การวิเคราะห์เวคเตอร์บนโครงข้อหมุนสองมิติโดยไมโครคอมพิวเตอร์

(Vector Analysis On 2-D Dimension Truss by Microcomputer)

โดย

นายสินเจริญ ตั้งขันติกุล

นายเอกราช ศรีตาล

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมโยธา

สาขาวิชาวิศวกรรมการก่อสร้าง

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

การวิเคราะห์เวคเตอร์บนโครงข้อหมุนสองมิติโดยไมโครคอมพิวเตอร์

(Vector Analysis On 2-D Dimension Truss by Microcomputer)

โดย

นายสินเจริญ ตั้งขันติกุล

นายเอกราช ศรีตาล

อาจารย์ที่ปรึกษา

อาจารย์ ศักดิ์ชัย สกานุพงษ์

เสนอ

ภาควิชาวิศวกรรมโยธา

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เพื่อความสมบูรณ์แห่งปริญญาวิศวกรรมศาสตร์บัณฑิต (วิศวกรรมการก่อสร้าง)

ปีการศึกษา 2537

Vector Analysis On 2-D Dimension Truss by Microcomputer

Mr. SINCHAROEN TANGKHANTIKUL

Mr. AKARAT SRITAL

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE BACHELOR DEGREE

OF CONSTRUCTION ENGINEERING

KING MONGKUT 'S INSTITUTE OF TECHNOLOGY LADKRABANG

1994

ภาควิชาวิศกรรมโยธา คณะวิศวกรรมศาตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองโครงงานพิเศษ

หัวข้อโครงงานพิเศษ การวิเคราะห์เวคเตอร์บนโครงข้อหมุนสองมิติโดยไมโครคอมพิวเตอร์

(Vector Analysis On 2-D Dimension Truss by Microcomputer)

นักศึกษา นาย สินเจริญ ตั้งขันติกุล รหัส 34108423

นาย เอกราช ศรีตาล รหัส 34109522

หลักสูตร วิศวกรรมศาสตร์บัณฑิต สาขาวิชา วิศวกรรมการก่อสร้าง

ภาควิชา วิศวกรรมโยธา

อาจารย์ที่ปรึกษา อาจารย์ศักดิ์ชัย สกานุพงษ์

คณะกรรมการสอบหัวข้อโครงงานพิเศษ	ลายมือชื่อ
1.....	.....
2.....	.....
3.....	.....
4.....	.....
5.....	.....

ภาควิชาวิศวกรรมโยธารับรองแล้ว

.....

(อาจารย์ อำนวย พานิชกุลพงศ์)

หัวหน้าภาควิชาวิศวกรรมโยธา

การวิเคราะห์เวคเตอร์บนโครงข้อหมุนสองมิติโดยไมโครคอมพิวเตอร์

(Vector Analysis On 2-D Dimension Truss by Microcomputer)

โดย

นายสินเจริญ ตั้งขันติกุล

นายเอกราช ศรีตาล

อาจารย์ที่ปรึกษา อ.ศักดิ์ชัย สกานุพงษ์

บทคัดย่อ

โครงการพิเศษนี้เป็นการศึกษาการนำเอาการวิเคราะห์เวคเตอร์มาช่วยในการวิเคราะห์โครงสร้างประเภทโครงข้อหมุน 2 มิติ โดยอาศัยไมโครคอมพิวเตอร์ เข้ามาช่วยในการวิเคราะห์การคำนวณ ซึ่งการศึกษาดังกล่าวเป็นอีกวิธีหนึ่ง ที่จะช่วยในการหาแรงและการโก่งตัวภายในชิ้นส่วนอันเกิดจาก แรงกระทำ ณ จุดต่อของโครงสร้าง ประเภทโครงข้อหมุน และวิธีการดังกล่าว ยังสามารถแก้ไขหาคำตอบกับโครงสร้าง Determinate ได้อย่างมีประสิทธิภาพ โดยอาศัยหลักการ Method of Section และ Method of Joint กับ วิธี Vector Analysis หาแรงที่เกิดภายในชิ้นส่วนจนได้แรงภายในทุกชิ้น จากนั้นก็ทำการวิเคราะห์หาค่าการโก่งตัว ณ จุดต่อต่างๆ โดยอาศัยหลักการของ Unit Load Method จนกระทั่งได้ค่าการโก่งตัวทุกจุดจนครบ โปรแกรมสามารถนำไปวิเคราะห์โครงสร้างชนิด Determinate ได้เป็นอย่างดี

Vector Analysis On 2-D Dimension Truss by Microcomputer

By

Mr. SINCHAROEN TANGKHANTIKUL

Mr. AKARAT SRITAL

ABSTRACT

This Project is to study how Vector Analysis On 2-D Dimension Truss by Microcomputer , this Method is another way to find force and defection that produced by forced at any node of structure and that way can determine a solution of Determinate structure efficiently by use Method of Section and Method of Joint combine with vector analysis to find whole force , and then find defection at any node by Unit Load Method until get all . This Program can use to Analysis Determinate structure efficiently.

กิตติกรรมประกาศ

การจัดทำโครงงานพิเศษ เรื่องการวิเคราะห์เวคเตอร์บนโครงข้อหมุนสองมิติ โดยไมโครคอมพิวเตอร์นี้สำเร็จได้ด้วยดี ผู้จัดทำขอขอบคุณบุคคลผู้ให้ความอนุเคราะห์ ดังรายนามต่อไปนี้

1. อาจารย์ ศักดิ์ชัย สถานุพงษ์ อาจารย์ที่ปรึกษาโครงการวิจัย
2. อาจารย์ภาควิชาวิศวกรรมโยธาทุกท่าน

นอกจากนี้ผู้จัดทำขอขอบคุณบุพการี ที่ให้ความช่วยเหลือด้านกำลังใจและกำลังทรัพย์ รวมทั้งเพื่อนๆ ทุกคนที่ช่วยให้ความสะดวกและคอยให้ความช่วยเหลือ ทำให้โครงงานพิเศษนี้สำเร็จลุล่วงไปได้ด้วยดี

สินเจริญ ตั้งขันติกุล
เอกราช ศรีตาล
นักศึกษาผู้ประกาศ

สารบัญ

บทที่ 1

1.1 ความเป็นมาของปัญหาและความสำคัญของปัญหา

ปัจจุบันในการก่อสร้างทั่วๆ ไปนั้น ก่อนที่จะทำการก่อสร้างสิ่งใดๆขึ้นมานั้น จะต้องมีการออกแบบก่อนเสมอ เพื่อที่จะให้สิ่งก่อสร้าง ตรงตามประสงค์ของเจ้าของ และมีความแข็งแรงและทนทานสามารถในการรับกำลัง จากแรงต่างๆ ที่กระทำต่อโครงสร้างได้ การออกแบบในอดีตและปัจจุบันส่วนใหญ่ก็ยังอาศัยมือ ในการคิดคำนวณ หาแรงที่เกิดขึ้นในชิ้นส่วนต่างๆ ของโครงสร้างซึ่งจำเป็นที่จะต้องมีความอดทน ความเที่ยงตรง ของผู้ออกแบบสูง รวมทั้งเวลาที่ใช้ไปในการคำนวณ ก็จะสูงตามไปด้วย ดังนั้น ในปัจจุบันจึงเริ่มมีการนำเอาเทคโนโลยี ทางด้านโปรแกรมคอมพิวเตอร์เข้ามาช่วยในการคำนวณมากขึ้นเป็นลำดับ ไม่จำกัดอยู่เฉพาะการออกแบบแต่จะรวมไปในงานทุกๆ ด้านเพราะคอมพิวเตอร์สามารถทำงานได้เร็ว ความเที่ยงตรงสูง ดังนั้น โครงงานนี้จึงจัดทำขึ้นเพื่อศึกษาการนำเอาโปรแกรมคอมพิวเตอร์ เข้ามาประยุกต์ใช้กับทฤษฎีในการวิเคราะห์โครงสร้าง ประเภทโครงข้อหมุนเพื่อไปใช้งาน และไปพัฒนาต่อไปในอนาคต

1.2 วัตถุประสงค์ของโครงงานพิเศษ

1. เพื่อศึกษาการนำเอาโปรแกรมคอมพิวเตอร์มาประยุกต์ใช้กับงานวิเคราะห์โครงสร้าง
2. เพื่อศึกษาการนำเอาวิธีการวิเคราะห์เวคเตอร์ มาประยุกต์หาแรง และค่าการโก่งตัวที่เกิดขึ้นในโครงข้อหมุน ภายใต้น้ำหนักที่กระทำ
3. เพื่อจัดทำโปรแกรมวิเคราะห์โครงข้อหมุน Determinate โดยอาศัยทฤษฎีที่ศึกษาในข้อ 2

1.3 ขอบข่ายของการวิจัย

ศึกษาถึงทฤษฎีและการพัฒนาเป็นโปรแกรมวิเคราะห์หาแรงและค่าการโกงตัวของโครงข้อหมุน 2 มิติ ชนิด Determinate โดยวิธีเวคเตอร์

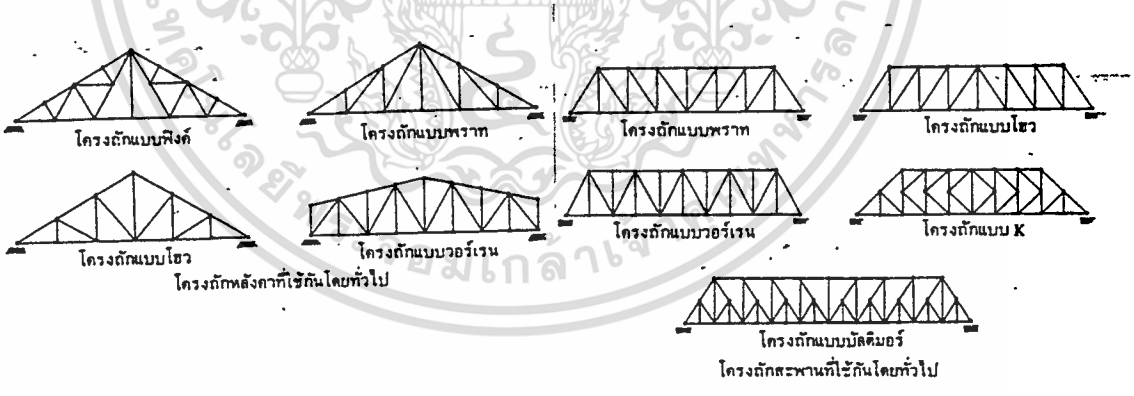
บทที่ 2

โครงข้อหมุน

2.1 ลักษณะโครงข้อหมุน

โครงข้อหมุน เป็นโครงสร้างชนิดหนึ่งที่พบเห็นได้โดยทั่วไปมักจะนำไปใช้เป็นโครงสร้างที่พาดช่วงยาวๆ โดยที่ไม่ต้องการให้มีเสาระหว่างกลางช่วง ซึ่งเป็นวัตถุประสงค์อย่างหนึ่งของการออกแบบอาคาร โอกาสที่นำไปใช้ได้แก่ โครงหลังคา (Roof Truss) โครงสะพาน (Bridge Truss) คานโครง (Trussed Girder)

โครงข้อหมุน (Truss) จะประกอบด้วยชิ้นส่วน ซึ่งยึดติดกันที่ปลาย เพื่อประกอบกันเป็นโครงสร้างที่แข็งแกร่ง ใช้ในงานต่างๆ เช่น สะพาน โครงหลังคา ปั้นจั่น และโครงสร้างในลักษณะเดียวกันเป็นตัวอย่างของโครงข้อหมุนที่พบอยู่เสมอ ชิ้นส่วนที่ใช้มักเป็นเหล็กรูปตัว I และ U เหล็กฉาก เหล็กกลมและแบบพิเศษอย่างที่ยึดกันที่ปลายด้วยการเชื่อมด้วยไฟฟ้า ใช้นอตขันหรือ ใช้หมุดย้ำ โครงสร้างสามารถประกอบได้ทั้งแบบสองหรือสามมิติ แต่ที่สนใจเป็นเพียงสองมิติเท่านั้น โครงข้อหมุนสองมิติ คือ โครงถักที่ชิ้นส่วนต่างๆ วางอยู่ในระนาบเดียวกัน เรียกว่า โครงถักระนาบ(Plane Truss) โครงถักระนาบพวกที่ใช้กับสะพานมักจะออกแบบเป็นคู่ๆ ซึ่งจะวางโครงถักเป็นแถวๆ ไว้แต่ละข้างของสะพานและยึดด้วยคานขวาง เพื่อที่จะรับน้ำหนักของพื้นถนนซึ่งจะถ่ายน้ำหนักไปยังชิ้นส่วนของโครงถัก ตัวอย่างของโครงถักระนาบที่ใช้กันทั่วๆ ไป ดังแสดงไว้ในรูปที่ 2.1 ดังนี้



รูปที่ 2.1

สำหรับโครงหลังคา เป็นโครงสร้างที่รับน้ำหนักของเครื่องมุง เช่น น้ำหนักของกระเบื้องหรือสังกะสี พร้อมทั้งระแนง หรือ แปรับกระเบื้อง น้ำหนักของแผ่นกันความร้อน น้ำหนักของฝ้าเพดาน (ถ้ามี) แรงลมและน้ำหนักของโครงหลังคา เป็นต้น

สำหรับโครงสะพานเป็นโครงสร้างที่รับน้ำหนักของ โครงสะพานเอง น้ำหนักของพื้นและคานรับพื้นสะพาน แ ละน้ำหนักของยานพาหนะ เป็นต้น

คานที่ใช้กับระบบพื้น เป็นโครงสร้างที่รับน้ำหนักของคานโครงเอง น้ำหนักของพื้นและตง รวมทั้งน้ำหนักบรรทุกจร เป็นต้น

การคิดน้ำหนักต่างๆ เช่น น้ำหนักบรรทุกจร ขึ้นอยู่กับชนิดของอาคาร เช่น อาคารที่พักอาศัย 200 กก/ม^2 สำหนักงาน 250 กก/ม^2 น้ำหนักของวัสดุต่างๆ เช่น พื้น กระเบื้องหลังคา หาได้จากหนังสือคู่มือทั่วๆ ไป แรงลมต้องคิดเมื่อโครงหลังคาเป็นแบบโครงชัน (Pitched Truss) ซึ่งมีความชันคือ อัตราส่วนระหว่างส่วนยก ต่อ ช่วงยาว ตั้งแต่ 1 ใน 6 หรือมากกว่า ส่วนโครงหลังคาแบบหลังแบน (Flat Truss) ซึ่งมีความชัน 1 ใน 8 ถึง 10 ไม่จำเป็นที่จะต้องคิดแรงลมในหลังคาประเภทนี้ เนื่องจากแรงลมในหลังคาประเภทนี้เป็นลมดูด(Suction) ซึ่งจะลดค่าของแรงในส่วนต่างๆ ของโครงหลังคาลง สำหรับน้ำหนักของโครงหลังคานั้นประมาณเอาจากสูตรต่อไปนี้

