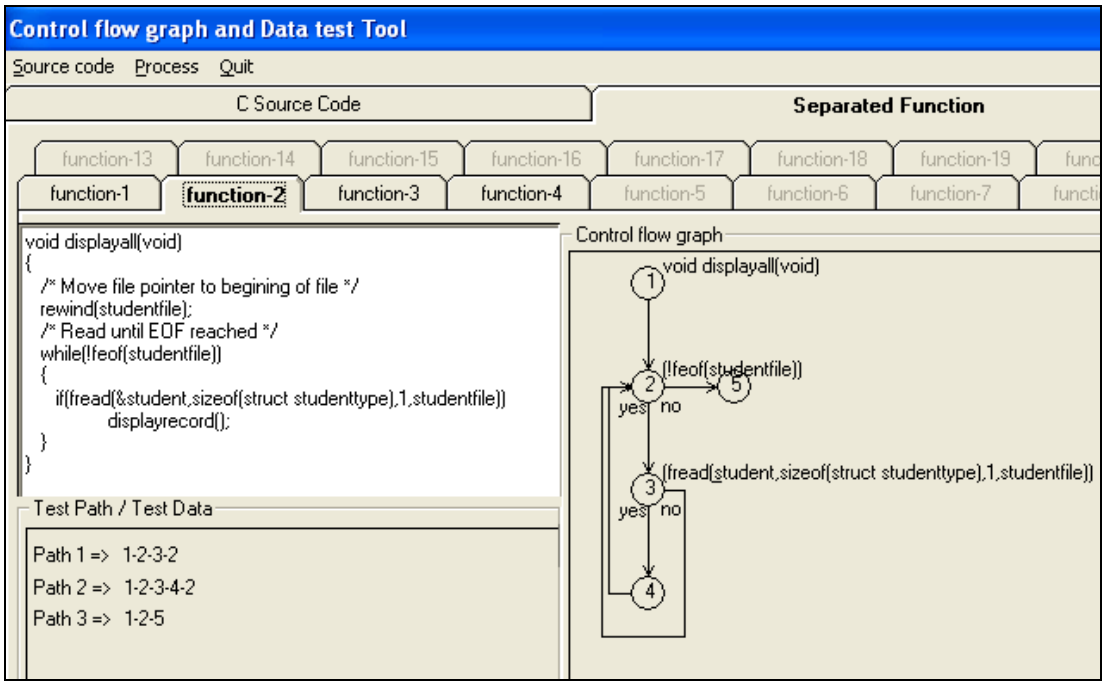


รูปที่ 4.2 แสดงผลการทำงานเมื่อเครื่องมือแยกฟังก์ชัน

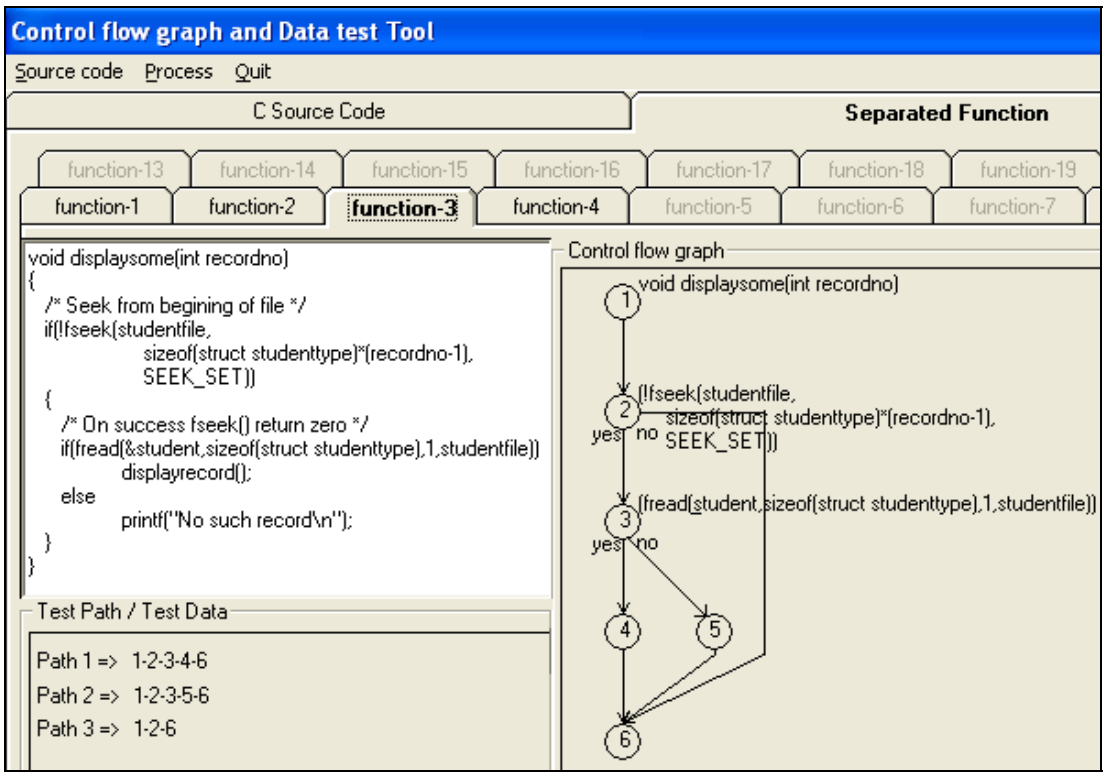
และเมื่อมีการทำงานในขั้นตอน “Create Graph” และ “Test Path” เครื่องมือจะสร้างกราฟควบคุมกระแส และสร้างเส้นทางทดสอบ ของแต่ละฟังก์ชัน ดังรูปที่ 4.3



รูปที่ 4.3 แสดงผลการสร้างกราฟและเส้นทางทดสอบของฟังก์ชันที่ 1 จากภาพที่ 4.2

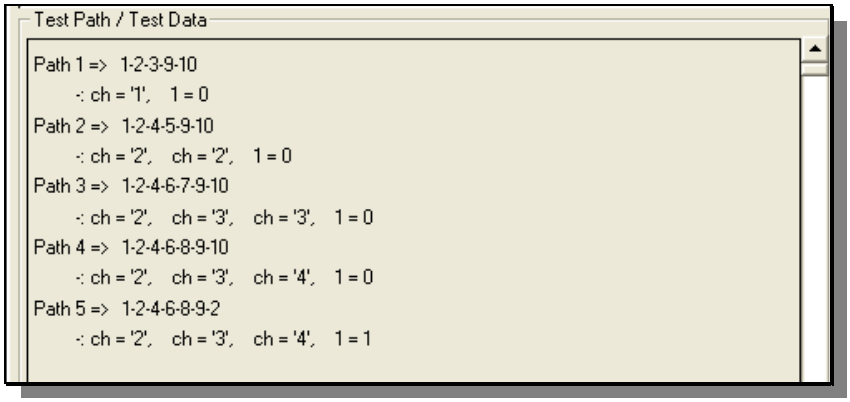


รูปที่ 4.4 แสดงผลการสร้างกราฟและเส้นทางทดสอบของฟังก์ชันที่ 2 จากรูปที่ 4.2

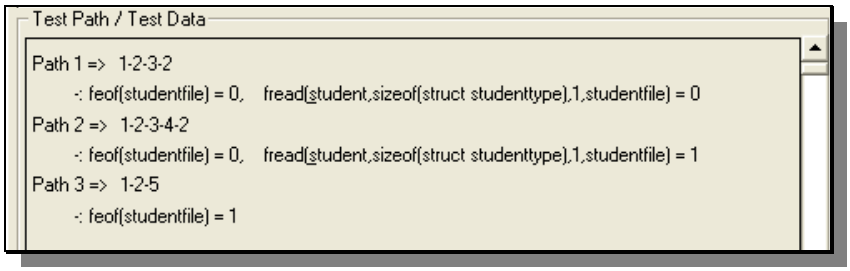


รูปที่ 4.5 แสดงผลการสร้างกราฟและเส้นทางทดสอบของฟังก์ชันที่ 3 จากรูปที่ 4.2

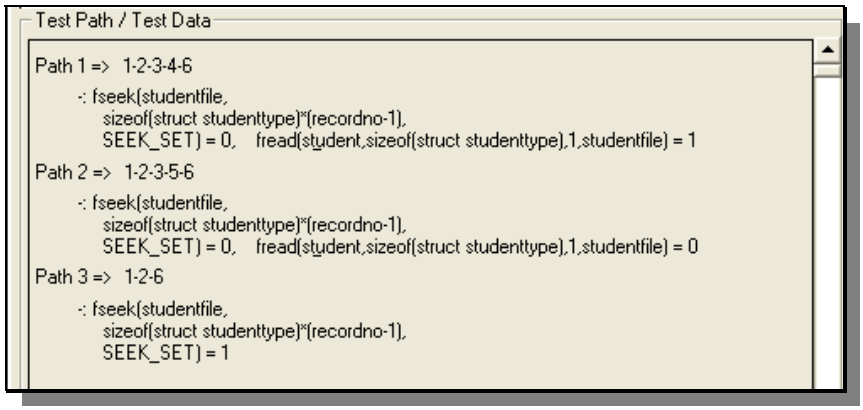
ส่วนสุดท้าย “Test Data” เครื่องมือจะสร้างข้อมูลทดสอบให้สอดคล้องกับเส้นทางทดสอบ ดังรูปที่ 4.6 รูปที่ 4.7 และรูปที่ 4.8



รูปที่ 4.6 แสดงข้อมูลทดสอบสำหรับเส้นทางทดสอบจากรูปที่ 4.3



รูปที่ 4.7 แสดงข้อมูลทดสอบสำหรับเส้นทางทดสอบจากรูปที่ 4.4



รูปที่ 4.8 แสดงข้อมูลทดสอบสำหรับเส้นทางทดสอบจากรูปที่ 4.5

ในส่วนแสดงเส้นทางทดสอบและข้อมูลทดสอบนั้นในการทำงานจะแสดงเส้นทางทดสอบก่อน เมื่อผู้ทดสอบเลือกคลิกที่ “Test path” ส่วนข้อมูลทดสอบนั้นจะปรากฏหลังจากที่ผู้ทดสอบสั่งให้สร้างข้อมูลทดสอบ

4.2 ส่วนกระจายนิพจน์

ในส่วนกระจายนิพจน์นี้เป็นการเตรียมข้อมูลหรือเก็บข้อมูลไว้สำหรับส่วนสร้างกราฟควบคุมกระแส สร้างเส้นทางทดสอบและข้อมูลทดสอบก็ว่าได้ ผลการทำงานของส่วนนี้จะไม่แสดงออกมาให้ผู้ทดสอบได้เห็นอย่างเป็นทางการเหมือนกับส่วนอื่น ๆ ผลลัพธ์ของส่วนนี้จะเป็นข้อมูลที่ถูกเก็บไว้ในส่วนเก็บข้อมูลแบบแถวลำดับ ที่อยู่ภายในของระบบและข้อมูลเหล่านี้จะถูกลบทิ้งไปเองเมื่อมีการเริ่มทำงานนำเข้าซอร์สโค้ดใหม่หรือเมื่อจบการทำงาน