สำหรับหลังคาโครงชัน

$$W = 1.024\ L \text{ กก/ม}^2$$

สำหรับหลังคาโครงแบน

$$W = 0.688\ L \text{ กก/ม}^2$$

เมื่อ W = น้ำหนักของโครงหลังคาเป็น กก/ม^2 บนพื้นที่ในแนวราบ

L = ช่วงยาวของโครงสร้างหลังคาเป็นเมตร

การประกอบโครงข้อหมุน จำเป็นจะต้องยกขื่อที่กลางช่วงของโครงสร้างหลังคา เมื่อยกโครงสร้างของหลังคาขึ้นติดตั้งเข้าที่แล้ว จะได้มีส่วนตกหรือโก่งในแนวดิ่ง พอดีกับส่วนยก ทำให้ขื่ออยู่ในแนวราบพอดี สามารถคำนวณโดยสูตรได้ดังนี้

$$Y = \frac{L^2\ (875L+23334) \times 10^{-6}}{H}$$

ในเมื่อ

Y = ส่วนยกที่กลางช่วงเป็นเซนติเมตร

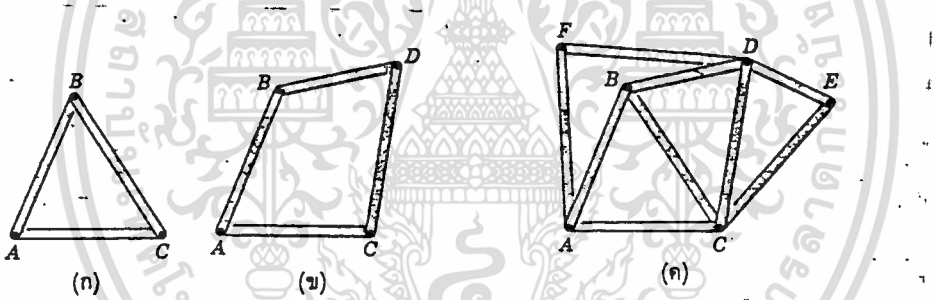
L = ช่วงยาวของโครงสร้างเป็นเซนติเมตร

H = ความลึกหรือส่วนสูงของโครงหลังคาเป็นเมตร

2.2 ความหมายของโครงข้อหมุน

โครงข้อหมุนหรือโครงถัก (Truss) หมายถึงโครงสร้างที่ประกอบขึ้นจากท่อนหรือแท่ง (Bars) ซึ่งรับแรงโดยตรง ต่อยึดกันเป็นรูปสามเหลี่ยมและประกอบรวมกันเป็นโครงสร้างที่มีรูปร่างและรับแรงหรือน้ำหนักได้ตามต้องการ

องค์ประกอบหลักของโครงถักระนาบ คือ สามเหลี่ยม ชิ้นส่วนสามชิ้นต่อกันที่ปลายด้วยสลักประกอบกันเป็นโครงที่แข็งเกร็ง(rigid frame) ดังแสดงในรูปที่ 2.2 (ก) หรือในลักษณะอื่นชิ้นส่วนสี่ชิ้นหรือมากกว่า ต่อกันด้วยสลักประกอบกันเป็นรูปหลายเหลี่ยมเป็นโครงที่ไม่แข็งเกร็ง (nonrigid frame) ดังรูปที่ 2.2(ข) โครงที่ไม่แข็งเกร็งสามารถทำให้แข็งแกร็งได้โดยการเพิ่มชิ้นส่วนทางด้านทะแยงมุม เชื่อมระหว่าง A และ D กลายเป็นรูปสามเหลี่ยม 2 รูป โครงสร้างสามารถขยายออกไปได้โดยการเพิ่มจำนวนขึ้น เช่น DE และ CE หรือ AF และ DF ดังแสดงในรูปที่ 2.2(ค) โดยวิธีนี้ยังคงรักษาความแข็งเกร็งไว้ได้ คำว่า แข็งเกร็งหมายถึง การที่โครงสร้างไม่ยุบ(noncollapsible) และการเปลี่ยนรูปไปของชิ้นส่วนเนื่องจากความเครียดภายในที่เกิดขึ้นสามารถละทิ้งได้

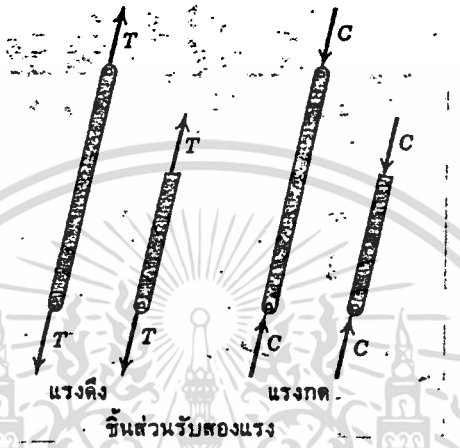


รูปที่ 2.2

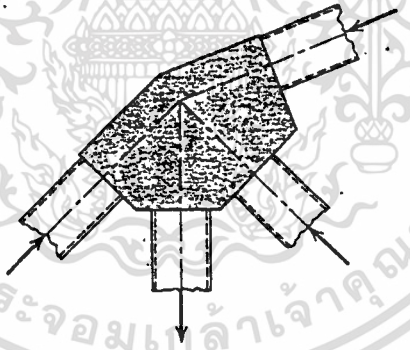
โครงสร้างที่ประกอบมาจากรูปสามเหลี่ยมในลักษณะที่กล่าวมาเรียกว่า โครงถักธรรมดา (Simple Truss) ถ้ามีชิ้นส่วนมากกว่าความจำเป็นต้องใช้เพื่อที่ไม่ให้โครงสร้างยุบตัว โครงถักนี้เรียกว่า statically ineterminate ซึ่งไม่สามารถในการวิเคราะห์ได้ถ้ามีเฉพาะสมการของสมดุลของเพียงอย่างเดียว ชิ้นส่วนที่เกินความจำเป็นเรียกว่า Redundant

ในการออกแบบโครงถักจะต้องหาแรงในชิ้นส่วนต่างๆ และเลือกขนาดและรูปร่างที่เหมาะสมเพื่อรับแรงดังกล่าว ในการวิเคราะห์หาแรงของโครงถักธรรมดา(Simple truss) จะต้องตั้งสมมุติฐานขึ้นหลายอย่าง ประการแรกจะต้องสมมุติว่าชิ้นส่วนทุกชิ้นเป็นชิ้นส่วนรับสองแรง ซึ่งหมายถึงชิ้นส่วนที่มีแรงกระทำเพียงสองแรงเท่านั้น เมื่ออยู่ในสภาวะสมดุลตามที่ได้นิยามไว้ในรูปที่ 2.3(ก) แรงสองแรงจะต้องกระทำที่ปลายของชิ้นส่วนโดยมีขนาดที่เท่ากัน ทิศทางตรงกันข้ามกัน และอยู่ในแนวเดียวกัน ชิ้นส่วนนั้นอาจจะถูกกดหรือถูกดึงดังรูปที่ 2.3 ขอให้สังเกตว่า

แรงดึง T หรือแรงกด C ที่กระทำต่อหน้าตัดใดๆ มีค่าเหมือนเดิมตลอด ต่อไปต้องสมมุติว่าไม่คิดน้ำหนักของชิ้นส่วน เพราะว่าน้อยมากเมื่อเปรียบเทียบกับแรงที่ชิ้นส่วนรับอยู่ แต่ถ้าต้องการคิดน้ำหนักของชิ้นส่วน Wเมื่อกระจายอย่างสม่ำเสมอสามารถแทนได้ด้วยแรง W/2 สองแรงกระทำที่ปลายของชิ้นส่วน ซึ่งเทียบได้จากแรงภายนอกที่กระทำต่อสลัก การคิดน้ำหนักของชิ้นส่วนเข้าไปด้วยแบบนี้จะได้คำตอบที่ถูกต้องของแรงดึงและแรงกด แรงกดเฉลี่ยที่กระทำต่อชิ้นส่วนแต่ไม่ได้นับรวมถึงผลการดัดตัว(bending) ของชิ้นส่วน

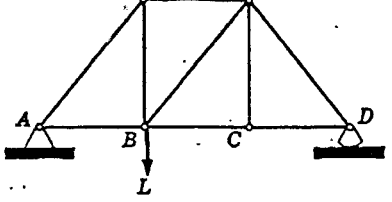


รูปที่ 2.3

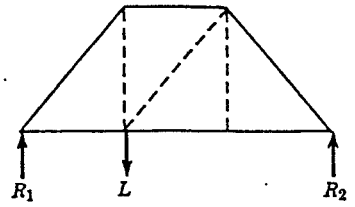


รูปที่ 2.4

โครงถักอย่างใหญ่มักจะมีอุปกรณ์ที่เผื่อไว้สำหรับการยืดหรือหดตัวที่เนื่องมาจากอุณหภูมิ หรือสำหรับการเปลี่ยนรูปเนื่องจากแรงกระทำ อุปกรณ์เหล่านี้มักอยู่ที่ฐานใดฐานหนึ่ง ตัวอย่างเช่น ลูกกลิ้ง (Roller) แป้นโยก(Rocker) หรืออาจจะเป็นพวกข้อต่อที่เคลื่อนที่ได้ (Slip joint) โครงถักที่ไม่มีอุปกรณ์เหล่านี้จะเป็นปัญหาแบบ Statically Indeterminate



(ก)



(ข)

รูปที่ 2.5

2.3 ลักษณะโครงข้อหมุนตามทฤษฎี

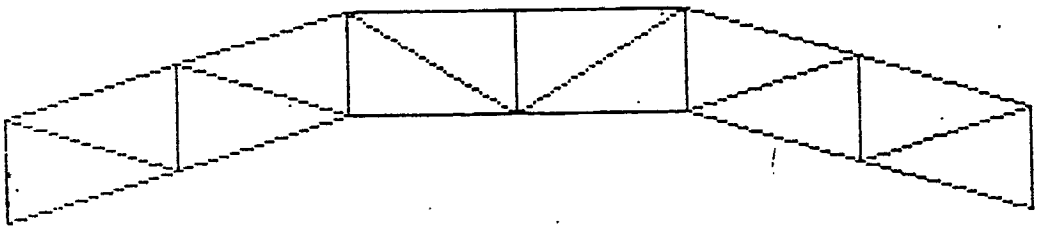
ก. ชิ้นส่วนแต่ละท่อนหรือแท่งที่นำมาประกอบกัน ต้องเป็นท่อนตรงตลอด (Straight)

ข. จุดต่อหรือข้อต่อ (Joint) ของชิ้นส่วนต่างๆ ถือว่าเป็นแบบหมุดยึดหมุน (Pin) ซึ่งหมุนได้และไม่มีความฝืด นั่นคือผลรวมของโมเมนต์ดัดที่ข้อต่อนี้ต้องเป็นศูนย์

ค. แรงหรือน้ำหนักที่บรรทุกกระทำ ต้องกระทำที่จุดต่อเท่านั่น

อย่างไรก็ตาม สิ่งต่างๆ ดังกล่าวข้างต้นนั้นเป็นลักษณะเฉพาะทางทฤษฎีเท่านั้นเพื่อที่จะให้ง่ายในการคำนวณ ซึ่งในทางปฏิบัติจริงๆ นั้น ชิ้นส่วนแต่ละท่อนหรือแท่งที่ในมาประกอบกันเป็นโครงสร้างข้อหมุนอาจไม่เป็นท่อนตรงตลอด อาจจะคดหรือแอ่นเล็กน้อยตามธรรมชาติ (เช่น ไม้หรือเหล็ก) ส่วนรอยต่ออาจใช้สลักเกลียวยึดสองตัวหรือมากกว่าก็ได้ และถ้าเป็นโครงเหล็กก็อาจนำเชื่อมตรงรอยต่อนี้ ซึ่งจะทำให้รอยต่อเป็นแบบยึดหมุนดังกล่าว อีกทั้งแรงหรือน้ำหนักบรรทุกก็อาจกระทำในช่วงระหว่างรอยต่อทั้งสอง เช่น การวางแป เป็นต้น

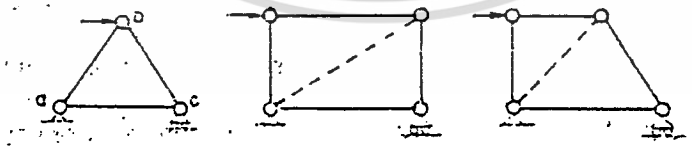
ชิ้นส่วนต่างๆ ที่นำมาประกอบรวมกันเป็นโครงข้อหมุนมีชื่อเรียกต่างๆ กัน คือ ส่วนที่อยู่ด้านบนเรียกว่า จันทัน (Top or Upper Chord) ส่วนที่อยู่ด้านล่างของโครงเรียกว่า ขื่อ (Bottom or Lower Chord) ส่วนที่ยึดอยู่ระหว่างคอร์ดบนกับคอร์ดล่าง เรียกว่า เวป (Web) ซึ่งอาจจะอยู่ในแนวทะแยง (Diagonal) หรืออยู่ในแนวตั้ง (Vertical) ก็ได้



รูปที่ 2.6

2.4 จำนวนของชิ้นส่วนที่ต้องการ

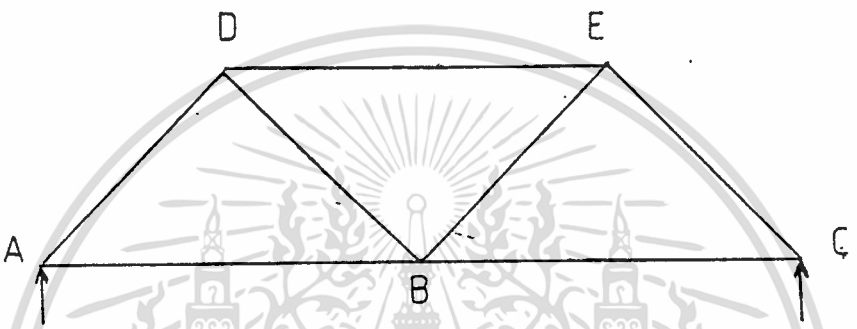
การที่โครงข้อหมุนต้องประกอบจากรูปสามเหลี่ยมหลายรูปรวมกันนั้น เพราะว่ารูปสามเหลี่ยมมีเป็นรูปทางเรขาคณิตที่มีเสถียรภาพที่มั่นคง (Stable) ไม่อาจเปลี่ยนไปเป็นรูปอื่นได้ภายใต้แรงกระทำที่กำหนด ซึ่งไม่ทำให้ความยาวทางด้านใดด้านหนึ่งของรูปเปลี่ยนแปลงไป เช่น โครงรูปสามเหลี่ยมที่ต่อกันแบบหมุดยึดหมุน ในรูปที่ 2.7 (ก) เมื่อรับแรงกระทำก็จะไม่เปลี่ยนรูป แต่สำหรับโครงสร้างแบบอย่างอื่น ดังรูปที่ 2.7 (ข) และ (ค) เมื่อรับแรงกระทำที่มีขนาดเดียวกัน ก็จะเปลี่ยนและเสียรูปทันที แต่ถ้าโครงสร้างทั้งสองนี้ถูกค้ำยันตามแนวเส้นประที่แสดง ซึ่งเป็นการทำให้โครงสร้างดังกล่าวประกอบกันด้วยรูปสามเหลี่ยมสองรูป โครงทั้งสองที่ถูกค้ำยันนี้จะมีเสถียรภาพที่มั่นคงมากขึ้น และไม่เปลี่ยนรูปเมื่อมีน้ำหนักหรือแรงมากระทำ



(ก) Stable frame (ข) Unstable frame (ค) Unstable frame

รูปที่ 2.7

สังเกตจากรูปที่ 2.7 (ก) จะเห็นได้ว่า ถ้าให้ชิ้นส่วน ac ยึดอยู่กับที่ (Fixed) ตำแหน่งของจุดต่อ b ก็จะอยู่กับที่ด้วย ซึ่งความสัมพันธ์กับชิ้นส่วน ac โดยอาศัยชิ้นส่วน ab และชิ้นส่วน bc ที่ประกอบรวมกันเป็นรูปสามเหลี่ยม abc ซึ่งมีเสถียรภาพและมั่นคง ในทำนองเดียวกันเมื่อพิจารณาโครงข้อหมุนในรูปที่ 2.8 จะเห็นได้ว่า ตำแหน่งของจุดต่อ A,B และ D จะยึดติดอยู่กับที่ เมื่อพิจารณาจุดต่อ E ต่อไปจะเห็นว่า เมื่อจุด D และ B ถูกยึดอยู่กับที่แล้ว (คือชิ้นส่วน DB ยึดติดอยู่กับที่) ตำแหน่งของจุดต่อ E ก็จะได้จากการประกอบของชิ้นส่วน DE และ BE ซึ่งถ้าพิจารณาต่อไป ก็พอที่จะสรุปได้ว่า ตำแหน่งของจุดต่อตัวใดๆ ต้องการชิ้นส่วนอีก 2 ชิ้นมาประกอบกัน



รูปที่ 2.8

ผลจากการพิจารณาดังกล่าว จะได้ความสัมพันธ์ของการประกอบรวมกัน เป็นโครงข้อหมุนที่สมบูรณ์มีเสถียรภาพมั่นคงกว่า โครงข้อหมุนต้องเริ่มต้นจากชิ้นส่วนหนึ่งที่ยึดติดอยู่กับที่ และที่ปลายทั้งสองของชิ้นส่วนนั้นเป็นตำแหน่งคงที่ ตำแหน่งของจุดต่อตัวหนึ่งต้องการชิ้นส่วนเพิ่มอีกเพียง 2 ชิ้นเท่านั้น ความสัมพันธ์นี้สามารถที่จะอธิบายได้จากสมการทางพีชคณิต ดังนี้

ถ้าให้ n = จำนวนของชิ้นส่วนที่ต้องการ

j = จำนวนของจุดต่อ

ดังนั้น $n = 1 + 2(j - 2)$

หรือ $n = 2j - 3$

นั้นคือ โครงข้อหมุนที่เป็นแบบ Determinate มีเสถียรภาพมั่นคง ต้องมีจำนวนของชิ้นส่วนเป็นสองเท่าของจำนวนจุดต่อลบด้วยสาม

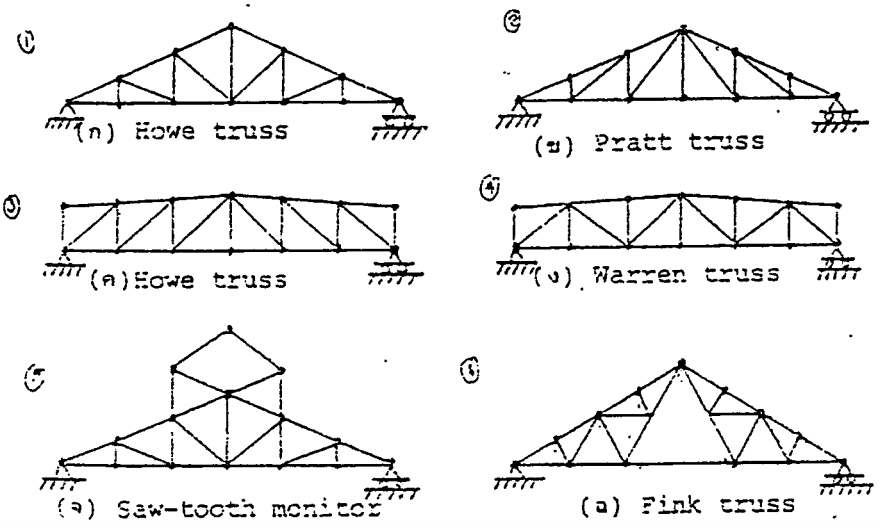
ถ้าโครงข้อหมุนใดมีจำนวนชิ้นส่วนน้อยกว่า (2j - 3) แสดงว่าโครงสร้างนั้นไม่มีความเสถียรภาพในการรับน้ำหนัก (Unstable) แต่ถ้าโครงข้อหมุนใดมีจำนวนชิ้นส่วนมากกว่า (2j - 3) แสดงว่าโครงสร้างนั้นเสถียรภาพเกินความจำเป็น จัดเป็นโครงข้อหมุนแบบ Indeterminater

2.5 ประเภทของโครงข้อหมุน

ประเภทของโครงข้อหมุนหรือโครงถักใช้กันทั่วไปในงานก่อสร้างสะพานและอาคาร สำหรับงานอาคารส่วนใหญ่มักใช้ โครงข้อหมุนเป็นโครงรับหลังคา ที่มีช่วงยาวและไม่ต้องการเสาในช่วงนั้น ในบางครั้งอาจใช้เป็นโครงรับพื้น ส่วนโครงข้อหมุนที่ใช้เป็นโครงสะพานรับน้ำหนักรถยนต์ รถบรรทุก หรือ รถไฟ ส่วนใหญ่มักมีความยาวของช่วงตั้งแต่ 20 เมตร จนถึง 40 เมตร หรือมากกว่านั้น

2.5.1 โครงหลังคา แบบที่นิยมใช้สำหรับโครงหลังคาที่มีความยาวปานกลางและมีความลาดเอียงของหลังคาตาที่ต้องการคือ แบบแพรตต์ (Pratt) แบบโฮว์ (Howe) และแบบฟิงค์ (Fink) โครงข้อหมุนแบบโฮว์ใช้กันทั้งที่ทำด้วยไม้และเหล็ก แต่ประหยัดน้อยกว่า แบบแพรตต์และแบบฟิงค์ แบบฟิงค์ใช้ได้กับระยะช่วงยาวมากกว่าและเปลืองวัสดุน้อยกว่าแบบแพรตต์ โครงข้อหมุนแบบฟิงค์อาจดัดแปลงมาใช้เป็นแบบผสม (Compound) หรือแบบพัด (Fan) เมื่อต้องการความยาวมากขึ้น แบบต่างๆ ของโครงข้อหมุนอาจดัดแปลงได้อีก เช่น ทำส่วนล่างทางโครง (Bottom Chord) ให้โค้งขึ้นเพื่อให้ช่วงกลางหรือเพดานสูงขึ้นซึ่งจะเรียก แบบแพรตต์โค้ง (Cambered Pratt) แบบโฮว์โค้ง แบบฟิงค์โค้ง เป็นต้น

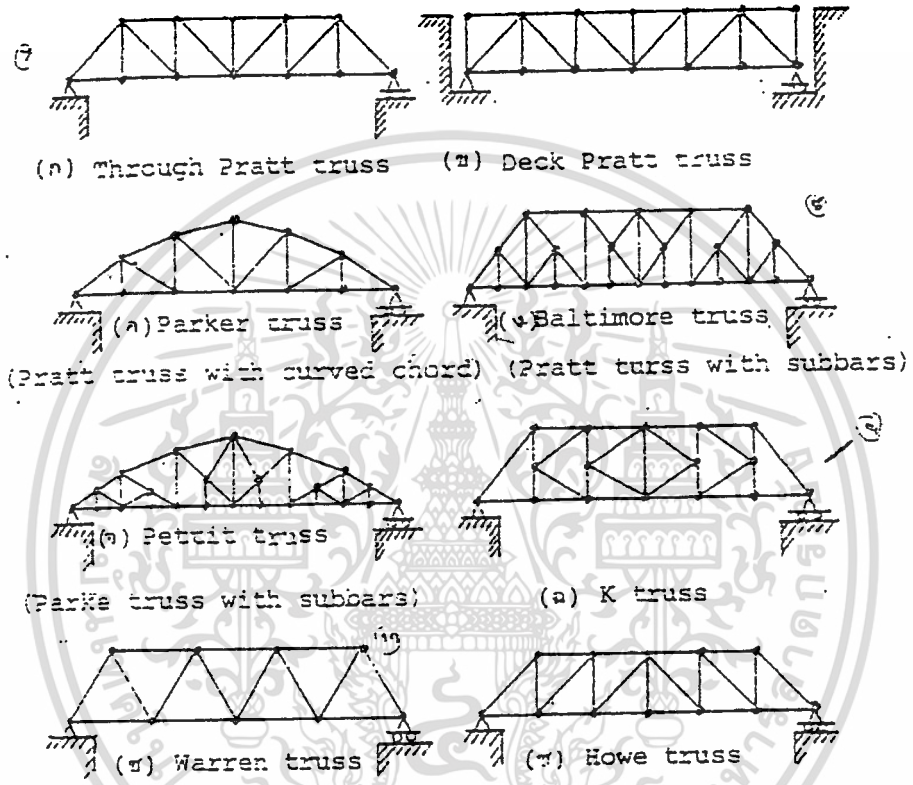
สำหรับโครงหลังคาแบบลาดๆ หรือเกือบแบนราบก็ใช้แบบวอเรน (Warren) แต่ก็อาจดัดแปลงแบบแพรตต์ หรือ แบบโฮว์ มาใช้ก็ได้ ส่วนแบบฟันเลื่อย (Saw Tooth) ต้องมีเสาเกะกะอยู่บ้างแต่ได้แสงสว่างดี โดยหันช่องแสงไปทางทิศเหนือ จึงมักเรียกว่า โครงหลังคาแบบแสงเหนือ ส่วนโครงหลังคาแบบคันธนู (Bow String) และแบบพระจันทร์เสี้ยว (Crescent) เหมาะสำหรับทำเป็นโครงหลังคาโรงรถ หรือโรงเครื่องบินขนาดย่อม โครงหลังคาแบบนี้เหมาะแก่การมุ่งหลังคาเพียงการใส่แปไม้ชิดๆ กันที่ส่วนบนตามขนาดของวัสดุมุ่ง แล้วปูลงไปก็ใช้ได้ โครงข้อหมุนแบบโค้งอาร์ต (Arch) ค่อนข้างจะแพงอยู่หน่อย จึงใช้เฉพาะช่วงที่มีความยามมากๆ และตรงช่วงกลางยกสูง เช่น ใช้เป็นโรงรถ โรงยิมเนเซี่ยม โรงอาหาร โรงละคร โรงแสดงนิทรรศการ เป็นต้น โครงข้อหมุนที่เป็นแบบอาร์ตที่แสดงในรูปเป็นแบบที่มีจุดยึด สามจุด (Three hinge arch) โดยมีจุดยึดหมุนสองจุดที่ตอม่อหรือฐานรอง และมีจุดยึดหมุนอีกจุดหนึ่งตรงจุดยอด (Crown) ของโค้งจึงเป็นโครงสร้างแบบ Indeterminate แบบนี้อาจใช้ท่อนหรือแท่งเป็นตัวยึด (Tie Rod) ของแรงปฏิกิริยาในแนวนอนก็ได้ โครงโค้งอาร์ตที่เป็นแบบ Indeterminate ก็ได้แก่แบบที่มีจุดยึดแน่นที่ฐานรองทั้งสอง (Fixed arch)



รูปที่ 2.9 แบบต่างๆ ของโครงหลังคา

2.5.2 โครงสะพาน แบบต่างๆโครงสะพานที่นิยมใช้แสดงไว้ดังรูปที่ 2.10 ถ้าสังเกตให้ดีจะเห็นได้ว่ามีแม่แบบอยู่ 4 แบบเป็นสำคัญ แบบอื่นๆ ทางโครงสะพานได้ดัดแปลงมาจากแม่แบบทั้ง 4 ทั้งสิ้น โครงสะพานแบบแพรตต์ (Pratt) เป็นที่นิยมใช้เป็นมาก่อนแบบอื่นๆ แต่ปัจจุบันนิยมใช้สะพานแบบวอเรนชนิดที่มีส่วนตั้งด้วย (warren with vertical) ทั้งนี้การต่อต้องใช้หมุด(Pin Connection) โครงสะพานแบบโฮว์ (Howe) ก็นิยมใช้กันมาก่อนโดยทำเป็นโครงสะพานไม้ หรือร่วมกับเหล็กแล้วแต่ความถาวรเป็นสำคัญ แต่ปัจจุบันนิยมทำเป็นโครงสร้างคอนกรีตเสริมเหล็ก โครงสะพานแบบวอเรนแท้ๆ ใช้กันมากในงานรถไฟ หรือสะพานถนนช่วงแคบชั่วคราว โครงสะพานแบบวอเรนชนิดที่ส่วนตั้งเป็นที่นิยมในปัจจุบันสำหรับสะพานที่ต่อด้วยหมุดยึด (Rivetd joint) โครงสะพานแบบต่างๆ ที่กล่าวข้างต้นนี้ เป็นแบบที่คอร์ดบนและคอร์ดล่างขนานกัน ซึ่งเหาะสำหรับสะพานที่มีความยาวตั้งแต่ 60 ถึง 70 เมตร ถ้าช่วงยาวมารกกกว่านี้นิยมใช้โครงสะพานที่คอร์ดไม่ขนานกัน ซึ่งจะช่วยลดน้ำหนักของโครงสร้างของสะพานได้ เช่น โครงสะพานแบบปากเกอร์ (Parker) ซึ่งเป็นแบบแพรตต์แต่มีแบบคอร์ดของสะพานไม่ขนานกันใช้ได้กับช่วงยาวถึง 120 เมตร ส่วนทะแยง (Diagonal) ควรมีความลาดเอียงทำมุมระหว่าง 45 ถึง 60 องศากับแนวราบจึงจะช่วยให้ประหยัดวัสดุก่อสร้าง ส่วนความลึกของคอร์ดของโครงสะพานจะเพิ่มมากขึ้นตามความยาวของช่วงสะพาน ทำให้ระยะของช่วง (panel) ยาวขึ้นและน้ำหนักของระบบพื้นมากขึ้น ฉะนั้นจึงหันมานิยมโครงสะพานแบบแยกชิ้นส่วน (Subdivided) ทั้งนี้เพื่อลดน้ำหนักของระบบพื้นให้น้อยลงและอยู่ในน้ำหนักพิกัด โครงสะพานแบบแพรตต์ที่มีคอร์ดขนานกันแต่แบ่งชิ้นส่วนระหว่างช่วงออกไป เรียกว่า โครงสะพานแบบ บัลติมอร์ (Baltimore) ซึ่งนิยมใช้กันในช่วงสะพานปานกลางมีพื้นที่ตื้นถี่ สำหรับโครงสะพานที่มีคอร์ดขนานกันแต่ช่วงสะพานปานกลาง จำกัดความลึกของพื้น และระยะของช่วงสั้นๆ ก็ใช้โครงสะพานแบบวอเรนที่มีแบ่งชิ้นส่วน (Subdivided warren) ส่วนโครงสะพานที่มีคอร์ดไม่ขนานกันและแบ่งชิ้นส่วน เรียกชื่อใหม่เป็นแบบแพตตี้ (Pettit) หรือแบบแพนซิลวานเนีย (Pennsylvania) เหมาะสำหรับโครงสะพานที่มีช่วงยาวที่มีความลึกระหว่างคอร์ดมาก เส้นประ