เพื่อให้เข้าใจ การทำงานของเครื่องมือจะแสดงข้อมูลที่เครื่องมือนี้เก็บไว้เมื่อการประมวลผลของส่วนกระจายนิพจน์นี้เสร็จสิ้น

```
#include <stdio.h>
void main(void){
 int testnum;
 do{
     printf("Put value over 100 or 50 to stop\n");
     scanf("%d",&testnum);
     if(testnum == 50) break;  /* Stop looping when enter 50 */
     printf("Hello ,you did\'t enter value 50\n");
     if(testnum == 0) continue;
     printf("Hello ,you did\'t enter value 0\n");
  }while (testnum <= 100);    /* Infinite loop */
 do{
     printf("Enter 0 to stop\n");
     scanf("%d",&testnum);
     if(testnum == 0) break;  /* Exit from loop when enter 0 */
  }while(1);
}
```

รูปที่ 4.9 แสดงตัวอย่างซอร์สโค้ดที่นำเข้าสู่เครื่องมือ

4.2.1 กรองฟังก์ชัน

เมื่อซอร์สโค้ดที่นำเข้าสู่เครื่องมือผ่านการประมวลผลในส่วนกระจายนิพจน์แล้ว ลำดับต่อไปซอร์สโค้ดที่ถูกแยกเป็นฟังก์ชันย่อยที่เก็บไว้ในตัวแปรแถวลำดับ จะถูกนำไปกรองเพื่อให้เหลือเฉพาะส่วนที่จำเป็นต้องใช้ เพื่อจะได้ง่ายต่อการประมวลผล

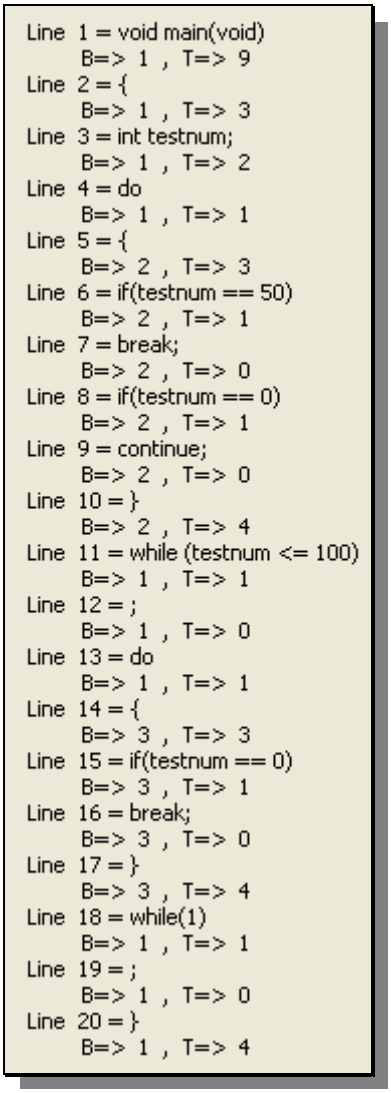
```
void main(void)
{
 int testnum;

 do{
     printf("Put value over 100 or 50 to stop\n");
     scanf("%d",&testnum);
     if(testnum == 50) break;  /* Stop looping when enter 50 */
     printf("Hello ,you did\'t enter value 50\n");
     if(testnum == 0) continue;
     printf("Hello ,you did\'t enter value 0\n");
  }while (testnum <= 100);

  /* Infinite loop */
 do{
     printf("Enter 0 to stop\n");
     scanf("%d",&testnum);
     if(testnum == 0) break;  /* Exit from loop when enter 0 */
  }while(1);
}
```

รูปที่ 4.10 แสดงซอร์สโค้ดจากรูปที่ 4.1 ที่ถูกแยกเป็นฟังก์ชันย่อย เพื่อแสดงในแต่ละหน้า

ในส่วนของการกรองฟังก์ชันได้มีการออกแบบส่วนจัดเก็บข้อมูลแบบแถวลำดับไว้สำหรับรองรับข้อมูลชื่อ *Line2Node(index-func,index-line)*, *BofLine(index-func,index-line)* และ *TypeLine(index-func,index-line)* สำหรับเก็บซอร์สโค้ดที่กรองแล้ว โดยมีการพิจารณาทีละบรรทัด คำว่าบรรทัดในเครื่องมือนี้คือประโยคคำสั่งหรือส่วนของคำสั่งที่จะต้องใช้ในการประมวลผลในส่วนอื่น ๆ ของเครื่องมือนี้ ฉะนั้นเมื่อผ่านการประมวลผลการกรองฟังก์ชันระบบภายในจะมีการเก็บข้อมูลที่แยกได้ลงไปเก็บที่ *Line2Node*, *BofLine* และ *TypeLine* ดังรูปที่ 4.11

จากรูปที่ 4.11 เป็นเพียงการแสดงผลข้อมูลที่ถูกเก็บไว้ในระบบหลังผ่านการกรองฟังก์ชัน ซึ่งในการทำงานจริงผู้ทดสอบจะมองไม่เห็นข้อมูลส่วนนี้ โดยจะมีการแสดงซอร์สโค้ดแต่ละบรรทัดพร้อมทั้งเก็บข้อมูลไว้ด้วยว่าแต่ละบรรทัดอยู่ในขอบเขตของบล็อกไหน ซึ่งจะมีผลกับตัวแปรที่มีการประกาศภายในบล็อกและมีการแยกประเภทของซอร์สโค้ดแต่ละบรรทัดที่จัดเก็บไว้ด้วย ซึ่งชนิดของบรรทัดจะมีชื่อฟังก์ชัน เปิดบล็อก ปิดบล็อก การประกาศตัวแปร คำสั่งควบคุม และคำสั่งอื่น ๆ ทั่วไปที่จะถูกดำเนินการเมื่อเงื่อนไขเป็นจริง-เท็จตามความเหมาะสมที่แยกออกได้ ซึ่งจะเก็บไว้ใน *Line2Node* ส่วนขอบเขตที่จะใช้ควบคุมการใช้งานตัวแปรจะถูกเก็บไว้ที่ *BofLine*

หรือที่เห็นในรูปก็คือ B=>... นั่นเอง และสุดท้ายคือชนิดของแต่ละบรรทัดจะเก็บไว้ที่ *TypeLine* ซึ่งในรูปจะเป็น T=>… นั่นเอง



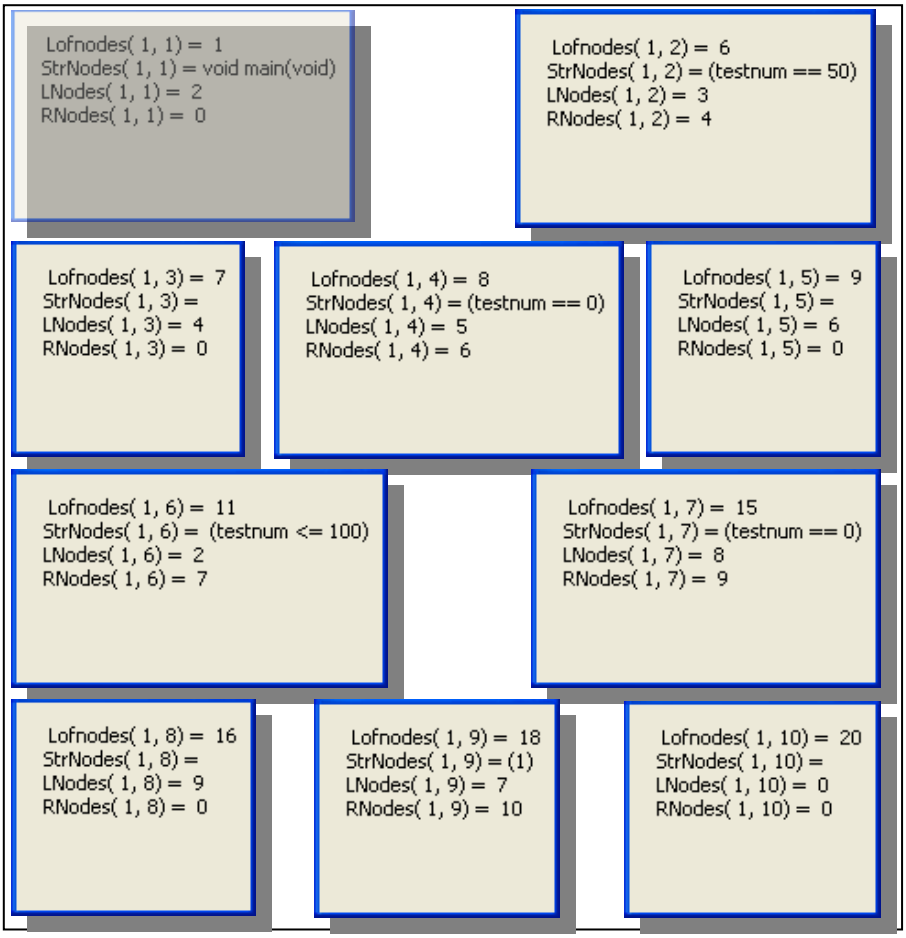
รูปที่ 4.11 แสดงข้อมูลซอร์สโค้ดที่ถูกเก็บไว้หลังกรองฟังก์ชัน

สำหรับในฟังก์ชันอื่น ๆ ข้อมูลที่ได้จากการกรองฟังก์ชันจะถูกเก็บไว้ในระบบในลักษณะที่แสดงในรูป และจะเก็บไว้ในทุกฟังก์ชัน

4.2.2 แยกโหนด

เมื่อสังเกตผลการเก็บข้อมูลซอร์สโค้ดที่กรองได้จากส่วนกรองฟังก์ชันจะพบว่าข้อมูลแต่ละบรรทัดนั้น ในบางบรรทัดประกอบด้วยคำสั่งควบคุม ซึ่งเป็นสิ่งสำคัญที่มีผลต่อโครงสร้างของเส้นทางทดสอบ ซึ่งหน้าที่หลักของส่วนแยกโหนดจะค้นหาว่าบรรทัดไหนที่เป็นโหนดและมีเงื่อนไขคืออะไรแล้วถ้าเงื่อนไขเป็นจริงโหนดต่อไปจะเป็นโหดใด ถ้าเงื่อนไขเป็นเท็จโหนดต่อไปจะเป็นโหนดใด ในส่วนแยกโหนดจะหาคำตอบเหล่านี้แล้วเก็บไว้ในตัวแปรแบบแถวลำดับที่ออกแบบไว้ซึ่งจะมี *StrNodes(index-func,index-node)*, *LNodes(index-func, index-node)* และ *RNodes(index-func , index-node)*

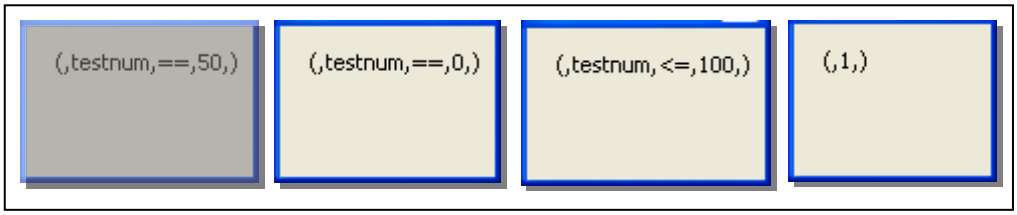
ผลลัพธ์จากการทำงานในส่วนนี้ผู้ทดสอบไม่เห็นการเก็บข้อมูลที่ได้ออกแบบไว้ทางหน้าจอเช่นเดียวกับส่วนกรองฟังก์ชัน แต่เพื่อให้เข้าใจการทำงานของระบบมากขึ้นจะแสดงการเก็บข้อมูลหลังการประมวลผลแยกโหนดดังนี้



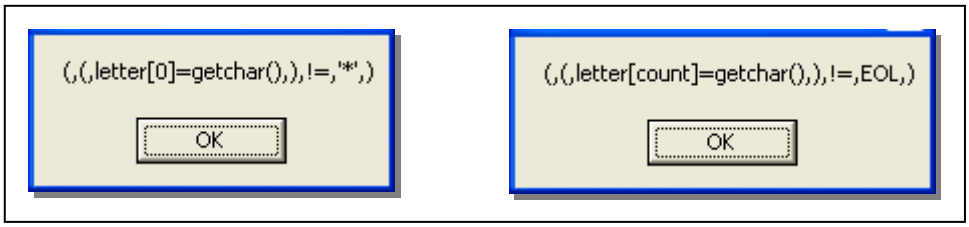
รูปที่ 4.12 แสดงข้อมูลโหนดแต่ละโหนดที่เก็บไว้หลังการแยกโหนด

4.2.3 แยกโทเคน

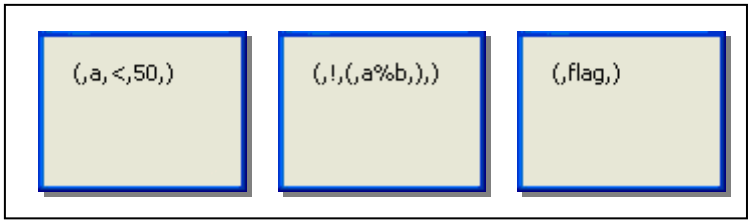
ผลลัพธ์จากการทำงานในส่วนนี้ผู้ทดสอบจะไม่เห็นการเก็บข้อมูลที่ได้ออกแบบไว้ทางหน้าจอเช่นเดียวกับส่วนกรองฟังก์ชัน แต่เพื่อให้เข้าใจการทำงานของระบบมากขึ้นจะแสดงการเก็บข้อมูลหลังการประมวลผลแยกโทเคนดังนี้



รูปที่ 4.13 แสดงตัวอย่างข้อมูลโทเคนแต่ละตัวที่จัดเก็บไว้หลังผ่านการแยกโทเคน



รูปที่ 4.14 แสดงตัวอย่างข้อมูลโทเคนแต่ละตัวที่จัดเก็บไว้หลังผ่านการแยกโทเคน



รูปที่ 4.15 แสดงตัวอย่างข้อมูลโทเคนแต่ละตัวที่จัดเก็บไว้หลังผ่านการแยกโทเคนแล้ว

จากรูปข้อมูลโหนดแต่ละโหนด จะถูกนำเข้าสู่ส่วนแยกโทเคนเฉพาะโหนดที่เป็นเงื่อนไข และเมื่อเงื่อนไขผ่านการแยกโทเคนจะมีการจัดเก็บข้อมูลโดยแยกโทเคน ที่เครื่องมือสนใจออกจากกัน โดยรูปที่แสดงจะมีเครื่องหมายจุลภาคคั่นระหว่างโทเคนแต่ละตัว

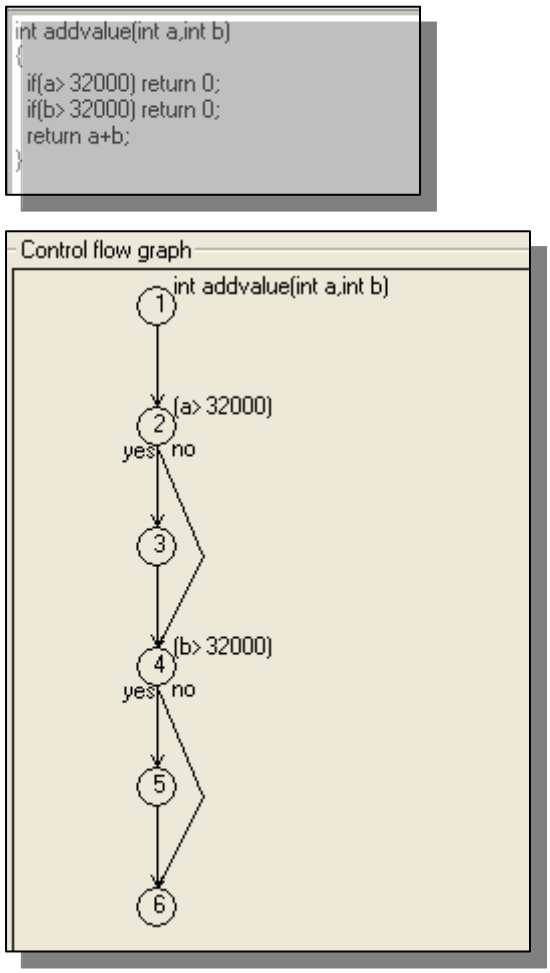
4.3 ส่วนสร้างกราฟควบคุมกระแส

ในส่วนสร้างกราฟควบคุมกระแส เครื่องมือจะนำข้อมูลจากส่วนต่าง ๆ มาใช้ในการสร้างกราฟ โดยกราฟที่สร้างจะสอดคล้องกับโครงสร้างของคำสั่งภายในซอร์สโค้ดแต่ละฟังก์ชัน โดยสามารถจำแนกได้ดังนี้

4.3.1 คำสั่งควบคุมแบบเลือก (Branching)

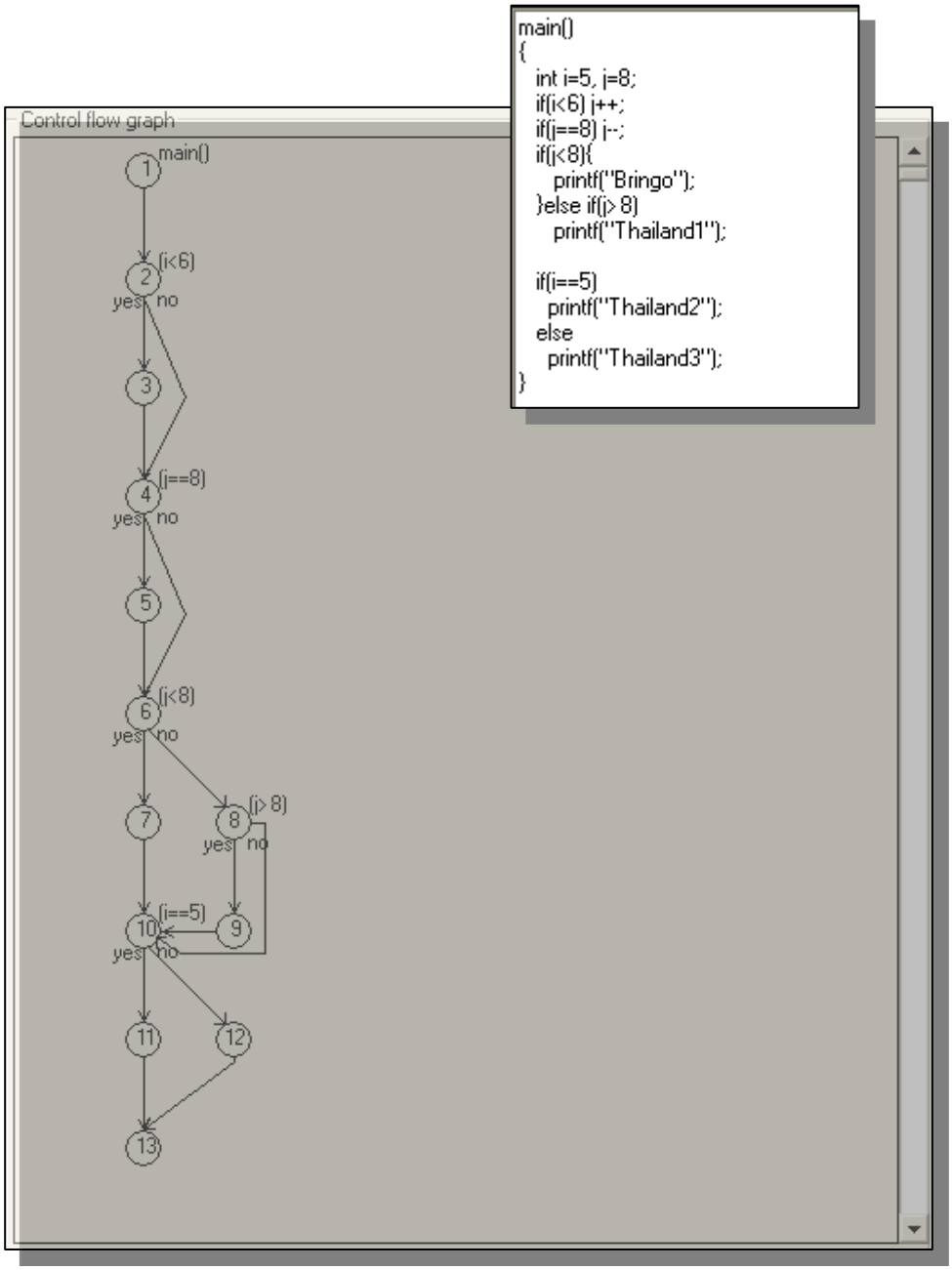
มีลักษณะเป็นคำสั่งที่ใช้การตรวจสอบผลลัพธ์ที่ได้จากนิพจน์ตรรกะ แล้วเลือกทำคำสั่ง หรือกลุ่มคำสั่งแบ่งเป็น 3 แบบคือ

คำสั่งควบคุมแบบหนึ่งทางเลือก (if statement) จากรูปที่ 4.16 เป็นตัวอย่างคำสั่งควบคุมแบบหนึ่งทางเลือก โดยจากการูปตัวอย่างซอร์สโค้ดเมื่อสร้างเป็นกราฟควบคุมกระแสแล้วจะได้ ดังรูปที่ 4.16



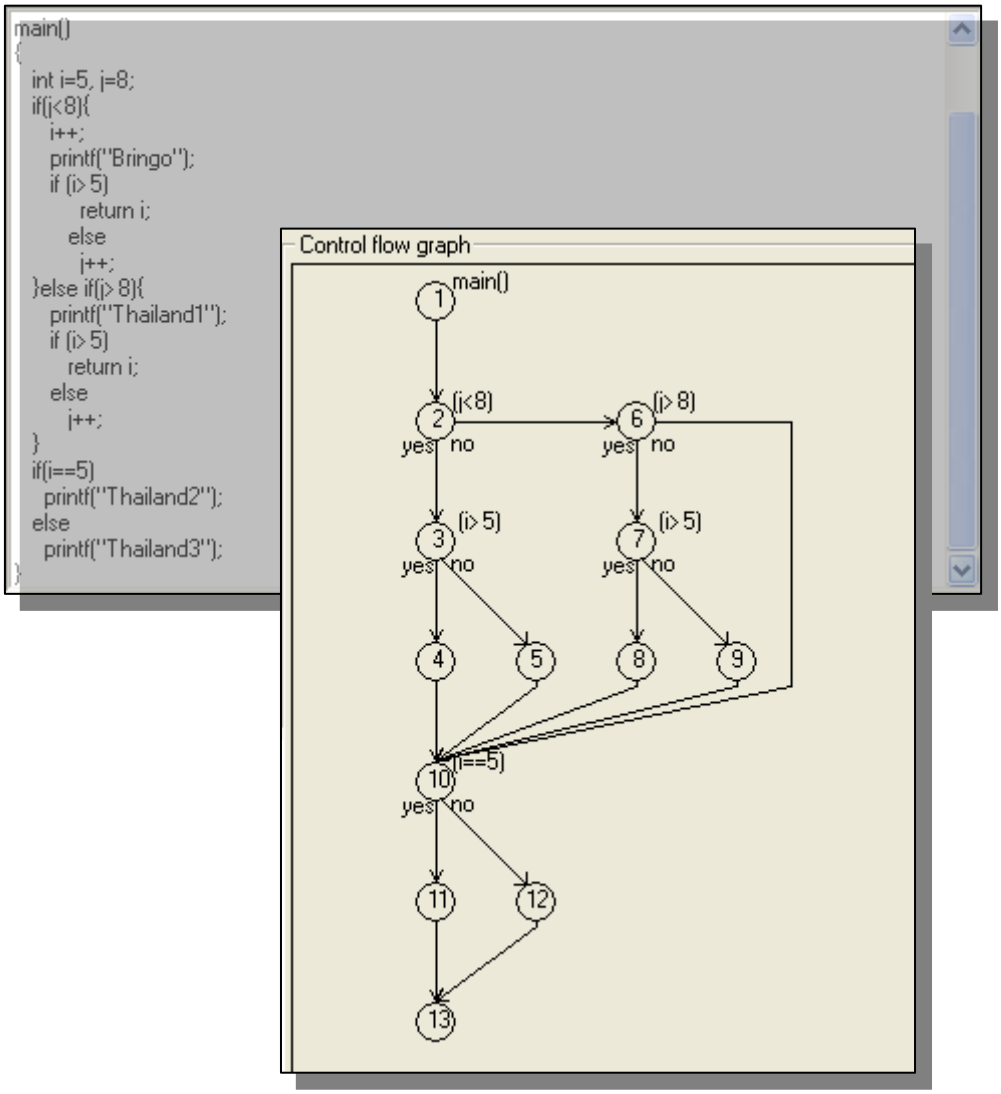
รูปที่ 4.16 แสดงซอร์สโค้ดที่มีคำสั่งควบคุมแบบหนึ่งทางเลือกและกราฟควบคุมกระแส