ทางแนวนอนที่แสดงในรูปเพื่อยึดส่วนตั้งที่ยาวมากๆ เป็นส่วนที่เพิ่มเข้ามาเฉยๆ เรียกว่าชิ้นส่วนรอง (Secondary member) โครงสะพานแบบเค (K- Truss) เป็นโครงสะพานแบบแบ่งชิ้นส่วนแบบใหม่ มีทั้งชนิดที่คอร์ดของโครงสะพานขนานกันและไม่ขนานกัน แต่ยังไม่ได้รับความนิยมมากนัก



รูปที่ 2.10 แบบต่างๆ ของโครงสะพาน

2.6 สมมุติฐานและหลักการคำนวณหาแรงภายในโครงข้อหมุน

สมมุติฐานเบื้องต้น สมมุติฐานที่ใช้ในการคำนวณหาแรงภายในโครงข้อหมุน มีดังนี้คือ

1. เส้นที่ลากผ่านศูนย์ถ่วงของหน้าตัดของชิ้นส่วนต่างๆ ต้องพบกันที่จุดต่อ

2. จุดต่อของโครงข้อหมุน ถือว่าหมุนได้และไม่มีความฝืด

3. แรงหรือน้ำหนักบรรทุกที่กระทำ ต้องกระทำที่จุดต่อเท่านั้น ฉะนั้นจึงไม่มีโมเมนต์ดัดในโครงชิ้นส่วนใดๆ ของโครงข้อหมุนเลย

4. ความยาวของแต่ละโครงข้อหมุน ต้องไม่เปลี่ยนแปลงก่อนและหลังรับน้ำหนัก

2.7 หลักในการคำนวณ

การคำนวณหาแรงภายในโครงสร้างใช้หลักการดังนี้

ก. แรงหรือน้ำหนักบรรทุกภายนอก และแรงปฏิกิริยาที่กระทำบนโครงสร้างต้องอยู่ในภาวะสมดุลย์

ข. แรงดึงหรือแรงอัดที่เกิดขึ้นภายในชิ้นส่วนต่างๆ ของโครงข้อหมุน จะต้องพบกันที่จุดต่อ ซึ่งอาจเป็นจุดที่มีแรงหรือน้ำหนักภายนอกกระทำอยู่ด้วย เมื่อแรงต่างๆ เหล่านี้พบกันที่จุดๆ หนึ่งต้องอยู่ในภาวะสมดุลย์

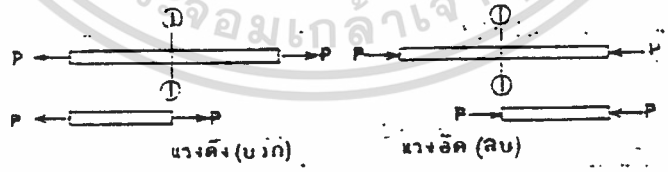
ค. ชิ้นส่วนใดๆ สามารถแทนได้ด้วยแรง 2 แรง ที่มีขนาดเท่ากัน แต่มีทิศตรงกันข้ามกัน และกระที่ปลายของชิ้นส่วนทั้งสองนั้น

ง. ส่วนหนึ่งส่วนใดของโครงสร้างที่นำมาพิจารณา จะต้องอยู่ในสภาวะสมดุลย์ด้วยแรงหรือน้ำหนักบรรทุกภายนอกที่กระทำ ณ จุดต่อ และแรงที่เกิดขึ้นภายในชิ้นส่วนของโครงข้อหมุนในส่วนที่ไม่ได้นำมาพิจารณา

เครื่องหมาย

แรงดึง ซึ่งเป็นแรงที่เกิดขึ้นต้านแรงกระทำภายนอกในทิศทางพุ่งออกจากรอยตัดให้ถือว่ามีค่าเป็นบวก

แรงอัด ซึ่งเป็นแรงที่เกิดขึ้นต้านแรงกระทำภายนอกในทิศทางพุ่งเข้าหารอยตัดให้ถือว่ามีค่าเป็นลบ



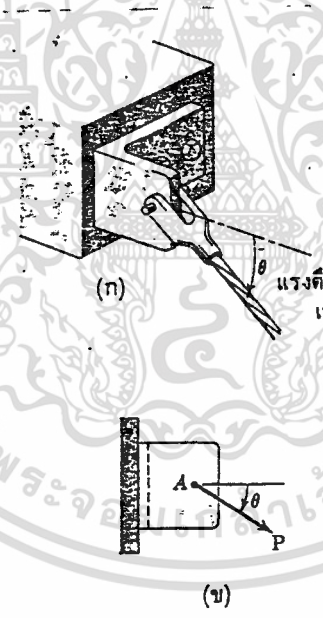
รูปที่ 2.11

บทที่ 3
ทฤษฎีที่ใช้ในการวิเคราะห์

3.1 การวิเคราะห์เวคเตอร์

3.1.1 เวคเตอร์ของแรง (Force Vector)

ก่อนที่จะได้ศึกษาระบบของแรง จำเป็นที่จะต้องศึกษาคุณสมบัติของแรงเฉพาะแรงเดียว ก่อนความหมายของแรงตามที่ได้นิยามเอาไว้คือ การกระทำของวัตถุหนึ่งต่ออีกวัตถุหนึ่ง และพบว่าแรงเป็นปริมาณเวคเตอร์ เพราะว่าผลลัพธ์ของมันขึ้นอยู่กับขนาดและทิศทางที่มันกระทำ และแรงยังสามารถรวมกันได้ตามกฎของ สี่เหลี่ยมด้านขนาน (Parallelogram Law) ในรูปที่ 3.1 แรงดึงในเส้นเชือก P ที่กระทำต่อแผ่นยึดดังรูปที่ 3.1(ก) แสดงไว้ด้วยเวคเตอร์ของแรงที่มีขนาดเท่ากับ P ผลของแรงที่กระทำต่อแผ่นยึดนี้ขึ้นอยู่กับขนาดของแรง P และมุม θ และตำแหน่งของจุดที่กระทำ A เมื่อสิ่งหนึ่งในสามสิ่งนี้เปลี่ยนไป ผลที่มีต่อแผ่นยึดนี้ก็จะเปลี่ยนแปลงไปด้วย เช่น แรงภายในตัวนอตที่ดึงแผ่นยึดให้ติดกับฐาน หรือความเค้นในเนื้อแผ่นยึดเปลี่ยนไป ดังนั้นจะเห็นได้ว่า ในการคิดเรื่องการกระทำของแรงแล้วจะต้องรู้ให้ครบทั้งสามสิ่งคือ ขนาด ทิศทาง และจุดกระทำ



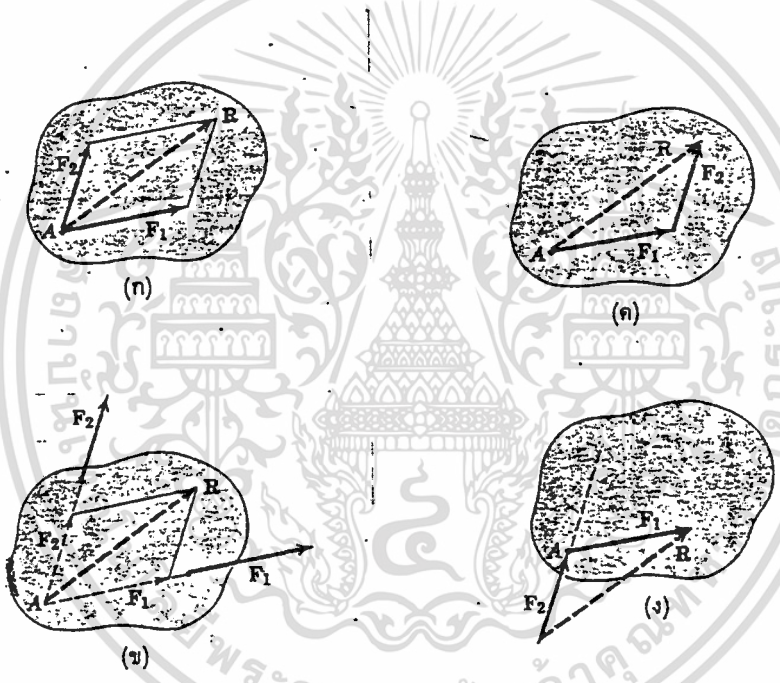
รูปที่ 3.1

ผลของแรงที่กระทำต่อวัตถุสามารถแบ่งออกได้เป็น 2 ประการคือ ผลภายนอกและภายใน ในรูปที่ 3.1 ผลภายนอกของแรง P คือ แรงปฏิกิริยาที่กระทำต่อแผ่นยึดโดยนอตและฐาน ดังนั้นจะเห็นได้ว่าแรงภายนอกที่กระทำต่อวัตถุมี 2 ชนิด คือ แรงกระทำ (Applied force) และแรงปฏิกิริยา (Reactive force) ผลภายในของแรง P ที่มีต่อแผ่นยึดคือ ความเค้น (Stress) และความเครียด (Strain) ที่เกิดขึ้นในแผ่นยึด ความสัมพันธ์ระหว่างความเค้นและความเครียดขึ้นอยู่

กับคุณสมบัติของวัสดุ ซึ่งสามารถศึกษาได้จากวิชา ความแข็งแรงของวัสดุ (Strength of materials) ความยืดหยุ่น(Elasticity) และความหยุ่นเหนียว (Plasticity)

จากกฎข้อที่ 3 ของนิวตัน เมื่อมีแรงมากระทำจะต้องมีแรงปฏิกิริยาขนาดเท่ากันและทิศทางตรงข้ามเสมอ แรงต่างจะปรากฏชัดเมื่อได้แยกวัสดุที่จะคิดออกจากวัสดุอื่นๆ และเขียนเป็นผังวัสดุอิสระ (Free body diagram) ซึ่งเริ่มแสดงแรงที่กระทำต่อวัตถุ(ไม่ใช่แรงที่กระทำโดยวัตถุ) ข้อผิดพลาดที่คิดแรงในแต่ละคู่นี้จะเกิดได้ง่าย ถ้าไม่พิจารณาความแตกต่างระหว่างแรงที่กระทำและแรงปฏิกิริยาให้ดี

แรง F_1 และ F_2 ตัดกันที่จุดๆ หนึ่ง แรงคู่นี้เรียกว่า แรงร่วมจุด (Concurrent forces) สามารถรวมกันได้ตามกฎของสี่เหลี่ยมด้านขนานในระนาบที่เกิดจากแรงทั้งสอง ผลลัพธ์คือ R ดังแสดงในรูปที่ 3.2 (ก)

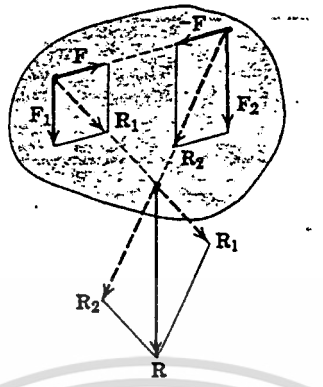


รูปที่ 3.2

การรวมแรงสองแรงเขียนให้อยู่ในรูปทางคณิตศาสตร์ได้โดยสมการเวกเตอร์

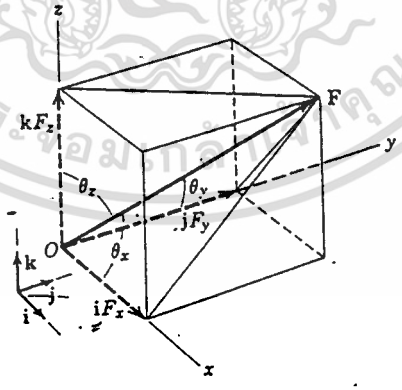
$$R = F_1 + F_2$$

นอกเหนือจากการรวมแรงเพื่อให้ได้ผลลัพธ์แล้ว บางครั้งต้องแยกแรงออกเป็นแรงย่อย (Components) ในทิศทางที่ต้องการ ดังนั้นแรง R ในรูปที่ 3.2(ก) สามารถแทนหรือแยกออกเป็นแรงย่อย F_1 และ F_2 ไปในทิศทางที่กำหนดให้ด้วยการเขียนสี่เหลี่ยมด้านขนานแทนแรง



รูปที่ 3.3

ในรูปที่ 3.3 เป็นกรณีพิเศษในการรวมแรง F_1 และ F_2 ซึ่งขนานกัน ในขั้นแรก เพิ่มแรง F และ -F ที่เท่ากัน แต่ทิศทางตรงกันข้ามและอยู่ในแนวเดียวกัน และมีขนาดที่เหมาะสมรวม F เข้ากับ F_1 ได้แรงลัพธ์ R_1 และรวม -F เข้ากับ F_2 ได้แรงลัพธ์ R_2 แล้วรวม R_1 และ R_2 ด้วยกัน จะได้ R ซึ่งประกอบด้วย ขนาด ทิศทาง และแนวแรงที่กระทำ วิธีการนี้มีประโยชน์สำหรับรวมแรงที่ขนานกันในทางกราฟิก