คำสั่งควบคุมแบบสองเลือก (if-else statement) จากรูปที่ 4.17 เป็นตัวอย่างคำสั่งควบคุมแบบสองทางเลือกและกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีชุดคำสั่งควบคุมแบบสองทางเลือกแต่สำหรับตัวอย่างนี้อาจจะมีชุดคำสั่งแบบหนึ่งทางเลือกรวมอยู่ด้วย



รูปที่ 4.17 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีชุดคำสั่งควบคุมแบบสองทางเลือก

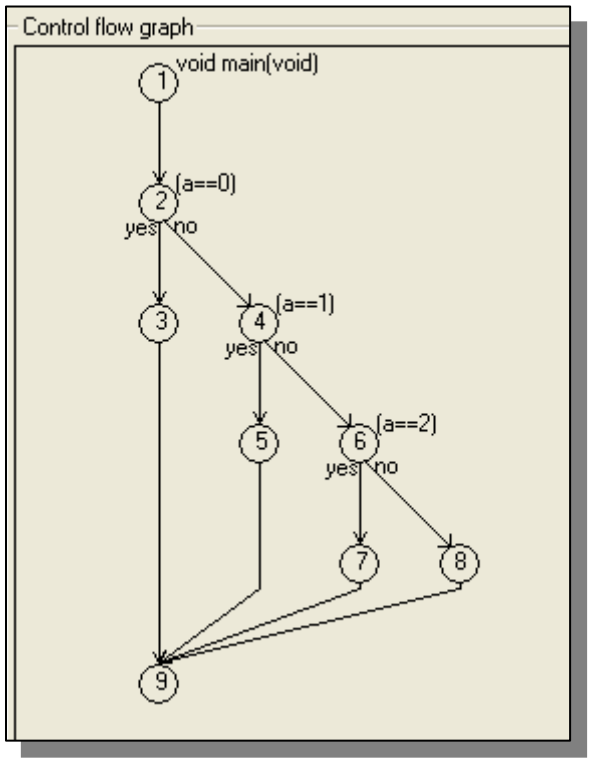
คำสั่งควบคุมแบบหลายทางเลือก (nested-if statement) จากรูปที่ 4.18 เป็นตัวอย่างคำสั่งควบคุมแบบหลายทางเลือก และกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีชุดคำสั่งควบคุมแบบหลายทางเลือก



รูปที่ 4.18 กราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีชุดคำสั่งควบคุมแบบหลายทางเลือก

คำสั่งควบคุมแบบหลายทางเลือก (case statement) คำสั่งควบคุมแบบหลายทางเลือกอีกแบบหนึ่งคือ case statement จากรูปที่ 4.19 เป็นตัวอย่างซอร์สโค้ดที่เป็นคำสั่งควบคุมแบบหลายทางเลือก case statement และกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีชุดคำสั่งควบคุมแบบหลายทางเลือก case statement

```
void main(void)
{
 int a;

 printf("Please enter value :");
 scanf("%d",&a);

 switch(a)
  {
    case 0 : printf("This value is 0\n");break;
    case 1 : printf("This value is 1\n");break;
    case 2 : printf("This value is 2\n");break;
    default : printf("Please enter value between 0 and 2\n");
  }
}
```

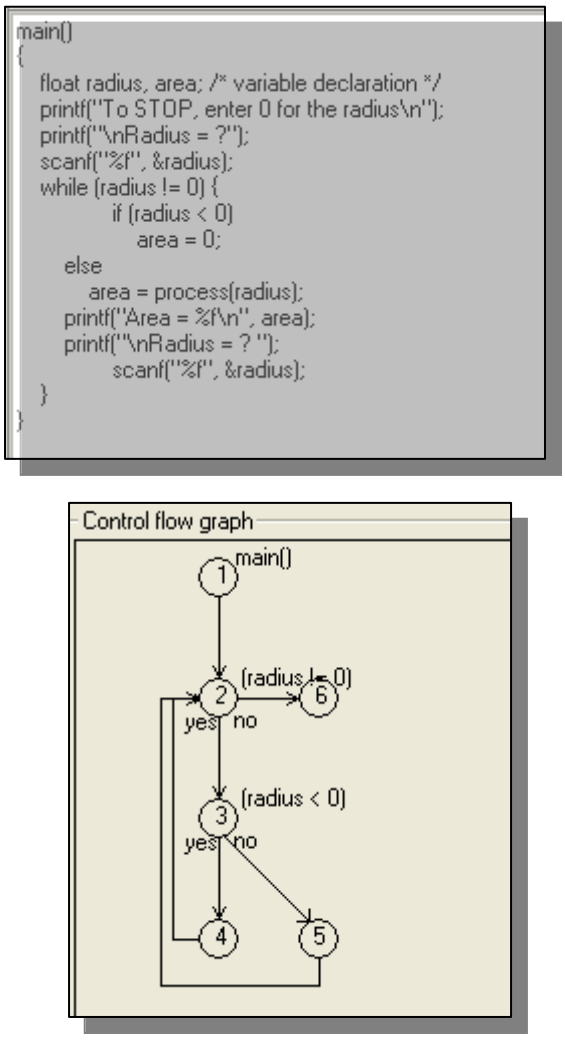


รูปที่ 4.19 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมแบบหลายทางเลือก case statement

4.3.2 คำสั่งควบคุมแบบวนซ้ำ (Looping)

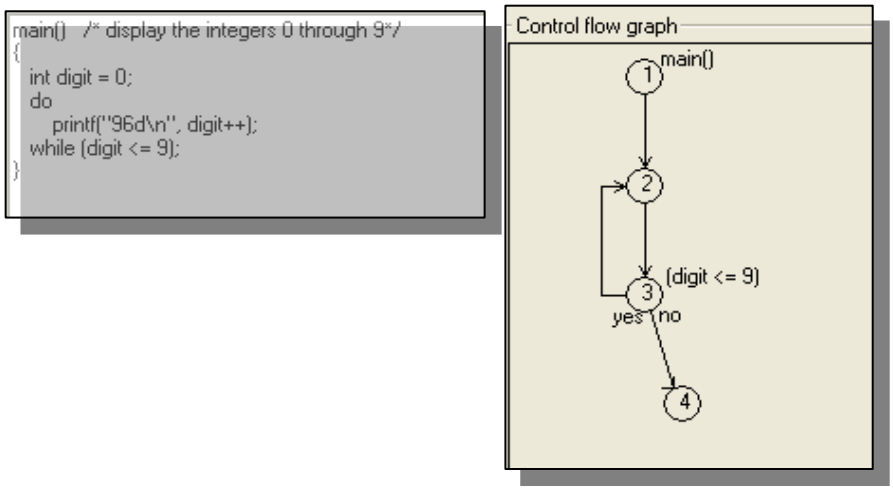
มีลักษณะเป็นคำสั่งที่ใช้ต้องทำชุดคำสั่งที่เป็นกิจกรรมเดียวกันหลายครั้งซึ่งในภาษาโปรแกรมเราเรียกว่า ลูป (loop) ในภาษาซีมีชุดคำสั่งที่ใช้ในการควบคุมการทำงานลักษณะดังกล่าวดังนี้

คำสั่งวนซ้ำแบบ while (while statement) จากรูปที่ 4.20 เป็นตัวอย่างคำสั่งควบคุมวนซ้ำแบบ while และกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีชุดคำสั่งควบคุมวนซ้ำแบบ while



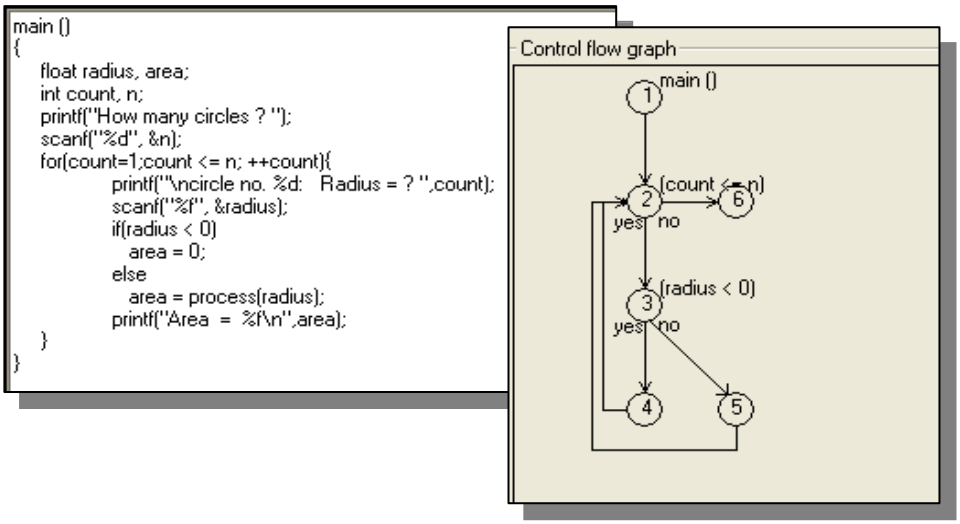
รูปที่ 4.20 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมวนซ้ำแบบ while

คำสั่งวนซ้ำแบบ do-while (do-while statement) จากรูปที่ 4.21 เป็นตัวอย่างคำสั่งควบคุมวนซ้ำแบบ do-while และกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีชุดคำสั่งควบคุมวนซ้ำแบบ do-while



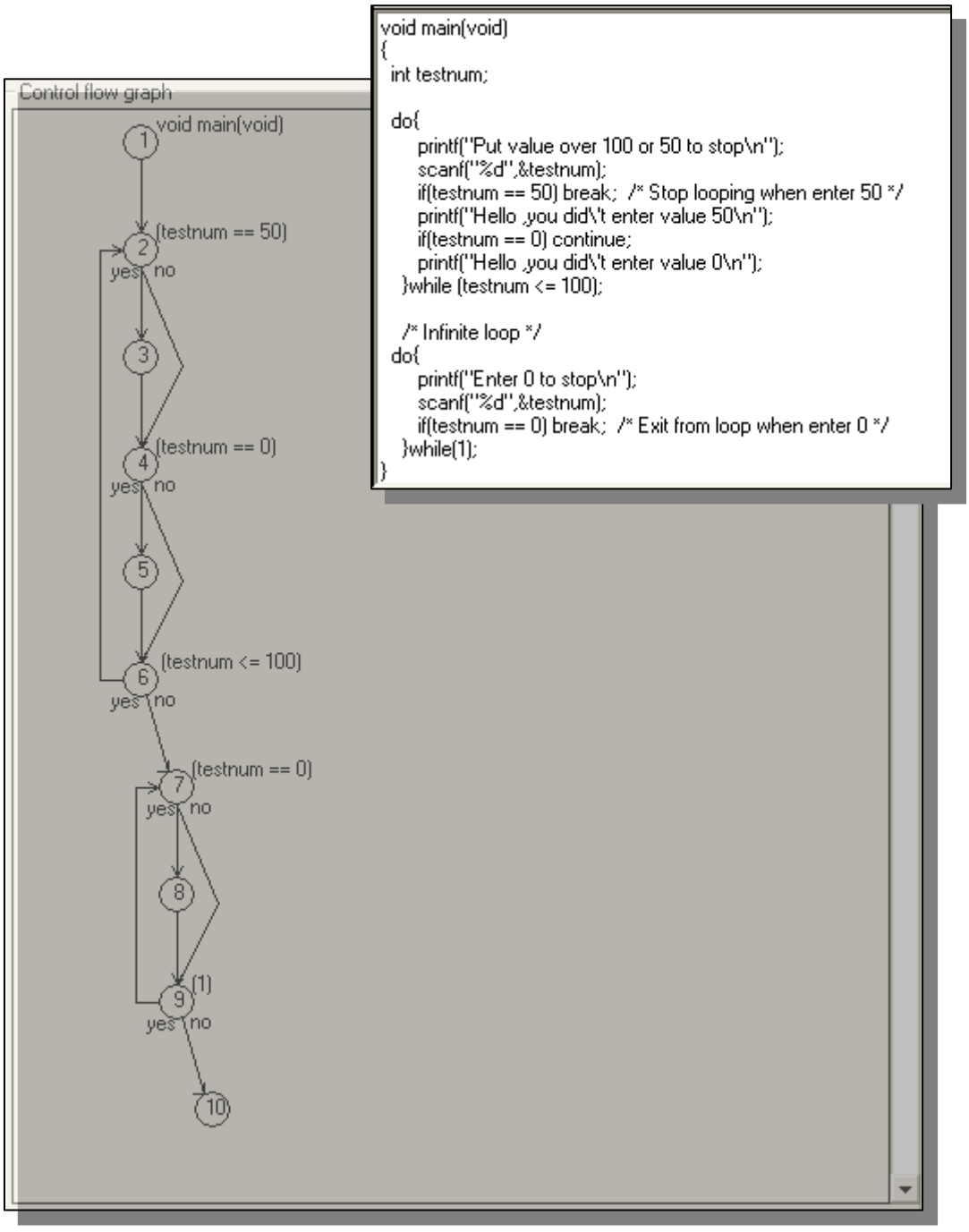
รูปที่ 4.21 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมวนซ้ำแบบ do-while

คำสั่งวนซ้ำแบบ for (for statement) จากรูปที่ 4.22 เป็นตัวอย่างของคำสั่งควบคุมวนซ้ำแบบ for และกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีชุดคำสั่งควบคุมวนซ้ำแบบ for



รูปที่ 4.22 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมวนซ้ำแบบ for

นอกจากนี้ ยังมีซอร์สโค้ดที่มีการใช้ชุดคำสั่งหลายรูปแบบรวมอยู่ซอร์สโค้ดเดียวกันและจากรูปที่ 4.23 จะเป็นตัวอย่างซอร์สโค้ดที่มีคำสั่งควบคุมหลายรูปแบบรวมอยู่ด้วยกันและแสดงกราฟควบคุมกระแสที่เครื่องมือสร้างขึ้น



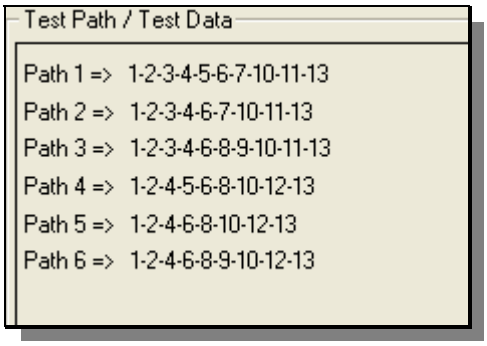
รูปที่ 4.23 แสดงกราฟควบคุมกระแสที่สร้างจากซอร์สโค้ดที่มีคำสั่งควบคุมแบบผสม

4.4 ส่วนสร้างเส้นทางทดสอบ

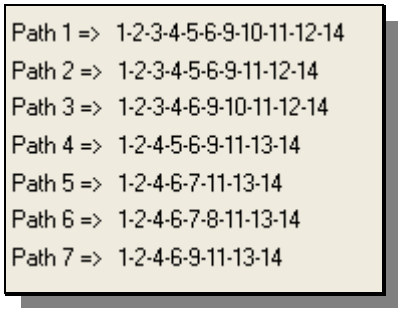
ผลลัพธ์ของส่วนสร้างเส้นทางทดสอบจะแสดงผลออกทางหน้าจอแสดงเส้นทางทดสอบที่เครื่องมือสร้างขึ้นเพื่อเตรียมไว้ให้สำหรับผู้ทดสอบ โดยเส้นทางทดสอบนี้จะครอบคลุมเส้นทางการทำงานทุกเส้นทางของซอร์สโค้ดที่นำเข้า

ลักษณะของเส้นทางทดสอบสอบจะสอดคล้องกับคำสั่งควบคุม กล่าวคือ

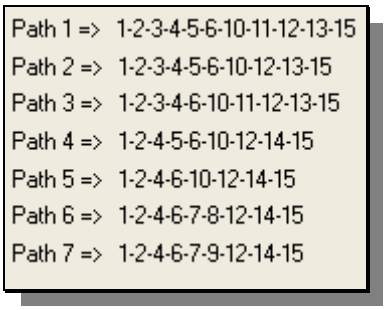
กรณีแรก ถ้าในซอร์สโค้ดเป็นคำสั่งควบคุมที่ไม่วนซ้ำทั้งหมดเส้นทางทดสอบจะสิ้นสุดที่โหนดสุดท้ายของกราฟควบคุมกระแส ของทุก ๆ เส้นทางทดสอบดังรูปที่ 4.24-4.46



รูปที่ 4.24 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 1

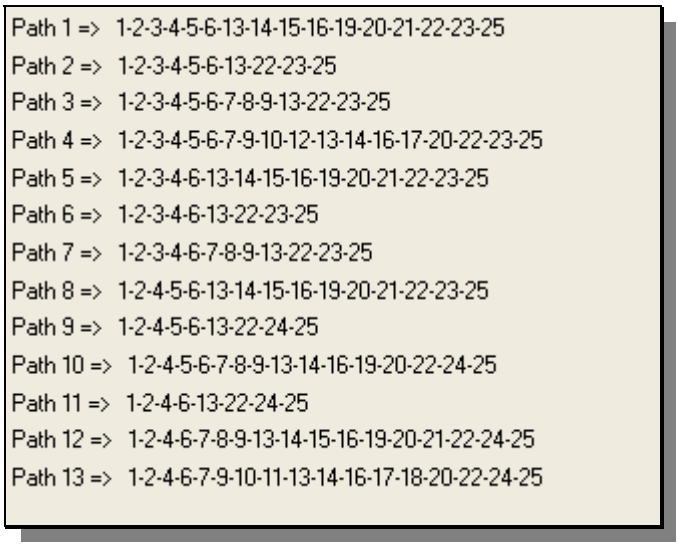


รูปที่ 4.25 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 2

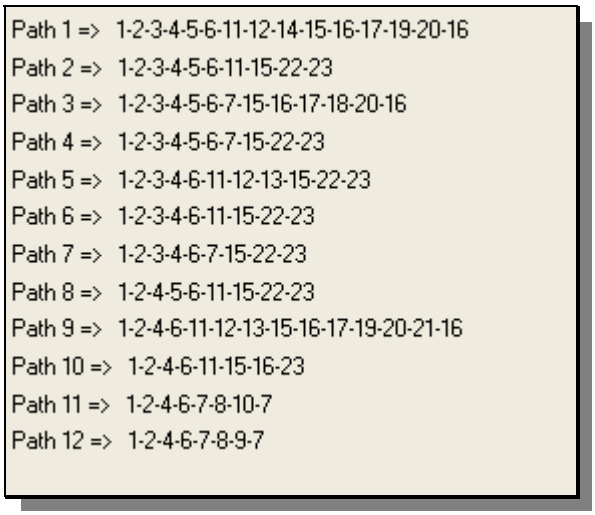


รูปที่ 4.26 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 3

กรณีถัดมา ถ้าในซอร์สโค้ดมีคำสั่งควบคุมแบบวนซ้ำอยู่ด้วยรูปแบบของเส้นทางทดสอบบางเส้นทางทดสอบจะสิ้นสุดที่โหนดที่เป็นคำสั่งควบคุมแบบวนซ้ำดังรูปที่ 4.27



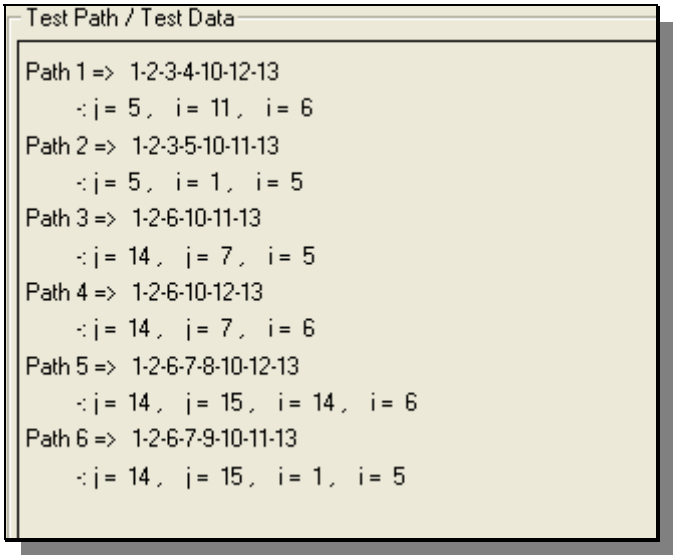
รูปที่ 4.27 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 4