รูปที่ 3.4

Cartesian Vectors

การปฏิบัติการของพีชคณิตเวคเตอร์ (Vectors Algebra) จะมีประโยชน์อย่างมากในการประยุกต์ เพื่อนำมาใช้แก้ปัญหาในโครงสร้าง 2 มิติและ 3 มิติ ถ้าแรงอยู่ในรูปของ Cartesia Vector ในหัวข้อนี้จะแสดงให้เห็นถึงหลักการโดยทั่วไป และในหัวข้อถัดไป จะประยุกต์วิธีนี้เพื่อใช้แก้ปัญหาที่เกี่ยวกับแรงและโมเมนต์

1. กฎมือขวาในระบบพิกัดฉาก (Right-Handed Coordinate System) กฎมือขวาจะใช้เป็นหลักเกณฑ์ในการพัฒนาทฤษฎีของพีชคณิตเวคเตอร์ที่จะตามมา Cartesia Vectors กล่าวว่า กฎมือขวาจะใช้นิ้วหัวแม่มือขวาเป็นตัวกำหนดทิศทาง การแสดงค่าแกน Z ที่เป็นบวก เมื่อเรากำมือจากทางแกน X ไปยังแกน Y

2. ส่วนประกอบของ Vector เวคเตอร์ A อาจจะมี หนึ่ง สอง หรือ สามส่วน ตามแกน X,Y และ Z ขึ้นอยู่กับทิศทางที่เวคเตอร์นั้นชี้ไปสัมพันธ์กับแกนใดบ้าง โดยทั่วไป เมื่อเวคเตอร์ A มีทิศทางสัมพันธ์กันทั้งแกน X,Yและ Z เราสามารถแก้สมการได้เป็น

$$A = A' + A_Z \quad \text{เมื่อ} \quad A' = A_X + A_Y$$

เมื่อรวมสมการเหล่านี้ A จะแสดงถึงผลรวมของ Vector ในสมการด้วย

$$A = A_X + A_Y + A_Z$$

3. เวคเตอร์หนึ่งหน่วย (Unit Vector) โดยทั่วไป Unit Vector จะเป็นเวคเตอร์ที่ขนาดเท่ากับ 1 ถ้า A เป็นเวคเตอร์ที่มีขนาดไม่เท่ากับ 0 แล้ว Unit Vector จะมีทิศทางเดียวกับเวคเตอร์ A"

$$U_A = \frac{A''}{A}$$

Cartesian Unit Vector ในระบบ 3 มิติชุดของ Cartasian Unit Vector ค่า i,j,k จะถูกนำไปใช้กำหนดทิศทางของแกน X,Y และ Z ตามลำดับ ค่าจะเป็นบวกหรือลบขึ้นอยู่กับว่า มันจะชี้ไปตามแกนเป็นลบ ของแกน X,Y และ Z ตามกฎมือขวา เราจะได้ Unit Vector ที่เป็นบวก

Cartesian Unit Repersentation การใช้ Cartesian Unit Vector จะประกอบไปด้วยทิศทางไปตามแกน โดยแทนในรูปของ i,j,k จะได้เป็น

$$A = A_X i + A_Y j + A_Z k$$

ข้อได้เปรียบอย่างหนึ่งของการเขียนเวคแตอร์ในรูปของ Cartesian Component คือ ในแต่ละส่วนประกอบจะมีรูปแบบของขนาดและทิศทาง สำหรับส่วนประกอบเวคเตอร์ แต่ละส่วนจะถูกแยกออกจากกัน และจะได้รับการแสดงในรูปที่ง่ายต่อการคำนวณ เกี่ยวกับพีชคณิตเวคเตอร์โดยเฉพาะในระบบสามมิติ

ขนาดของ Cartesian Vector

เราสามารถหาขนาดของ Vector ได้ดังนี้

$$A = \sqrt{Ax2 + Ay2 + Az2}$$

ขนาดของ Vector จะเท่ากับ ค่าบวกของรากที่สอง ของผลรวมค่ายกกำลังของ Vector ในแต่ละแกน

ทิศทางของ Cartesian Vector

ทิศทางของ Vector A จะระบุโดย มุม α , β และ γ ซึ่งวัดระยะระหว่างหางของเวคเตอร์กับแกน X,Y และ Z มุมจะอยู่ระหว่าง 0-180 องศา การพิจารณาα , β และ γ จะแสดงออกมา

$$\cos\alpha = \frac{Ax}{A}$$

$$\cos\beta = \frac{Ay}{A}$$

$$\cos\gamma = \frac{Az}{A}$$

$$U_A = \frac{A}{A} = \frac{Ax}{A}I + \frac{Ay}{A}j + \frac{Az}{A}k$$

$$A = \sqrt{Ax2 + Ay2 + Az2}$$

$$U_A = \cos\alpha i + \cos\beta j + \cos\gamma k$$

$$\cos^2\alpha i + \cos^2\beta j + \cos^2\gamma k = 1$$

$$A = A\,U_A$$

$$= A\cos\alpha i + A\cos\beta j + A\cos\gamma k$$

$$= A_X i + A_Y j + A_Z k$$

การบวกและลบของ Cartesian Vector

การคำนวณเกี่ยวกับเวคเตอร์ของ 2 หรือมากกว่า 2 เวคเตอร์ขึ้นไปจะง่ายขึ้นมากถ้าเราแสดงเวคเตอร์ออกมาในรูป Cartesian Component ตัวอย่างเช่น เวคตอร์ 2 ตัว คือ A และ B ทั้งสองมีทิศทาง ถ้า

$$A = A_X i + A_Y j + A_Z k$$

$$B = B_X i + B_Y j + B_Z k$$

แล้วผลลัพท์ของเวคเตอร์ R จะแสดงออกมาดังนี้

$$R = (A-B)$$

$$= (A_X - B_X)i + (A_Y - B_Y)j + (A_Z - B_Z)k$$

Concurrent force System

การรวมแรงหลายๆ แรงในระบบแรง เราสามารถหาได้โดยวิธีการเดียวกัน ซึ่งสามารถเขียนได้เป็น

$$F_R = \sum F = \sum F_X i + \sum F_y j + \sum F_Z k$$

เมื่อ $\sum F_X$, $\sum F_y$, $\sum F_Z$ แสดงถึงผลรวมของพีชคณิตในแกน X,Y และ Z หรือ i,j,k ของเวคเตอร์ประกอบของแรงแต่ละตัวในระบบแรงที่กระทำที่จุด O ดังในรูปที่ 3.4 สามารถแยกออกเป็นแรงย่อย F_X , F_Y , F_Z ซึ่ง

$$F_X = F \cos \theta x \qquad F = \sqrt{F_X^2 + F_Y^2 + F_Z^2}$$

$$F_Y = F \cos \theta y \qquad F = iF_X + jF_Y + kF_Z$$

$$F_Z = F \cos \theta z \qquad F = F(i \cos \theta x + j \cos \theta y + k \cos \theta z)$$

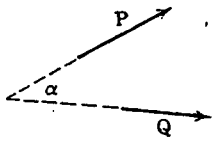
i,j,k เป็นเวคเตอร์หนึ่งหน่วย (Unit vector) ตามแกน x,y และ z ตามลำดับ และการวางตำแหน่งของแกนสามารถวางในลักษณะใดก็ได้ แต่ถือความสะดวกเป็นหลัก อย่างไรก็ดี ระบบของแกนต้องเป็นไปตามกฎมือขวา เพื่อที่ให้ตำแหน่งสัมพันธ์ของแกน x,y และ z คงเดิมสำหรับปัญหาสองมิติเช่น เมื่อไม่มีแกน z แรงย่อยเขียนได้ดังนี้

$$F_X = F \cos \theta x \qquad F_Y = F \cos \theta y \qquad \tan \theta x = \frac{F_Y}{F_X}$$

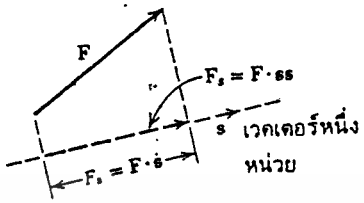
เพื่อกำจัดความสับสนในการเขียนแรงรวมและแรงย่อย แรงย่อยควรเขียนด้วยเส้นประและแรงรวมเขียนด้วยเส้นเต็ม(หรือกลับกัน) เมื่อเข้าใจตามนี้จะทำให้เห็นชัดระหว่างแรงรวมและแรงย่อย

แรงย่อยในพิกัดฉากของแรง F (หรือเวกเตอร์อื่นๆ) อาจจะเขียนได้อีกอย่างหนึ่ง โดยใช้คุณสมบัติการคูณกันของเวคเตอร์ที่เรียกว่า ผลคูณแบบสเกลาร์ (dot or scalar product) จากนิยามผลคูณแบบสเกลลาร์ของสองเวคเตอร์ P และ Q ในรูปที่ 3.5(ก) คือ ผลคูณของขนาดของเวคเตอร์ทั้งสอง และค่า cosine ของมุมระหว่างเวคเตอร์ทั้งสอง ผลคูณนี้สามารถคิดเป็นภาพฉายหรือองค์ประกอบ P cos α ของ P ในทิศทาง Q คูณด้วยขนาดของ Q หรือเป็นภาพฉายหรือองค์ประกอบ Q cos α ของ Q ในทิศทาง P คูณด้วยขนาดของ P ก็ได้ ในทั้งสองกรณีจะเห็นได้ว่าผลคูณแบบสเกลลาร์ของสองเวคเตอร์เป็นปริมาณสเกลลาร์ และเขียนได้ดังนี้

$$P \cdot Q = PQ \cos \alpha$$



(ก)



(ข)

รูปที่ 3.5

ดังนั้น แรงย่อย $F_x = F \cos \theta_x$ ของแรง F ในรูปที่ 3.4 สามารถเขียนได้เป็น $F_x = F \cdot i$ ซึ่ง i คือ เวคเตอร์หนึ่งหน่วยในทิศทาง x สำหรับกรณีทั่วๆ ไป ถ้า s เป็นเวคเตอร์หนึ่งหน่วยในทิศทางที่กำหนดให้ แรงย่อยของ F ในทิศทาง s (รูปที่ 3.5(ข)) มีขนาดเป็น $F_s = F \cdot s$ คูณกับเวคเตอร์หนึ่งหน่วย s เขียนได้เป็น $F_s = (F \cdot s)s$ หรือ $F_s = F \cdot ss$

ถ้า s มีโคซายน์แสดงทิศทาง(direction cosine) α,β,γ และ F มีโคซายน์แสดง ทิศทาง l , m , n ดังนั้นแรงย่อยของ F ในทิศทาง s คือ

$$F_s = F \cdot s = F(il + jm + kn). (i\alpha + j\beta + k\gamma)$$

$$= F(l\alpha + m\beta + k\gamma)$$

เพราะว่า

$i.i = j.j = k.k = 1$ และ $i.j = j.i = i.k = k.i = j.k = k.j = 0$

3.1.2 ผลลัพท์ของระบบแรง (Force System Resultant)

จากสภาวะการสมดุลย์กันของระบบแรง ณ จุดที่พบกันที่จุดใดๆ โดยแรงลัพท์ของ ณ จุดนั้นต้องมีค่าเท่ากับศูนย์

หลักการที่เพิ่มขึ้นมาอีกอย่างหนึ่งคือ โมเมนต์ ซึ่งจะต้องหาโมเมนต์ของแรงรอบจุดหรือรอบแกนที่พิจารณา โดยจะแสดงให้เห็นถึงวิธีการพิจารณา แรงลัพท์ของระบบแรงที่ไม่พบกันที่จุดเดียว ซึ่งมีความสำคัญต่อการประยุกต์ของสมการระบบแรง ของโครงสร้างเป็นอย่างมาก ผลของระบบแรงจะมีอิทธิพลต่อสมดุบลหรือการเคลื่อนที่ของวัตถุหรือ โครงสร้าง ดังนั้น เราจำเป็นจะต้องพิจารณาผลลัพท์ของแรง

1. Cross Product

โมเมนต์ของแรงหาได้โดยใช้ Cartesian Vector ดังนั้น สิ่งที่จำเป็นต้องทราบ คือ ความรู้เกี่ยวกับ Vector และวิธีการคูณ Vector แบบ Cross Product Method

Cross Product ของเวคเตอร์ 2 ตัว คือ A และ B ได้ผลลัพท์เป็น C สามารถเขียนได้ดังนี้

$$C = A \times B$$

2. ขนาด(Magnitude)

ขนาดของ C จะระบุเป็นผลคูณขนาดของ A และ B และ ค่าของ sin ของมุม θ ระหว่างหางของเวคเตอร์ ($0<\theta<180$)

$$C = AB \sin\theta$$

3. ทิศทาง(Direction)

เวคเตอร์ C ทิศทางที่ตั้งฉากกับระนาบ A และ B ตามกฎมือขวา

$$C = A \times B = AB \sin\theta \, U_C$$

เมื่อ $AB \sin\theta$ = ขนาดของ C

U_C = Unit Vector แสดงทิศทางของ C

สูตรของ Cartesian Vector

$$ixj = k \quad jxk = i \quad kxi = j$$

$$ixk = -j \quad jxi = -k \quad kxj = -i$$

$$ixi = 0 \quad jxj = 0 \quad kxk = 0$$

$$AXB = (A_Xi + A_Yj + A_Zk) + (B_Xi + B_Yj + B_Zk)$$

$$= A_XB_X(ixj) + A_XB_Y(ixj) + A_XB_Z(ixk) + A_YB_X(jxi) + A_YB_Y(ixj) + A_YB_Z(jxk)$$

$$+A_ZB_X(kxi) + A_ZB_Y(kxj) + A_ZB_Z(kxk)$$

$$AXB = (A_YB_Z - A_ZB_Y)i - (A_XB_Z - A_ZB_X)j + (A_XB_Y - A_YB_X)k$$

$$AXB = \begin{vmatrix} i & j & k \\ A_X & A_Y & A_Z \\ B_X & B_Y & B_Z \end{vmatrix}$$

3.1.3 โมเมนต์(Moment)

แรงนอกจากจะพยายามทำให้วัตถุเคลื่อนที่ ไปในทิศทางที่มันกระทำแล้ว ยังพยายามที่จะทำให้วัตถุหมุนรอบแกนที่ไม่ตัดกับแนวแรง หรือขนานกับแนวแรงนั้น ความพยายามนี้เรียกว่า โมเมนต์ (Moment) ของแรงรอบแกนหนึ่ง โมเมนต์อาจจะเรียกว่า แรงบิด(Torque) ก็ได้

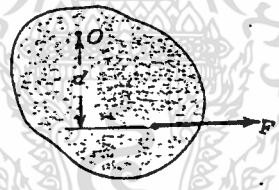
รูปที่ 3.6

ที่ 3.6(ก) แรง R กระทำต่อวัตถุก้อนหนึ่งที่จุด A แกน 0-0 เป็นแกนที่ไม่ตัดกับแนวแรง R แรงR สามารถแยกเป็นแรงย่อยได้สองแรงคือ แรงย่อยของแรง P ซึ่งขนานกับแกน 0-0 และแรงย่อย F ซึ่งอยู่ในระนาบที่ตั้งฉากกับแกน 0-0 จะเห็นได้ชัดว่า แรงย่อย P ไม่สามารถที่จะหมุนวัตถุรอบแกน 0-0 ได้ แต่แรงย่อย F ซึ่งอยู่ในระนาบที่ตั้งฉากกับแกน 0-0 จะพยายามหมุนวัตถุรอบแกน 0-0 ขนาดของความพยายามนี้เป็นสัดส่วนกับขนาดของแรง F และแขนของโมเมนต์ d ขนาดของโมเมนต์คือ

$$M = Fd$$

ทิศทางของโมเมนต์ขึ้นอยู่กับทิศทางของแรง F ที่พยายามจะหมุนวัตถุนั้นการกำหนดทิศทางนี้ใช้กฎมือขวา ในรูปที่ 3.6 (ข) กำหนดให้โมเมนต์ของแรง F รอบแกน 0-0 แทนได้ด้วยเวคเตอร์ ที่ชี้ไปตามหัวแม่มือในขณะที่นิ้วอื่นๆ จะชี้ทิศทางความพยายามที่จะหมุน โมเมนต์ M นี้เป็นไปตามกฎการรวมตัวของเวคเตอร์ทุกประการ และสามารถคิดให้เป็นเวคเตอร์เลื่อนไถล (Sliding vector) ที่แนวกระทำทับกับแกนหมุนได้ หน่วยหลักของโมเมนต์คือ นิวตัน-เมตร (N.m)

ในการคำนวณเกี่ยวกับเรื่องแรงที่อยู่ในระนาบเดียวกัน มักจะพูดถึงโมเมนต์ว่า เป็นโมเมนต์รอบจุดๆ หนึ่ง ซึ่งแท้ที่จริงแล้ว โมเมนต์นั้นเป็นโมเมนต์ที่แกนตั้งฉากกับระราบเดียวกันนั้น และผ่านจุดที่กล่าวถึง ดังนั้นโมเมนต์ของแรง F ที่ผ่านจุด O ในรูปที่ 3.7 จึงมีขนาด M_O = Fd และทิศทางการหมุนทวนเข็มนาฬิกา สำหรับแรงที่อยู่ในระนาบเดียวกันนี้ ไม่จำเป็นที่ต้องแทนโมเมนต์ด้วยเวคเตอร์เพราะว่าเวคเตอร์โมเมนต์ จะมีอยู่สองทิศทางคือ ชี้ออกจากกระดาษ (ทวนเข็ม) และพุ่งเข้าหากระดาษ (ตามเข็ม) เวคเตอร์โมเมนต์ที่ขนานกัน รวมกันได้โดยนำขนาดมารวมกันแบบสเกลลาร์ และทิศทางอาจจะกำหนดได้โดย ใช้เครื่องหมายบวก(+) สำหรับโมเมนต์ทวนเข็ม และลบ(-) สำหรับโมเมนต์ตามเข็ม หรืออาจจะสลับกันก็ได้ ซึ่งการกำหนดเครื่องหมายนี้จะต้องใช้ให้เหมือนกันตลอด



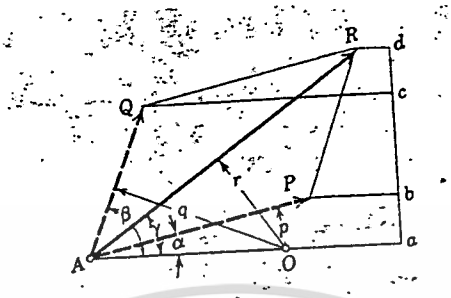
รูปที่ 3.7

หลักที่สำคัญอันหนึ่งคือ ทฤษฎีของวารียอง(Varignon's theorem) หรือ หลักของโมเมนต์(Principle of moment) ซึ่งกล่าวว่าแรงที่อยู่ในระนาบเดียวกัน โมเมนต์ของแรงรวมรอบจุดใดๆ มีค่าเท่ากับผลรวมของโมเมนต์ของแรงย่อยรอบจุดนั้น เพื่อที่เป็นการพิสูจน์ข้อความนี้ พิจารณาแรง R ซึ่งมีค่าเท่ากับผลรวมของ P และ Q กระทำที่จุด A ดังรูปที่ 3.8 เป็นจุดใดๆ ที่เลือกให้เป็นแกนของโมเมนต์ ลากเส้น AO แล้วหาภาพฉายของเวคเตอร์ทั้งสามบนเส้นที่ตั้งฉากกับ AO ลากแขนของโมเมนต์ p , q , r ของแรงทั้งสามไปยังจุด O และกำหนดมุมของเวคเตอร์เป็น α,β,γ ดังแสดงในรูป จากรูปสี่เหลี่ยมด้านขนานซึ่งต้องมีขนานซึ่งมีด้านเป็น P และ Q จะเห็นได้ว่า

$$\overline{ac} = \overline{bd} \text{ และ } \overline{ad} + \overline{bd} = \overline{ab} + \overline{ac}$$

หรือ

$$R \sin\gamma = P \sin\alpha + Q \sin\beta$$



รูปที่ 3.8

คูณด้วย AO และแทนค่า p , q , r จะได้ $Rr = Pp + Qq$

ซึ่งเป็นการพิสูจน์ว่า โมเมนต์รอบจุดใดๆ มีค่าเท่ากับผลรวมของโมเมนต์ของแรงย่อยสองแรงรอบจุดนั้น ทฤษฎีของวารียองไม่ได้จำกัดว่าต้องเป็นแรงสองแรงย่อยเท่านั้น แต่ใช้ได้เช่นเดียวกัน เมื่อมีแรงย่อยสามแรงหรือมากกว่า เพราะว่าสามารถลดจำนวนแรงย่อยต่างๆ ให้เหลือสองแรงได้ หลักการนี้สามารถประยุกต์ใช้กับโมเมนต์ของเวคเตอร์ตรึง หรือ เวคเตอร์เลื่อนไถลได้

พิจารณาแรง F ซึ่งมีแรงดังแสดงในรูปที่ 3.9 (ก) O เป็นจุดใดๆ ที่ไม่ได้อยู่ในแนวแรงนี้ จุด O และแนวแรง F นี้ประกอบกันเป็นระนาบ a โมเมนต์ M_O ของแรง F รอบแกนที่ผ่านจุด O และตั้งฉากกับแนวระนาบมีขนาด $M_O = Fd$ ซึ่ง d คือระยะตั้งฉากจากจุด O ไปยังแนวแรง F โมเมนต์นี้ยังหมายถึงขอแรง F รอบจุด O ด้วย เวคเตอร์ M_O นี้ตั้งฉากกับระนาบและชี้ไปตามแกนที่ผ่านจุด O ทั้งขนาดและทิศทางของ M_O นี้ หาได้จากคุณสมบัติการคูณกันของเวคเตอร์ที่เรียกว่า ผลคูณแบบเวคเตอร์(Cross or vector product) ให้เวคเตอร์ r ชี้จากจุด O ไปยังจุดใดๆ ในแนวแรง F จากนิยามผลคูณแบบเวคเตอร์ของ r และ F เขียนได้เป็น r x F และมีขนาด (r sin α) F มีค่าเท่ากับ Fd ซึ่งก็คือขนาดของ M_O ทิศทางของโมเมนต์เป็นไปตามกฎของมือขวาตามที่ได้กล่าวมาแล้ว ดังนั้นเมื่อคิดให้ r และ F เป็นเวคเตอร์อิสระ รูปที่ 3.9 (ข) นิ้วหัวแม่มือ ชี้ทิศของ M_O นิ้วอื่นชี้ไปตามการหมุนของ r และ F โมเมนต์ของแรง F รอบแกนที่ผ่านจุด O เขียนได้เป็น

$$M_O = r \times F$$

ลำดับของเวคเตอร์จะต้องเป็น r x F เพราะ F x r จะได้เวคเตอร์ที่มีทิศทางตรงกันข้ามกับ M_O หรือ $F \times r = -M_O$

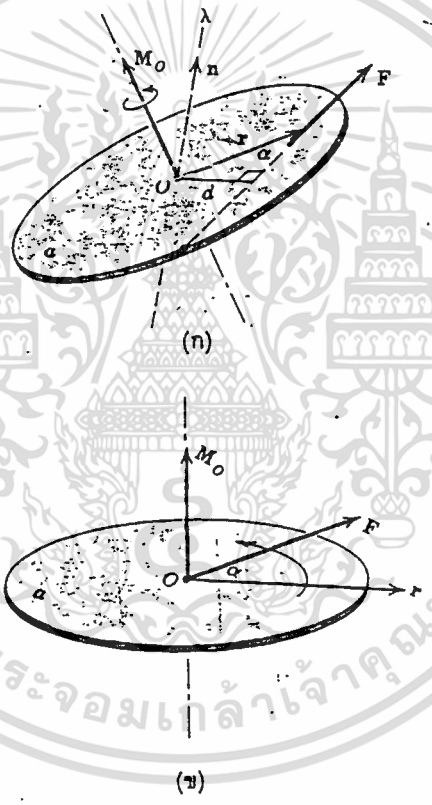
ค่าของ r x F สามารถเขียนได้ในรูปของดีเทอร์มินนท์ (Determinant) ดังนี้

$$M = \begin{vmatrix} i & j & k \\ r_X & r_Y & r_Z \\ F_X & F_Y & F_Z \end{vmatrix}$$

เมื่อ $r_X\ r_Y\ r_Z$ = เวคเตอร์ประกอบแนวแกน X,Y และ Z จากจุด O ไปยังจุดที่มีแรงกระทำ

$F_X\ F_Y\ F_Z$ = แรงในแนวแกน X,Y และ Z ซึ่งแสดงในรูปของเวคเตอร์

$M_O = (r_YF_Z - r_ZF_Y)i - (r_XF_Z - r_ZF_X)j + (r_XF_Y - r_YF_X)k$



รูปที่ 3.9

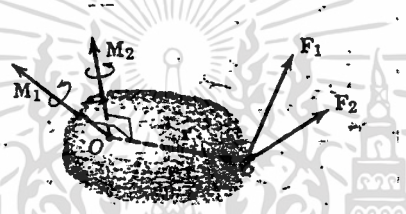
โมเมนต์ M_λ ของแรง F รอบแกน λ ใดๆ ที่ผ่านจุด O ในรูปที่ 3.9(ก) หาได้ดังนี้ ถ้า n เป็นเวคเตอร์หนึ่งหน่วยในทิศทาง λ โดยการใช้ผลคูณสเกลลาร์โมเมนต์ย่อยของ M_O ในทิศทาง λ คือ $M_O \cdot n$ ซึ่งเป็นขนาดของโมเมนต์ M_λ ของแรง F รอบแกน λ เวคเตอร์ของ M_λ หาได้โดยการเอาขนาดของ M_λ คูณด้วยเวคเตอร์หนึ่งหน่วยของ n จะได้

$$M_\lambda = (r \times F \cdot n)n$$

ซึ่ง r x F คือ M_O r x F . n คือ triple product และไม่จำเป็นต้องเขียน (r x F).n เนื่องจาก r x (F . n) ไม่มีความหมายเพราะว่าเวคเตอร์และสเกลลาร์ไม่สามารถจะหาผลคูณแบบเวคเตอร์ได้ สมการข้างต้น สามารถเขียนในรูปของดีเทอร์มิเนนท์ได้ดังนี้

$$M = \begin{vmatrix} r_X & r_Y & r_Z \\ F_X & F_Y & F_Z \\ \alpha & \beta & \gamma \end{vmatrix} \qquad (i\alpha + j\beta + k\gamma)$$

ซึ่ง α β γ คือ โคซายน์แสดงทิศทางของเวคเตอร์หนึ่งหน่วย n



รูปที่ 3.10

พิจารณาแรง F_1 และ F_2 ซึ่งตัดกันที่จุด A ดังแสดงในรูปที่ 3.10 เวคเตอร์บอกตำแหน่ง (Position Vector) ของ A จากจุด O ใดๆ คือ r เวคเตอร์สองเวคเตอร์รอบจุด O เนื่องมาจากแรงทั้งสองนั้น สามารถรวมกันได้ดังนี้

$$M_1 + M_2 = r \times F_1 + r \times F_2$$

ผลรวมคือ

$$M = r \times (F_1 + F_2)$$

สมการนี้คือ หลักของโมเมนต์สำหรับระบบสามมิติ ซึ่งจะกล่าวว่าผลรวมของโมเมนต์รอบจุด O ใดๆ ของแรงสองแรงที่ตัดกันที่จุดๆ หนึ่งมีค่าเท่ากับ โมเมนต์ของผลรวมของแรง $F_1 + F_2$ รอบจุด O นั้น ทฤษฎีนี้ใช้ได้ไม่เฉพาะแต่สองแรงที่ตัดกันเท่านั้น แต่ใช้ได้กับจำนวนแรงเท่าใดก็ได้ที่ตัดกัน

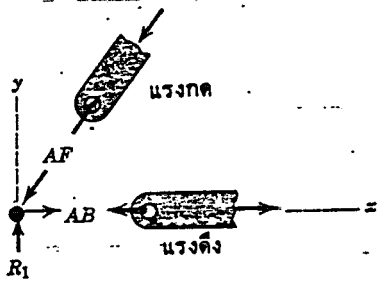
3.2 การวิเคราะห์โดยวิธีการใช้จุดต่อ(Method of Joints)