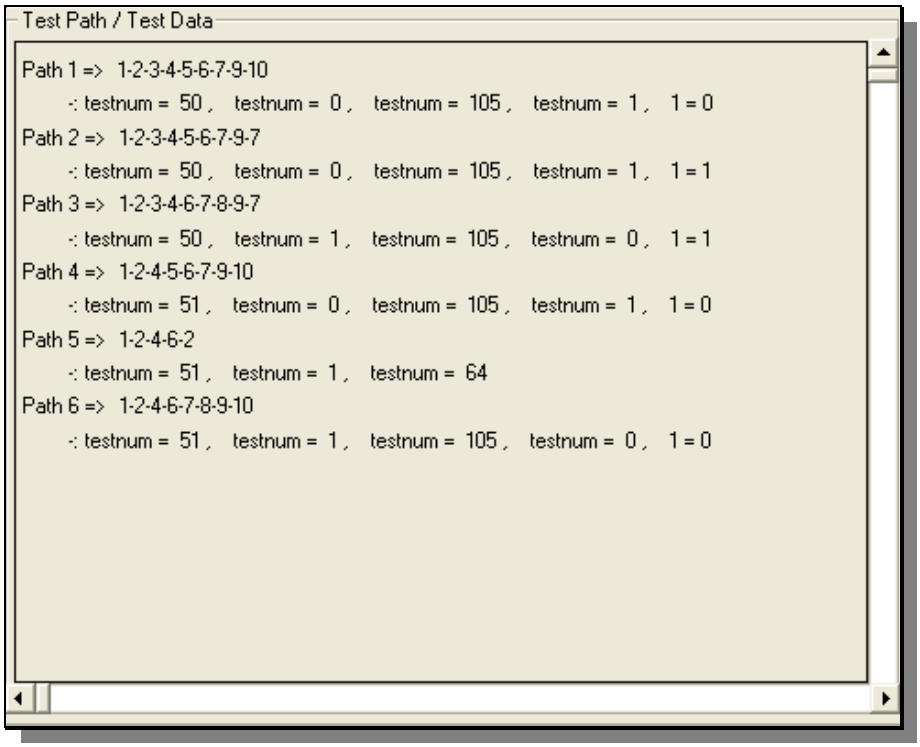
รูปที่ 4.28 แสดงตัวอย่างเส้นทางทดสอบแบบที่ 5

4.5 ส่วนสร้างข้อมูลทดสอบ

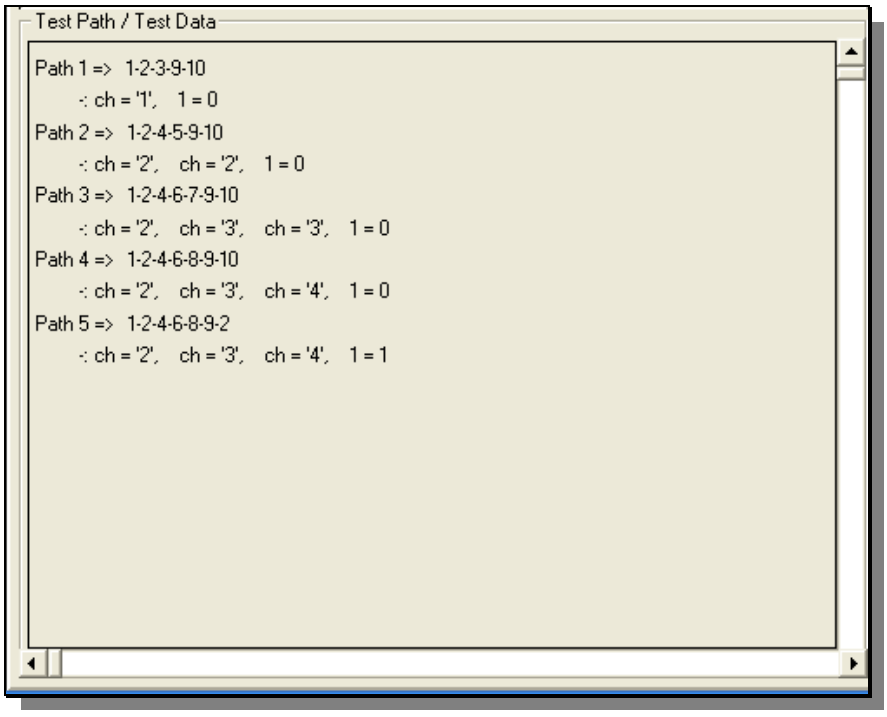
ผลลัพธ์ของส่วนสร้างข้อมูลทดสอบจะสร้างข้อมูลทดสอบให้กับแต่ละเส้นทาง โดยจะแสดงเฉพาะโทเคนที่กำหนดข้อมูลทดสอบให้เท่านั้น แสดงได้ดังรูปที่ 4.29-4.31



รูปที่ 4.29 แสดงตัวอย่างการสร้างข้อมูลทดสอบแบบที่ 1



รูปที่ 4.30 แสดงตัวอย่างการสร้างข้อมูลทดสอบแบบที่ 2



รูปที่ 4.31 แสดงตัวอย่างการสร้างข้อมูลทดสอบแบบที่ 3

จากผลลัพธ์ที่ได้จากเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบสำหรับภาษาซีที่ผู้วิจัยได้พัฒนาขึ้นนี้ พบว่าเครื่องมือสามารถนำเข้าซอร์สโค้ด ประมวลผลสร้างกราฟ แสดงเส้นทางทดสอบตามโครงสร้างการทำงานของซอร์สโค้ดและสร้างข้อมูลทดสอบเพื่อใช้สำหรับให้การทำงานของซอร์สโค้ดวิ่งไปตามเส้นทางทดสอบที่สร้างขึ้น ซึ่งจะช่วยอำนวยความสะดวกให้กับผู้ทดสอบ ผลลัพธ์ของเครื่องมือจะถูกนำเสนอทุกส่วนไว้ในหน้าจอเดียวกันเพื่อความสะดวกในการใช้งานของผู้ทดสอบ แต่มีข้อจำกัดในเรื่องของพื้นที่ของหน้าจอและการนำผลลัพธ์ที่ได้ออกจากเครื่องมือไปใช้งานต่อในภายหลัง

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

ในกระบวนการการพัฒนาซอฟต์แวร์ มีองค์ประกอบหลายส่วนด้วยกันและองค์ประกอบหนึ่งที่ทำให้ซอฟต์แวร์ที่ได้มีความถูกต้องและมีคุณภาพยิ่งขึ้นก็คือการทดสอบ (validation) เพื่อหาข้อผิดพลาดและการตรวจสอบ (verification) เพื่อยืนยันความถูกต้องของซอฟต์แวร์ ซึ่งซอร์สโค้ดที่พัฒนาจากภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่งจำเป็นต้องได้รับการทดสอบเพื่อให้มั่นใจว่าโปรแกรมที่ได้ผิดพลาดน้อยที่สุดและทำงานได้ตรงตามข้อกำหนดและหนึ่งในกระบวนการทดสอบซอฟต์แวร์ที่ใช้ คือการทดสอบระดับหน่วย (unit testing) ซึ่งต้องอาศัยกราฟควบคุมกระแส (control flow graph) เพื่อให้เห็นผังการทำงานของซอร์สโค้ดและเห็นเส้นทางสำหรับการทดสอบ ซึ่งจะช่วยให้สามารถสร้างข้อมูลทดสอบเพื่อใช้สำหรับการทดสอบซอฟต์แวร์ในระดับนี้ได้สะดวกขึ้น ถ้ามีการทดสอบตามเส้นทางที่ได้จนครบจะทำให้เรามั่นใจได้ว่าในซอฟต์แวร์ที่พัฒนานั้นได้ทดสอบทุกเส้นทางแล้วอย่างน้อย 1 ครั้ง ซึ่งจะช่วยให้เกิดความมั่นใจในตัวซอฟต์แวร์เพิ่มขึ้นด้วย

ดังนั้นงานวิจัยนี้มุ่งเน้นเพื่อศึกษาและพัฒนาเครื่องมือสร้างกราฟควบคุมกระแสที่สามารถแสดงผังควบคุมการทำงานของซอร์สโค้ดภาษาซี โดยจะนำเอากราฟที่ได้นั้นมาสร้างเส้นทางสำหรับทดสอบซอฟต์แวร์ให้ครอบคลุมทุกเส้นทางที่เป็นไปได้และสร้างข้อมูลทดสอบเพื่อใช้สำหรับทดสอบตามเส้นทางทดสอบที่สร้างขึ้นให้ครบทุกเส้นทาง เพื่อสร้างความมั่นใจในตัวซอร์สโค้ดว่าได้ผ่านการทดสอบทุกเส้นทางแล้วอย่างน้อย 1 ครั้ง โดยซอร์สโค้ดที่จะนำเข้าสู่เครื่องมือนี้เป็นซอร์สโค้ดที่พัฒนาแค่ในระดับพื้นฐานเท่านั้น ยังไม่สนับสนุนซอร์สโค้ดที่พัฒนาแบบขั้นสูง นอกจากนี้ซอร์สโค้ดนั้นจะต้องผ่านการคอมไพล์หรือตรวจสอบหลักการเขียนและแก้ไขให้ถูกต้องก่อนเพราะถ้าไม่เป็นไปตามข้อกำหนดนี้กราฟควบคุมกระแส เส้นทางสำหรับทดสอบและข้อมูลทดสอบที่ได้อาจจะมีความผิดพลาดหรือไม่ครอบคลุมได้

5.1 สรุปผลการวิจัย

5.1.1 สรุปผลการทำงานของส่วนนำเข้าซอร์สโค้ดและติดต่อผู้ใช้งาน

สำหรับการทำงานของเครื่องมือในส่วนนี้ ผู้ทดสอบสามารถนำเข้าซอร์สโค้ดได้ทีละหนึ่งไฟล์ แต่ผู้ทดสอบสามารถที่จะนำเข้าใหม่กี่ครั้งก็ได้จะทำงานต่อเนื่องไปได้เรื่อย ๆ แต่ผลลัพธ์ของแต่ละซอร์สโค้ดที่ประมวลผลแล้วจะถูกแทนด้วยผลลัพธ์ของซอร์สโค้ดใหม่ทุกครั้ง

ซอร์สโค้ดที่นำเข้าสู่เครื่องมือจะต้องมีฟังก์ชันย่อย ๆ ภายใน (user-define function) ไม่เกิน 24 ฟังก์ชัน เนื่องจากเครื่องมือพัฒนาด้วยโปรแกรมประเภท GUI ผลลัพธ์ที่ได้จึงอยู่ในรูปแบบ GUI ด้วยซึ่งเป็นรูปแบบที่ง่ายต่อการใช้งาน

5.1.2 สรุปผลการทำงานส่วนกระจายนิพจน์

ส่วนต่าง ๆ ของเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบสำหรับภาษาซี ล้วนมีความสำคัญใกล้เคียงกัน ถ้าจะจัดลำดับแล้วส่วนกระจายนิพจน์ถือเป็นส่วนที่มีบทบาทสำคัญและมีผลต่อส่วนอื่น ๆ เป็นลำดับแรก เนื่องจากว่าเครื่องมือจะสามารถสร้างกราฟเส้นทางทดสอบและข้อมูลทดสอบได้ถูกต้องนั้น ข้อมูลที่เก็บไว้ในส่วนกระจายนิพจน์จะต้องมีความถูกต้อง ข้อมูลที่ได้จากส่วนกระจายนิพจน์ข้อมูลแรกคือโหนด รายละเอียดของโหนดและความสำพันธ์ของโหนดแต่ละโหนดที่จะสร้างเป็นกราฟควบคุมกระแส ข้อมูลตัวที่สองคือส่วนย่อยที่สุดของโปรแกรมหรือโทเคนแต่ละตัวที่เกี่ยวข้องกับการสร้างข้อมูลทดสอบ

โหนดที่ได้จากส่วนกระจายนิพจน์นี้เกิดจากการประมวลผลเพื่อแยกซอร์สโค้ดแต่ละคำสั่ง แต่ละบรรทัดออกจากกัน ซึ่งจะสนใจเฉพาะคำสั่งพื้นฐานเท่านั้นฟังก์ชันแต่ละฟังก์ชันที่แยกไม่ควรใหญ่จนเกินไปซึ่งกำหนดให้แต่ละฟังก์ชันมีโหนดได้ไม่เกิน 50 โหนด