หลักการวิธีนี้ประกอบด้วยการสมดุลของแรงต่างๆ ที่กระทำต่อสลักข้อต่อที่จุดต่อต่างๆ ซึ่งก็คือการสมมุติของแรงที่ตัดกัน(concurrent force) นั่นเองและใช้สมการของการสมดุลเพียงสองสามสมการเท่านั้น วิธีการเริ่มต้นที่จุดต่อจุดใดก็ได้ที่แรงที่ทราบค่าอย่างน้อยหนึ่งแรง และมีแรงที่ไม่ทราบค่าไม่เกินสองแรง อาจจะเริ่มจากสลักทางซ้ายก่อนซึ่งผังวัสดุอิสระ(Free body diagram) แสดงไว้ในรูปที่ 3.11 จุดต่อต่างๆ กำหนดด้วยตัวอักษร แรงในชิ้นส่วนแสดงด้วยตัวอักษรสองตัวของชิ้นส่วนนั้น ทิศทางของแรงที่ควรจะเป็นดูได้จากหลักการดังนี้ ผังวัสดุอิสระของเสี้ยวหนึ่งของชิ้นส่วน AF และ AB ได้แสดงไว้ด้วยเพื่อให้เห็นชัดถึงแรงกระทำและแรงปฏิกิริยา จริงๆ แล้วชิ้นส่วน AB จะกดทางด้านซ้ายของสลัก ถึงแม้จะเขียนแรง AB ไว้ทางด้านขวาและชี้ออกจากสลัก ดังนั้นถ้าเขียนแรงไว้ทางด้านที่มีชิ้นส่วนอยู่เป็นหลัก แล้วแรงดึงจะชี้ออกจากสลักและแรงกด จะชี้เข้าหาสลักเสมอ ขนาดของแรง AF หาได้จากสมการ $\sum F_Y = 0$ และ AB หาได้จาก $\sum F_X = 0$

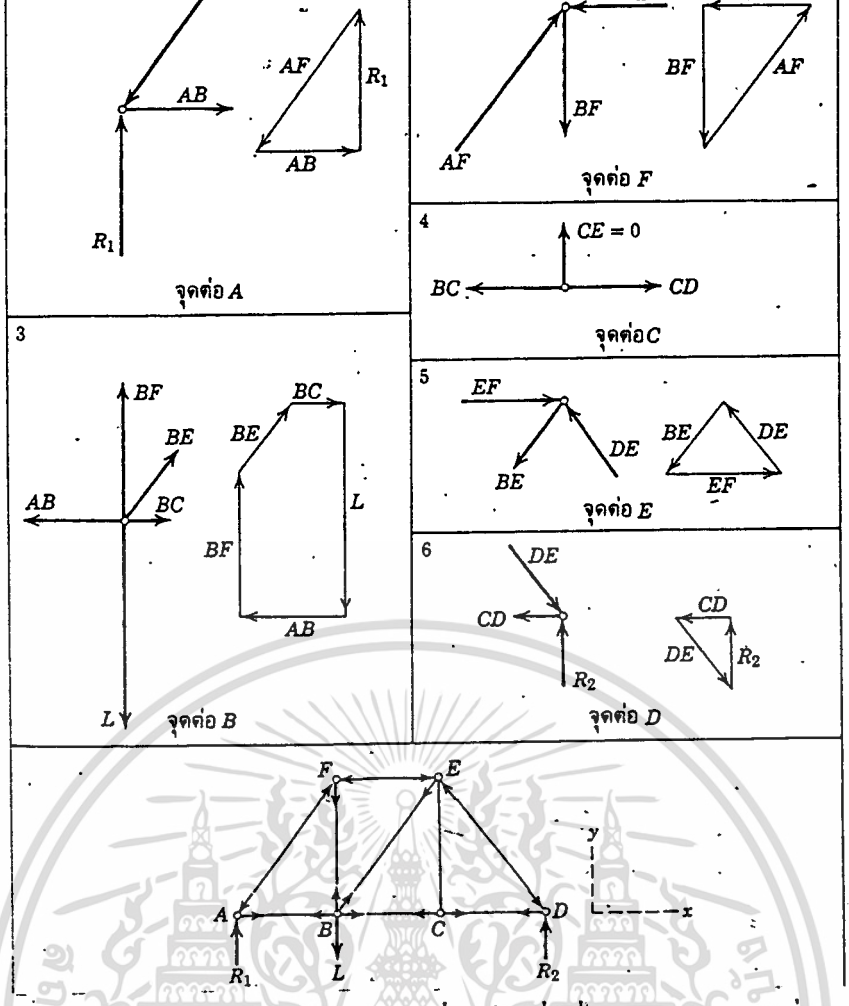
ต่อไปคิดที่หลัก F เพราะว่าในตอนนี้มีตัวไม่ทราบค่าเพียงสองค่าเท่านั้น คือ EF และ BF สลัก B,C,E และ D เป็นลำดับที่จะคิดต่อไป ผังวัสดุอิสระและรูปเหลี่ยมของแรงของแต่ละสลักซึ่งเป็นการแสดงสมการสมดุลของแรง $\sum F_X = 0$ และ $\sum F_Y = 0$ แบบกราฟิกได้แสดงไว้ในรูปที่ 3.12 ตัวเลขหมายถึงลำดับในการคิดของสลักต่างๆ มีข้อควรสังเกตไว้ในที่นี้ว่า เมื่อถึงสลัก D ซึ่งอยู่สุดท้ายแล้ว แรงปฏิกิริยา R2 จะต้องสมดุลกับแรงใน CD และ ED ซึ่งได้หามาจากสองสลักที่อยู่ข้างๆ แล้ว สิ่งนี้เป็นการตรวจความถูกต้องของการคำนวณ และมีข้อสังเกตอีกประการหนึ่งว่าในการแยกสลัก C ออกเป็นอิสระ จะเห็นได้ทันทีว่า แรงใน CE เป็นศูนย์ ตามสมการ $\sum F_Y = 0$ แต่แรงในชิ้นส่วนจะไม่เป็นศูนย์ ถ้ามีแรงภายนอกมากระทำที่ C

บ่อยครั้งที่มักจะแสดงแรงดึง T และแรงกด C ของชิ้นส่วนต่างๆ ลงในรูปเดิม โดยเขียนลูกศรชี้ออกจากสลักเมื่อเป็นแรงดึง และชี้เข้าหาสลักเมื่อเป็นแรงกด ดังแสดงไว้ในตอนล่างของรูปที่ 3.12

ในบางกรณีไม่สามารถที่จะกำหนดทิศทางที่ถูกต้องของแรงที่ไม่ทราบค่าที่กระทำต่อสลักได้ เมื่อเป็นเช่นนี้สามารถที่กำหนดเช่นไรก็ได้ ค่าลบที่ได้จากการคำนวณ หมายถึงว่า คำตอบที่ถูกต้องจะมีทิศทางกลับกับที่กำหนดไว้



รูปที่ 3.11



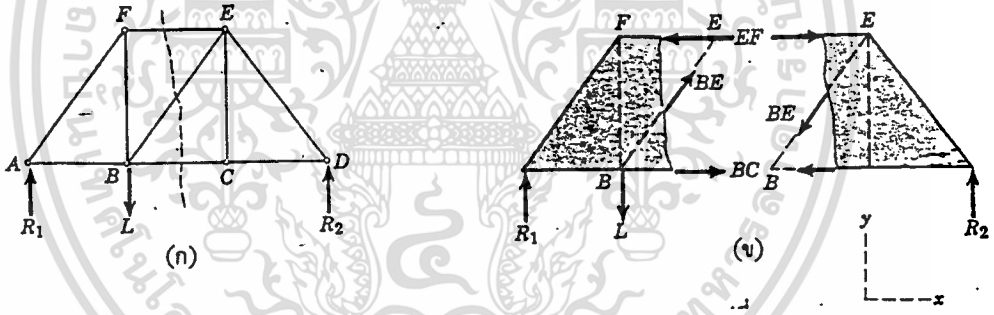
ถ้าโครงถักธรรมดา มีฐานรับมากเกินกว่าที่จำเป็นเพื่อที่จะให้อยู่ในสภาพที่สมดุล โครงถักชนิดนี้เรียกว่า Statically Indeterminate และฐานที่รับมากเกินมานี้เป็น External Redundancy ถ้าโครงถักมีชิ้นส่วนภายในมากเกินกว่าความจำเป็นที่จะป้องกันไม่ให้เกิดการยุบตัว ชิ้นส่วนที่เกินมานี้เรียกว่า Internal redundancy และโครงถักนี้เรียกว่า Statically indeterminate สำหรับโครงถักเป็น statically decterminate externally จะมีความสัมพันธ์ที่แน่นอนระหว่างจำนวนของชิ้นส่วนและจำนวนของสลักของชิ้นส่วนที่จำเป็นสำหรับความมั่นคงภายในโดยปราศจาก Redundancy เนื่องมีสมการของแรงสองสมการในการสมดุลของแต่ละสลัก ดังนั้นจะมี 2j สมการสำหรับโครงถักธรรมดาที่มี j สลัก ถ้าโครงถักมี m two-force member และมีตัวไม่ทราบค่าของแรงปฏิกิริยาที่ฐานรับมากที่สุดอยู่สามแรงจะตัวไม่ทราบค่าทั้งหมดเท่ากับ m+3 ดังนั้นโครงถักระนาบธรรมดา ซึ่งมีองค์ประกอบพื้นฐาน คือ รูปสามเหลี่ยม จะต้องมีเงื่อนไขตามสมการ m+3 = 2j สำหรับโครงถักแบบ Statically determinate internally

ความสัมพันธ์นี้เป็นเงื่อนไขที่จำเป็นสำหรับความมั่นคง แต่ว่าไม่เพียงพอที่จะกำหนดความมั่นคงได้ ทั้งนี้ ชิ้นส่วนหนึ่งหรือมากกว่าหนึ่งในจำนวนของ m ชิ้นนี้อาจจะประกอบกันเป็นรูปที่ไม่มั่นคงได้ ถ้า m+3 > 2j แสดงว่า จำนวนชิ้นส่วนมากกว่าจำนวนของสมการที่มี และโครงถักเป็น Statically Indeterminate internally ถ้าโครงถักจะไม่มั่นคงเพราะชิ้นส่วนขาดไปและยุบตัวได้เมื่อมีแรงมากระทำ

รูปเหลี่ยมของแรงสำหรับแต่ละสลักแสดงไว้ในรูปที่ 3.12 ซึ่งเป็นอีกวิธีหนึ่งที่ใช้หาแรงที่ไม่ทราบค่า และยังเป็นตัวที่ตรวจสอบคำตอบที่ได้จากการคำนวณเมื่อใช้สมการของการสมดุลด้วย ถ้ากำหนดลำดับในการรวมกันของแรงที่แต่ละสลักเหมือนกันเช่น ตามนาฬิกา รูปเหลี่ยมของแรงเหล่าสามารถนำมาเขียนทับกันได้เรียกว่า Maxwell Diagram ขนาดของแรงและทิศทางสามารถที่จะหาได้โดยตรงจาก Diagram นี้

3.3 การวิเคราะห์โดยวิธีการใช้ภาคตัด (Method of Section)

ข้อได้เปรียบของวิธีการใช้จุดต่อ(Method of joint) คือจะใช้สมการของการสมดุลเพียงสองสมการเท่านั้น ทั้งนี้เพราะวิธีการเกี่ยวข้องกับระบบของแรงที่ตัดกันที่จุดหนึ่ง ข้อได้เปรียบของวิธีการใช้ภาคตัดคือ สามารถที่จะหาแรงในชิ้นส่วนใดๆ ได้โดยตรงจากการตัด Section ผ่านชิ้นส่วนนั้น ดังนั้นจึงไม่จำเป็นที่จะต้องคำนวณจากสลักแรกไปจนถึงชิ้นส่วนที่ต้องการ ในการเลือกตัด Section มีข้อพึงสังเกตว่า ชิ้นส่วนที่ไม่ทราบค่าของแรงจะต้องมีไม่เกินสามในแต่ละภาคตัดเพราะว่ามีเงื่อนไขของสมการสมดุลเพียงสามสมการเท่านั้น



รูปที่ 3.13

จากโครงถักในรูปที่ 3.13 ขั้นแรกต้องหาแรงปฏิกิริยาภายนอกโดยวิธีการเช่นเดียวกันวิธีการใช้จุด โดยการคิดว่าโครงถักทั้งหมดเป็นวัตถุชิ้นเดียว เมื่อต้องการหาแรงในชิ้นส่วนใด เช่น BE ให้ตัดโครงถักออกโดยภาคสมมุติ ที่แสดงไว้ด้วยเส้นประดังในรูปที่ 3.13(ข) ภาคตัดนี้ชิ้นส่วนทั้งสามซึ่งยังไม่ทราบแรง เพื่อที่แต่ละส่วนของโครงถักที่ถูกตัดยังคงอยู่ในสมดุลจึงจำเป็นต้องใส่แรงเท่ากับแรงที่ชิ้นส่วนที่ถูกตัดออกไปกระทำ แรงเหล่านี้ไม่ว่าจะเป็นแรงดึงหรือแรงกดจะต้องอยู่ในแนวของชิ้นส่วนเสมอ เพราะชิ้นส่วนเหล่านี้เป็นชิ้นส่วนนี้เป็นชิ้นส่วนรับสองแรง ส่วนที่ถูกตัดทางด้านซ้ายอยู่ในสมดุลภายใต้แรงกระทำต่างๆ ดังนี้ ภาระ L แรงปฏิกิริยา R_1 และแรงที่ชิ้นส่วนทั้งสามที่ถูกตัดออกไป (ส่วนทางด้านขวา) กระทำ ทิศทางของแรงที่เขียนสามารถสังเกตได้จากเงื่อนไขของการสมดุล ดังนั้นถ้าคิดโมเมนต์รอบจุด B สำหรับส่วนทางซ้ายจะเห็นได้ว่าแรง EF ควรจะชี้ไปทางซ้ายซึ่งก็คือแรงกด ภาระ L ค่ามากกว่าแรงปฏิกิริยา R1 ดังนั้นแรง

BF จะต้องชี้ขึ้นเพื่อให้สมดุลย์ของแรงในแนวดิ่ง แรง BE เป็นแรงดึงที่กระทำออกจากพื้นที่หน้าตัดของชิ้นส่วน ด้วยการประมาณค่าของแรง R1 และ L ไว้ในใจ แรง BC ควรจะไปทางขวาเพื่อความสมดุลของโมเมนต์รอบจุด E ซึ่งถ้าสังเกตดูจากโครงถักก็จะพบเช่นเดียวกันว่า ชิ้นส่วนของแรงในทางแนวราบทางด้านล่างจะต้องถูกดึงเนื่องจากการแอ่นตัวที่เนื่องมาจากโมเมนต์คัด เมื่อใช้สมการของแรงในทางด้านแกน Y BC หาได้จากสมการของโมเมนต์รอบจุด E ด้วยวิธีการที่กล่าวมานี้แรงที่ไม่ทราบค่าทั้งสามก็สามารถหาได้โดยอิสระจากกัน