ส่วนโทเคนที่เครื่องมือนี้ใช้แบ่งออกเป็น 4 ประเภทตาม คือตัวดำเนินการ ตัวแปรนิพจน์และค่าคงที่ ซึ่งแต่ละประเภทนั้นจะแยกเฉพาะที่เกี่ยวข้องกับโหนด เส้นทางทดสอบและข้อมูลทดสอบเท่านั้นและกำหนดว่าในแต่ละโหนดไม่ควรมีโทเคนเกิน 150 โทเคน

5.1.3 สรุปผลการทำงานส่วนสร้างกราฟควบคุมกระแส

ส่วนของกราฟที่เครื่องมือนี้สร้าง จะมีข้อจำกัดในเรื่องของพื้นที่แสดงผล เนื่องจากในหน้าจอต้องมีการแสดงส่วนสำคัญ ๆ ของผลลัพธ์ถึง 4 ส่วนพื้นที่ ที่จะต้องแสดงแต่ละส่วนจึงมีน้อย ฉะนั้น ถ้ากราฟใดมีโหนดมากและซอร์สโค้ดมีคำสั่งที่มีความซับซ้อนมากจะส่งผลให้กราฟมีขนาดที่ใหญ่ ซึ่งถ้ากราฟมีขนาดใหญ่จนล้นหน้าจอผู้ทดสอบอาจจะใช้งานไม่สะดวกเนื่องจากจะต้องปรับแถบเลื่อนดูข้อมูล (scrollbar) เพื่อเลื่อนดูกราฟด้วยตัวเอง

ในบางกราฟที่มีเงื่อนไขหลายนิพจน์ (ข้อความยาว) ที่การแสดงผลของข้อความประจำแต่ละโหนดของกราฟก็จะไม่สวยงาม เพราะบางโหนดอาจมีข้อความเงื่อนไขของโหนดทับซ้อนกันได้ (อาจจะเกิดขึ้นเพียงส่วนน้อย ในกรณีที่ซอร์สโค้ดนั้นมีเงื่อนไขยาว)

เนื่องจากเครื่องมือนี้ถูกพัฒนาขึ้นเพื่อ สร้างกราฟควบคุมกระแส เส้นทางทดสอบและข้อมูลทดสอบ ดังนั้นผู้วิจัยจึงไม่สนใจที่จะตรวจสอบหลักการเขียนของซอร์สโค้ดที่นำเข้า

เครื่องมือ นั่นหมายความว่ากราฟที่สร้างขึ้น อาจจะผิดพลาดหรือไม่เป็นอย่างที่ผู้ทดสอบต้องการได้ ถ้าซอร์สโค้ดนั้นยังไม่ผ่านการตรวจสอบด้วยตัวแปลภาษาคอมพิวเตอร์

5.1.4 สรุปผลการทำงานส่วนสร้างเส้นทางทดสอบ

สำหรับเส้นทางทดสอบเป็นส่วนที่จะแสดงให้ผู้ทดสอบเห็นภาพได้ชัดเจนขึ้นว่าควรจะทำการทดสอบมากน้อยแค่ไหนและการทำงานจะวิ่งไปตามเส้นทางใดบ้าง ในเครื่องมือนี้มีการออกแบบให้แต่ละฟังก์ชันมีเส้นทางทดสอบไม่เกิน 50 เส้นทางด้วยเหตุผลเกี่ยวกับพื้นที่ของหน้าจอเช่นเดียวกับเหตุผลของส่วนอื่น ๆ ถ้าเส้นทางใดมีเส้นทางยาวมากหรือจำนวนเส้นทางทดสอบมีมาก ก็จะส่งผลให้ล้นหน้าจอ ซึ่งผู้ทดสอบต้องใช้แถบเลื่อนดูข้อมูล เพื่อเลื่อนหน้าจอดูข้อมูลด้วยตนเอง

เส้นทางทดสอบที่เครื่องมือนี้สร้างขึ้นบางฟังก์ชันอาจจะมีเส้นทางบางเส้นทางที่ต้องมีการทดสอบผ่านโหนดบางโหนดบ่อย ๆ แต่ถ้ามองภาพรวมทั้งเส้นทางจะมั่นใจได้ว่า เส้นทางแต่ละเส้นทางไม่ซ้ำซ้อนและถ้าผู้ทดสอบใช้เส้นทางที่ได้นี้ช่วยในการทดสอบซอร์สโค้ดแล้วจะมั่นใจได้ว่าโปรแกรมที่พัฒนาขึ้นจะเคยผ่านการทดสอบมาแล้วทุกเส้นทาง อย่างน้อย 1 ครั้ง

5.1.5 สรุปผลการทำงานส่วนสร้างข้อมูลทดสอบ

ในการทดสอบเพื่อให้สามารถควบคุมเส้นทางการทำงานของโปรแกรมได้นั้น เราต้องควบคุมผลลัพธ์ของเงื่อนไขของแต่ละโหนดด้วย นั่นคือผู้ทดสอบต้องให้ค่ากับตัวแปรหรือนิพจน์ที่เกี่ยวข้องกับเงื่อนไขของแต่ละโหนดและส่วนสร้างข้อมูลทดสอบนี้เองจะเตรียมข้อมูลไว้ให้ผู้ทดสอบใช้งานเพื่อควบคุมเส้นทางการทำงานของโปรแกรมได้ ข้อมูลที่สร้างจากส่วนนี้จะได้จากการสุ่มโดยภายใต้ขอบเขตของค่าข้อมูลที่ตัวแปรหรือนิพจน์นั้นจะมีโอกาสเป็นไปได้ เช่น อักขระ ตัวเลขจำนวนเต็ม กรณีต้องกำหนดข้อมูลให้กับตัวแปร ข้อมูลที่กำหนดก็จะอยู่ภายใต้ขอบเขตของชนิดข้อมูลของตัวแปรนั้น ๆ ด้วย

5.2 การประยุกต์ผลการวิจัย

ผลลัพธ์ของงานวิจัยนี้คือ เครื่องมือที่สามารถสร้างกราฟควบคุมกระแสที่สามารถแสดงผังควบคุมการทำงานของซอร์สโค้ดภาษาซี และเตรียมข้อมูลสำหรับใช้ในการทดสอบโปรแกรม ซึ่งตามจุดประสงค์หลักเพื่อการสร้างเครื่องมือเพื่อช่วยอำนวยความสะดวกในกระบวนการทดสอบซอฟต์แวร์ในระยะของการทดสอบระดับหน่วย ดังนั้นสามารถนำงานวิจัยนี้ไปประยุกต์ใช้สำหรับการทดสอบซอร์สโค้ดที่พัฒนาด้วยภาษาซี ได้หลายส่วน ดังเช่น

- เมื่อผู้ทดสอบต้องการต้องการจะยืนยันหรือสร้างความมั่นใจว่าได้ทดสอบและบันทึกผลการทดสอบอย่างครอบคลุมทุกเงื่อนไขของทุกคำสั่งภายในฟังก์ชันแล้ว ผู้ทดสอบต้องรู้ว่าเส้นทางการทำงานของแต่ละฟังก์ชันเป็นอย่างไรและเครื่องมือจะช่วยให้ผู้ทดสอบวางแผนทดสอบและดำเนินการทดสอบได้สะดวกรวดเร็วและครอบคลุมยิ่งขึ้น

- เมื่อผู้ทดสอบต้องการได้ผังการทำงานของซอร์สโค้ดเพื่อแก้ปัญหาหรือปรับปรุงซอร์สโค้ดให้มีความถูกต้องกรณีที่โปรแกรมมีความผิดพลาด

- เมื่อผู้ทดสอบต้องการภาพรวมของซอร์สโค้ดแต่ละฟังก์ชันที่พัฒนาว่ามีความซับซ้อนมากน้อยแค่ไหน

5.3 ข้อเสนอแนะในการวิจัยต่อไป

ในกระบวนการพัฒนาซอฟต์แวร์ไม่ว่าจะพัฒนาด้วยโปรแกรมภาษาอะไรก็ตามทีล้วนต้องมีการทดสอบทั้งสิ้นและเพื่อให้การทดสอบไม่เป็นเรื่องที่น่าเบื่อ เปลี่ยนให้เป็นงานที่ง่ายสะดวกรวดเร็วขึ้น สำหรับผู้ทดสอบ จำเป็นต้องมีเครื่องมือที่มีความอัตโนมัติ ช่วยในขั้นตอนการทดสอบ ซึ่งในงานวิจัยนี้เครื่องมือจะช่วยอำนวยความสะดวกเฉพาะกับซอร์สโค้ดที่พัฒนาด้วยภาษาซี และยังมีข้อจำกัดหลาย ๆ อย่าง ฉะนั้นเพื่อให้เครื่องมือมีความสามารถเพิ่มขึ้นและมีประโยชน์อย่างกว้างขวางขึ้นจึงน่าจะมีการวิจัยเพื่อพัฒนาเครื่องมือนี้เพิ่มเติมแบ่งได้หลาย ๆ ด้านด้วยกัน อาทิเช่น

- เรื่องของพื้นที่การแสดงผล ควรมีการปรับปรุงส่วนแสดงผลให้สามารถแสดงผลได้กว้างมากขึ้น เพื่อผู้ทดสอบจะได้เห็นผลลัพธ์ได้อย่างต่อเนื่องและครบทุกส่วน

- การจัดเก็บหรือรูปแบบการแสดงผลเมื่อการประมวลผลเสร็จสิ้นลงควรที่จะสามารถนำผลลัพธ์นั้นออกไปจากเครื่องมือได้ อาทิเช่น การพิมพ์ออกทางเครื่องพิมพ์ หรือ การนำออกเป็นเอกสารหรือรูปภาพเพื่อจะได้นำไปใช้ต่อได้โดยไม่ต้องประมวลผลใหม่

- สำหรับตัวซอร์สโค้ดที่นำเข้าเดิมเป็นแค่ภาษาซีพื้นฐาน ในการพัฒตาต่อไปควรให้สามารถใช้ได้กับภาษาซี ที่เป็นขั้นสูงขึ้นไป

- และท้ายสุดคือเครื่องมือนี้ควรมีความสามารถรองรับซอร์สโค้ดได้หลากหลายมากขึ้น เนื่องจากปัจจุบันพัฒนาการของ ซอฟต์แวร์ที่เป็นโปรแกรมภาษามีความก้าวหน้ามากขึ้น ในขณะที่ความจำเป็นพื้นฐานในการทดสอบซอฟต์แวร์ล้วนมีเหมือน ๆ กัน ฉะนั้นไม่ว่าซอฟต์แวร์จะพัฒนาด้วยโปรแกรมภาษาใดก็ต้องการเครื่องมือช่วยในการทดสอบทั้งสิ้น

รายการอ้างอิง

ซิลด์ เฮอร์เบิร์ต, ศิวัฒน์ ศิวะบวร, พรชัย จักรธำรง, จิรศักดิ์ ชัยวิริยะกุล. (2536). **การประยุกต์ใช้ภาษาซี**. ซีเอ็ด : กรุงเทพ. เรียบเรียงจาก : C Power User's Guide. 608 pp.

ธนพล สิชฌนกฤษฎ์. (2544). **รายงานการวิจัยฉบับสมบูรณ์** : การพัฒนาระบบจัดการกรณีทดสอบซอฟต์แวร์. กรุงเทพมหานคร : สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย.

ภพพงศ์ สกุลพิพัฒน์ศิลป์. (2544). **รายงานการวิจัยฉบับสมบูรณ์** : การพัฒนาเครื่องมือซอฟต์แวร์สำหรับสร้างข้อมูลทดสอบ. กรุงเทพมหานคร : สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย.

สิทธิศักดิ์ คล่องดี. (2542). **คัมภีร์ Visual Basic 6.0 สำหรับโปรแกรมเมอร์**. กรุงเทพมหานคร : เยลโล่การพิมพ์. 347 หน้า.

Byron S. Gottfried. (1996). **Schaum's Outlines of Theory and problems of programming with C, Second Edition**. WCB McGraw-Hill International Editions. 531 pp.

H, H. Tan and T, B. D'Orazio. (1999). **C programming for engineering and computer science**. WCB McGraw-Hill International Editions. 748 pp.

John Watkins. (2001). **Testing IT An off-the-shelf Software Testing Process**. New York : Cambridge University Press. 315 pp.

Kuo-chung Tai and Yu Lei. (2002). A Test Generation Strategy for Pairwise Testing. Dept. of Comput. Sci., North Carolina State Univ., Raleigh, NC. In **proceedings of IEEE Transactions on Software Engineering**. p.109-111.

Ian Sommerville. (2001). **Software Engineering**. Harlow : Pearson Education. 742 pp.

Neelam Gupta, Aditya P. Mathur, Mary Lou Softa. (2000). Generating Test Data For Branch Coverage. Dept. of Comput. Sci., Arizona Univ., Tucson. In **proceedings of the 15th IEEE International Conference**, Location: Grenoble, France. p.219-227.

Nguyen Tran Sy, Yves Deville. (2001). Automatic Test Data Generation for Programs with Integer and Float Variables. Univ. Catholique de Louvain, Louvain-la-Neuve, Belgium. In **proceedings of the 16th Annual International Conference**. p.13-21.

Tippayasot, T. (2005). Development of control flow graph and test data for C language. In **proceedings of the 31th Congress on Science and technology of Thailand**, Suranaree University of Technology, Thailand.

Visser, W. Havelund, K. Brat, G. Seungjoon Park. (2000). Model Checking Programs. Comput. Sci., NASA Ames Res. Center, Moffett Field. In **proceedings of The 15th IEEE International Conference**. Location: Grenoble, France. p.3-11.

ภาคผนวก

บทความผลงานวิจัยที่นำเสนอในการประชุมวิชาการวิทยาศาสตร์และเทคโนโลยีแห่งประเทศไทย ครั้งที่ 31

การพัฒนาเครื่องมือสร้างกราฟควบคุมกระแสและข้อมูลทดสอบสำหรับภาษาซี

DEVELOPMENT OF CONTROL FLOW GRAPH AND TEST DATA TOOL FOR C LANGUAGE

ทิพยา ทิพยโสต, พิชโยทัย มหัทธนาภิวัฒน์

Tippaya Tippayasot, Pichayotai Mahatthanapiwat

School of Computer Engineering, Suranaree University of Technology, Nakhon Ratchasima 30000, Thailand,

E-mail address: tippayasot@hotmail.com, pmh@sut.ac.th

บทคัดย่อ: ในการพัฒนาซอฟต์แวร์จำเป็นต้องมีการทดสอบ เพื่อหาข้อผิดพลาดและตรวจสอบเพื่อให้ถูกต้องตามความต้องการ ซึ่งจะพบว่าในระบบที่ซับซ้อนส่งผลให้การทดสอบและการตรวจสอบยากตามไปด้วย ซอสโค้ดที่พัฒนาจากภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่งจำเป็นต้องได้รับการทดสอบ เพื่อให้มั่นใจว่าโปรแกรมที่ได้ผิดพลาดน้อยที่สุดและทำงานได้ตรงตามข้อกำหนด ในการทดสอบซอฟต์แวร์มีเฟส ของการทดสอบที่สำคัญเฟสหนึ่งคือการทดสอบระดับหน่วย เป็นเฟสที่ทดสอบการทำงานของแต่ละหน่วย แต่ละฟังก์ชัน หรือโปรซีเจอร์ ให้มีข้อผิดพลาดน้อยที่สุด ในการทดสอบระดับหน่วยจำเป็นต้องรู้โครงสร้างของซอสโค้ด ซึ่งกราฟควบคุมกระแสเป็นเครื่องมือที่แสดงให้เห็นผังทำงานของซอสโค้ดและเห็นเส้นทางสำหรับการทดสอบได้ชัดเจน ฉะนั้นเครื่องมือที่มีความสามารถสร้างกราฟควบคุมกระแสจากซอสโค้ด สร้างเส้นทางสำหรับการทดสอบซอฟต์แวร์ และสร้างข้อมูลทดสอบได้นั้น จึงเป็นเครื่องมือที่มีประโยชน์สำหรับกระบวนการทดสอบซอฟต์แวร์อย่างยิ่ง

Abstract : In software development process, validation and verification is necessary for software testing, i.e. how software performs according to the right specification and how the right process is performed to develop the software. The more complex system, the mere difficult for validation and verification. In unit testing phase, source code of the program in the module, for example, function or procedure, will be exercised if every path in the source code is traversed. Control flow graph generated from source code and test data will be used to visualize the flow of a program. If each path in the flow graph is exercised. Therefore, the tool to generate control flow graph and test data will be useful for the tester in unit testing phase.

Introduction: C language is widely used for many application due to its flexibility in coding and many available functions. C can handle data in bit or byte and used to operate hardware

effectively. Moreover, C has pointer data type that can access memory directly. Thus, it will be useful if we have a tool to test source code developed by c language.

In this paper, we give the characteristic of the tool that will be develop. This tool consists of 5 sections as follow : interface, parser, control flow graph generator, test path, generator and test data test data generator.

Methodology: The following is 5 sections of the tool.

1. **Interface section:** It is used as a user interface. C source code will be input by using this interface. However, source codes of standard C have to be checked for no syntax errors by c compiler before using as input for the interface. The c source codes will be used to generate control flow graph and test data.

2. **Parser section:** C source code will be passed to extract tokens. We can divide this section into 3 subsections as follows.

2.1 Token extractor: This part will extract tokens from c source code. Some tokens will be used as nodes in the control flow graph for example, token about condition such as if, else, while, etc.

2.2 Semantic analyzer: The analyzer will analyze tokens and classify their types such as, operator, identifier, command, number, as follows.

```
1   int play(int snd[],int dly[],int count){
        int     cnt;
2       for(cnt = 0;cnt < count;cnt++){
            printf("for command, cnt = %d\n",cnt);
3           if(bioskey(1)){
4               printf("if command, bioskey != 0\n");
                return 0;
            }
        }
5       printf("Exit for command, cnt = %d\n", cnt);
        return 1;
    }
```

Figure 1. C source code

From the source code, play is an integer function numbered by 1. This function has 3 arguments(2 arrays of integer and an integer). The repetition is controlled by for loop, so, it is numbered by 2. The condition of if is considered next by number 3.

2.3 Tracer: Tracer will find the relation between tokens for example, the group of code numbered 3 and 4 depends on the operation of for loop numbered 2.

3. Control flow graph section: The control flow graph will be generated as in figure 2 by using the information from parser section.

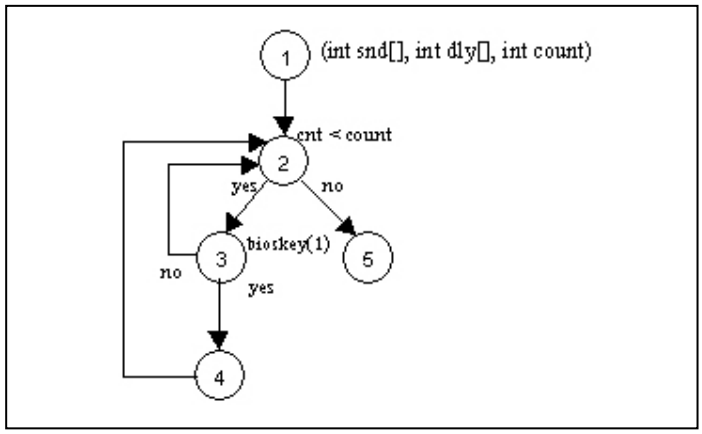


Figure 2. Control flow graph

4. Test Path section: Control flow graph generated from previous section will be use to generate the possible test paths as follows.

The first path : 1-2-3-4-2

The second path : 1-2-3-2

The third path : 1-2-5

5. Test Data section: This section examines how test paths will be exercised. So the value of variables of the condition will be generated to test the traversal for each path.

Conclusion : Control flow graph and test data tool is proposed for unit testing. This tool will be used with the right syntax standard c source code and generate control flow graph for a chosen module. Furthermore, test data will be automatically generated to exercise each path in the control flow graph. So, this tool will be useful for the tester in case of unit testing with c language. The tester can see the flow of the program and use the generated test data for unit testing, making the development process faster.

References : [1]. Ian Sommerville. (2001). Software Engineering. Pearson Education Limited.

[2]. John Watkins. (2001). Testing IT An off-the-shelf Software Testing Process. Cambridge University Press.

[3]. H. H. Tan and T. B. D'Orazio. (1999). C programming for engineering and computer science. WCB McGraw-Hill International Editions.

Keywords : Unit testing, Control flow graph, Test Path, Test Data.

ประวัติผู้เขียน

นางสาวทิพยา ทิพยโสต เกิดเมื่อวันที่ 2 พฤษภาคม พ.ศ. 2519 สำเร็จการศึกษาระดับปริญญาตรี เกียรตินิยมอันดับ 2 โปรแกรมวิชาวิทยาการคอมพิวเตอร์ สถาบันราชภัฏนครราชสีมา ในปี 2542 โดยระหว่างศึกษาได้เข้าทำงานเป็นพนักงานรายชั่วโมงที่บริษัทสยามมัลติซอฟต์ ตำแหน่งโปรแกรมเมอร์ หลังจบการศึกษาได้เข้าทำงานที่บริษัทอัลฟาออฟฟิชออโตเมชั่น จำกัด ตำแหน่งนักวิเคราะห์ระบบ ตั้งแต่ เดือนเมษายน - เดือนพฤศจิกายน 2542 และทำงานในตำแหน่งนักวิเคราะห์ระบบที่บริษัทเทอร์โบซอฟต์ ตั้งแต่ เดือนธันวาคม 2542 - เดือนมิถุนายน 2544 และตั้งแต่เดือนกรกฎาคม 2544 จนถึงปัจจุบัน ได้เข้าทำงานตำแหน่งอาจารย์พิเศษตามสัญญา ที่มหาวิทยาลัยราชภัฏนครราชสีมา สังกัดโปรแกรมวิชาวิทยาการคอมพิวเตอร์ และในปี 2547 ได้เข้าศึกษาต่อในระดับปริญญาโท ในสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี โดยได้รับทุนสนับสนุนการศึกษาจากมหาวิทยาลัยราชภัฏนครราชสีมา

ในระหว่างการศึกษาได้รับความอนุเคราะห์อย่างดียิ่งจากคณาจารย์ในสาขาวิชาทุกท่าน และยังได้รับความไว้วางใจให้เป็นผู้ช่วยวิจัยและผู้ช่วยสอนปฏิบัติการในหลายรายวิชา ดังนี้ 1) Event-Driven Programming, 2) Object-Oriented Technology, 3) System Analysis and Design, 4) Software Engineering