ส่วนของโครงถักทางด้านขวาอยู่ภายใต้ของแรงปฏิกิริยา R2 และแรงในชิ้นส่วนที่ถูกตัดทั้งสามซึ่งมีทิศทางกระตรงข้ามกับเมื่ออยู่ในส่วนทางด้านซ้าย ทิศทางของแรงในแนวราบดูได้จากการสมดุลของโมเมนต์รอบจุด B และ E

โครงถักทั้งสองส่วนนี้นำไปใช้ในการคำนวณหาแรงได้เหมือนกัน แต่ส่วนที่มีแรงเกี่ยวข้องน้อยกว่าจะคำนวณได้ง่ายกว่า

สิ่งที่สำคัญจะต้องเข้าใจว่าในการใช้วิธีการใช้ภาคตัดนี้คิดส่วนของโครงถักทั้งหมดเป็นวัตถุเพียงชิ้นเดียวที่อยู่ในสมดุล ดังนั้นแรงในชิ้นส่วนที่อยู่ภายในภาคตัดจะไม่นำมาพุ่งด้วยในการคำนวณ เพื่อให้วัตถุและแรงภายนอกที่กระทำเห็นได้ชัดจึงมักจะให้รอยตัดผ่านชิ้นส่วนที่ชิ้นส่วนที่ไม่ได้ผ่านสลัก

ในวิธีการใช้ภาคตัดนี้ สมการของโมเมนต์มีประโยชน์มาก และจุดศูนย์กลางของโมเมนต์นั้นมักจะเลือกจุดที่ผ่านแรงมากที่สุด ซึ่งอาจจะอยู่ในหรือนอกส่วนที่ตัดก็ได้ เมื่อเขียนผังวัตถุอิสระของภาคตัดแล้วไม่เสมอนักที่จะทราบทิศทางที่ควรจะเป็นของแรงที่ไม่ทราบค่า เมื่อเป็นเช่นนี้จะสมมุติเป็นเช่นใดก็ได้ ถ้าค่าที่ได้มีเครื่องหมายบวกแสดงว่าสมมุติได้ถูกต้อง ถ้าเครื่องหมายลบแสดงว่าสมมุติกลับทิศ การกำหนดสัญลักษณ์ของชิ้นส่วนและแรงมักจะใช้ตัวอักษรที่อยู่ที่ปลายทั้งสองของชิ้นส่วนนั้น

3.4 หลักกำลังงานสะสม (Virtual Method)

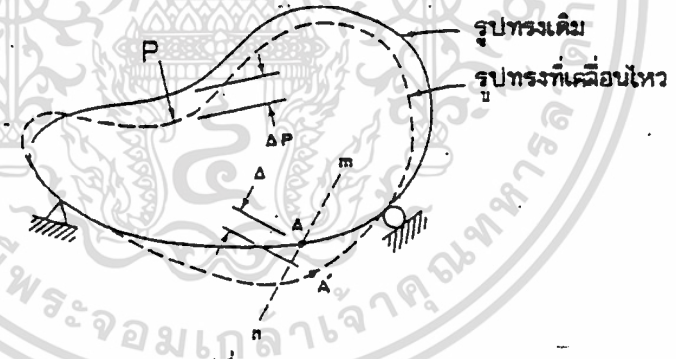
หลักกำลังงานสะสม เป็นวิธีการหาค่าการแอ่นเอนของโครงสร้างอีกวิธีหนึ่ง ซึ่ง Jean Bernoulli เป็นผู้ที่คิดขึ้นเมื่อปี ค.ศ 1717 และได้แตกแยกออกไปเป็นทฤษฎีอื่นๆ อีกมากมาย หลักการของกำลังสะสม มีดังนี้

"ชิ้นส่วนโครงสร้างใดๆ ก็ตาม รับแรงกระทำภายนอก และอยู่ใต้สภาวะสมดุลย์ เกิดการเคลื่อนที่หรือเคลื่อนไหวเพียงเล็กน้อย กำลังงานทั้งสิ้นจะมีค่าเท่ากับศูนย์"

การเคลื่อนที่หรือการเคลื่อนไหวเพียงเล็กน้อยดังกล่าว จะต้องเคลื่อนไหวเล็กน้อยจริงๆ เพื่อที่ตำแหน่งของแรงกระทำ จะไม่เปลี่ยนแปลงไปจากเดิม

แรงภายนอกหมายรวมถึง แรง(Force) และโมเมนต์(Moment) ซึ่งแบ่งออกเป็นสองชนิด คือ แรงกระทำ(Active) และแรงต้าน (Inactive) แรงกระทำดำเนินอยู่ระหว่างเกิดการเคลื่อนที่หรือเคลื่อนไหวเพียงเล็กน้อย ในขณะนั้นจะไม่มีแรงต้านเกิดขึ้น ส่วนแรงต้านได้แก่ แรงภายในวัตถุที่เกิดขึ้น เพื่อรักษาระบบให้อยู่ในสภาวะที่สมดุลย์

พิจารณาวัสดุยืดหยุ่นชิ้นหนึ่ง ภายใต้สภาวะสมดุลย์ ขณะเกิดแรง P กระทำ ซึ่งทำให้รูปร่างไหวไปเล็กน้อย ดังแสดงในรูปที่ 3.14

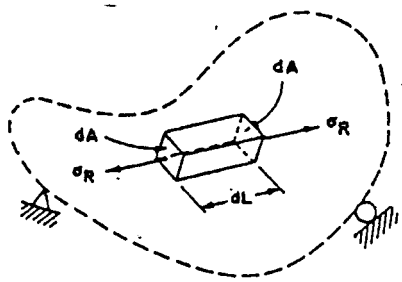


รูปที่ 3.14

จะเห็นได้ว่า ขณะที่แรง P กระทำ รูปทรงวัสดุเคลื่อนไหวไปเป็นระยะ Δp ตามแนวแรงกระทำ จะเกิดกำลังงานภายนอกเท่ากับ $\frac{1}{2}P\Delta p$ และภายในวัสดุที่ประกอบด้วยมวลต่อเนื่อง จะเกิดความเค้นและความเครียดดังแสดงในรูปที่ 3.15 ซึ่งขยายให้เห็นมวลประกอบเล็กๆ ของวัสดุที่เกิดความเค้น σr บนพื้นที่ dA

ความเครียด εr ที่เกิดขึ้นเป็นเหตุให้มวลประกอบชิ้นเล็กๆ ภายในวัสดุ เปลี่ยนแปลงความยาวเป็น $\varepsilon r dA$ ดังนั้น จะมีกำลังงานเกิดขึ้นในมวลเท่ากับ $(\frac{1}{2}\sigma_R dA)\varepsilon_R dL$ หรือเท่ากับ $(\frac{1}{2}\sigma_R dV)$ ซึ่งเมื่อคิดรวมทั้งวัสดุแล้ว จะได้กำลังงานรวมภายในทั้งหมด เท่ากับ

$$\frac{1}{2}\int_0^Y \sigma_R \varepsilon_{Rd}$$



รูปที่ 3.15

กำลังงานนี้จะถูกสะสมอยู่ในภายในวัสดุเป็นพลังงานภายใน เนื่องจากกฎเกณฑ์สมดุลของพลังงานซึ่งกล่าวว่า พลังงานสะสมภายในจะเท่ากับพลังงานที่เกิดจากแรงกระทำภายนอก ดังนั้น

$$\frac{1}{2}P\Delta_P = \frac{1}{2}\int_0^V \sigma_R \varepsilon_R dV$$

สมมุติต้องการทราบค่า Δ ซึ่งเป็นระยะที่วัสดุเคลื่อนที่จากจุด A ไปยังจุด A' ภายใต้แรงกระทำ P ก่อนที่จะใส่แรงกระทำ P ลงบนวัสดุจะสร้างเงื่อนไขบางประการ ด้วยการสมมุติให้มีหน่วยแรงเล็กๆ มากระทำชั่วคราวที่จุด A และกำหนดทิศทางของแรงดังกล่าวด้วย ให้กระทำตามทิศทางที่ต้องการทราบค่า Δ หน่วยแรงนี้จะทำให้เกิดการเคลื่อนไหวเพียงเล็กน้อยที่จุด A มีค่าเท่ากับ Δ_U และทำให้เกิดกำลังงานขึ้นเท่ากับ $\frac{1}{2}(1)(\Delta_U)$ และจะเกิดความเค้นและความเครียดเล็กน้อยเช่นกัน ดังนั้น จากหลักเกณฑ์การสมดุลย์ของกำลังงานดังที่ได้กล่าวมาแล้ว จะได้

$$\frac{1}{2}(1)(\Delta_U) = \frac{1}{2}$$

จากนั้นจึงให้แรง P กระทำที่ตำแหน่งจริงบนวัสดุ โดยยังไม่ยกเลิกหน่วยแรงเล็กๆ ที่กำลังดำเนินอยู่ก็จะได้ความสัมพันธ์ระหว่างพลังงานภายในและภายนอก เช่นเดิมคือ

$$\frac{1}{2}P\Delta_P = \frac{1}{2}\int_0^V \sigma_R \varepsilon_R dV$$

พิจารณาจากผลของการกระทำของแรงภายนอกทั้งสองกรณี จะเห็นว่าหากค่อยๆ เพิ่มหน่วยแรงเล็กๆ ขึ้นที่ละน้อย จนเท่ากับ P แสดงว่าหน่วยแรงที่เพิ่มขึ้นก็จะค่อยๆ เปลี่ยนแปลงรูปทรงการเคลื่อนไหวจาก Δ_U กระทั่งเป็น Δ จริงๆ และ Δ_U ที่เกิดขึ้นภายในมวลประกอบเล็ก ๆ ก็จะเพิ่มขึ้นที่ละเล็กละน้อย จนได้ความยาวที่เปลี่ยนแปลงเป็น $\varepsilon_{rd}L$ จะเห็นได้ว่ากำลังงานภายนอกจะต้องเท่ากับพลังงานที่สะสมอยู่ภายในตลอดเวลาที่แรงกระทำเพิ่มขึ้น ดังนั้นจากกฎการสมดุลย์ของกำลังงาน จะได้

$$(1)\Delta = \int_0^Y \sigma_{vd}\ \varepsilon_{Rd} \qquad (ก)$$

$$(1)\Delta = \int_0^Y \sigma_v \varepsilon_{Rd} \qquad (ข)$$

สมการทั้งสอง เป็นสมการที่แสดงถึงหลักการของกำลังงานสะสม และเนื่องจากมีหน่วยแรงเข้ามาเกี่ยวข้อง จึงมีชื่อเรียกวิธีนี้อีกชื่อหนึ่งคือ วิธีหน่วยแรงกระทำ (Unit_load Method) ซึ่งจะใช้วิเคราะห์การแอ่นเอนของชิ้นส่วนโครงสร้างได้ทั้งในกรณีรับแรงกระทำตามแกน โมเมนต์แรงเฉือน และโมเมนต์บิด

การวิเคราะห์หาการโก่งตัวของโครงข้อหมุน (Truss) โดยวิธีกำลังงานสะสม

หลักวิธีกำลังงานสะสมสามารถนำมาวิเคราะห์หาระยะการแอ่นและมุมลาดของโครงข้อหมุน สำหรับการหาค่าระยะแอ่นของโครงข้อหมุน(Truss) ความหมายของ σ_u และ ε_r ในสมการ 3.7.1 เนื่องจากแรงที่เกิดขึ้นภายในชิ้นส่วนของโครงข้อหมุน เป็นแรงตามแนวแกน ดังนั้นความหมายของ σ_u และ ε_r ก็คือ

$$\sigma_u = \frac{U}{A}$$

ซึ่ง

U = แรงภายในชิ้นส่วน ที่เกิดจากแรง(1) หน่วยกระทำตรงตำแหน่งและทิศทางที่ต้องการทราบระยะแอ่น

A = พื้นที่หน้าตัดขวางของชิ้นส่วน

เช่นเดียวกัน

$$\varepsilon_r = \frac{N}{A}$$

โดยที่

N = แรงภายในชิ้นส่วน ที่เกิดจากแรงกระทำจริงๆ

E = โมดูลัสยืดหยุ่นของชิ้นส่วน

นำไปแทนในสมการ (ก) จะได้

$$\Delta = \bar{Z}\frac{U}{A}\left(\frac{N}{A}\right)AL = \bar{Z}\frac{u}{A}$$

โดยเปลี่ยนจาก dA มาเป็น A และใช้แทนค่า L แทน dL (ไม่มีความจำเป็นที่จะต้องใช้หลักการอินทิเกรต เนื่องจากความเค้นกระจายสม่ำเสมอ ตลอดหน้าตัดขวาง และคงที่ตลอดความยาวของชิ้นส่วน)

บทที่ 4

การออกแบบโปรแกรม

4.1 ข้อมูลที่ต้องการและผลลัพธ์ที่ได้จากโปรแกรม

ข้อมูลที่ต้องการ

ข้อมูลที่ต้องป้อนเข้าเครื่องคอมพิวเตอร์ แบ่งออกเป็น 5 กลุ่ม คือ

1. ข้อมูลของจุดต่อ (Node Data) ได้แก่ จำนวนจุดต่อ , หมายเลขจุดต่อ และค่าโคออร์ดิเนตของจุดต่อทุกจุด

2. ข้อมูลของชิ้นส่วน (Element Data) ได้แก่ หมายเลขชิ้นส่วนของจุดต่อ , หมายเลขของจุดต่อที่ปลายของชิ้นส่วน

3. ข้อมูลของจุดรองรับ (Boundary Data) ได้แก่ หมายเลขของจุดรองรับ , สภาพการยึดรั้งที่จุดรองรับนั้นๆ

4. ข้อมูลของวัสดุ (Meterial Data) ได้แก่ พื้นที่หน้าตัด และโมดูลัสยืดหยุ่น(E)

5. ข้อมูลของน้ำหนักบรรทุก (Load Data) ได้แก่ จุดต่อที่มีแรงกระทำ และขนาดของแรง

คำตอบที่ได้จากการคำนวณของเครื่องคอมพิวเตอร์

คำตอบที่ได้จากเครื่องคอมพิวเตอร์ แบ่งออกเป็น 4 ชนิด คือ

1. ค่าแรงปฏิกิริยาที่จุดรองรับ

X-Reaction = แรงปฏิกิริยาในแนวแกน X

Y-Reaction = แรงปฏิกิริยาในแนวแกน Y

2. แรงภายในชิ้นส่วน

Member Force = แรงที่เกิดภายในชิ้นส่วน

Member Stress = ความเค้นภายในชิ้นส่วน

เครื่องหมาย + แสดงว่าชิ้นส่วนนั้นรับแรงดึง

เครื่องหมาย - แสดงว่าชิ้นส่วนนั้นรับแรงอัด

3. ค่าการโก่งตัวของจุดต่อ

X-Disp. = ค่าการโก่งตัวของจุดต่อในแนวแกน X

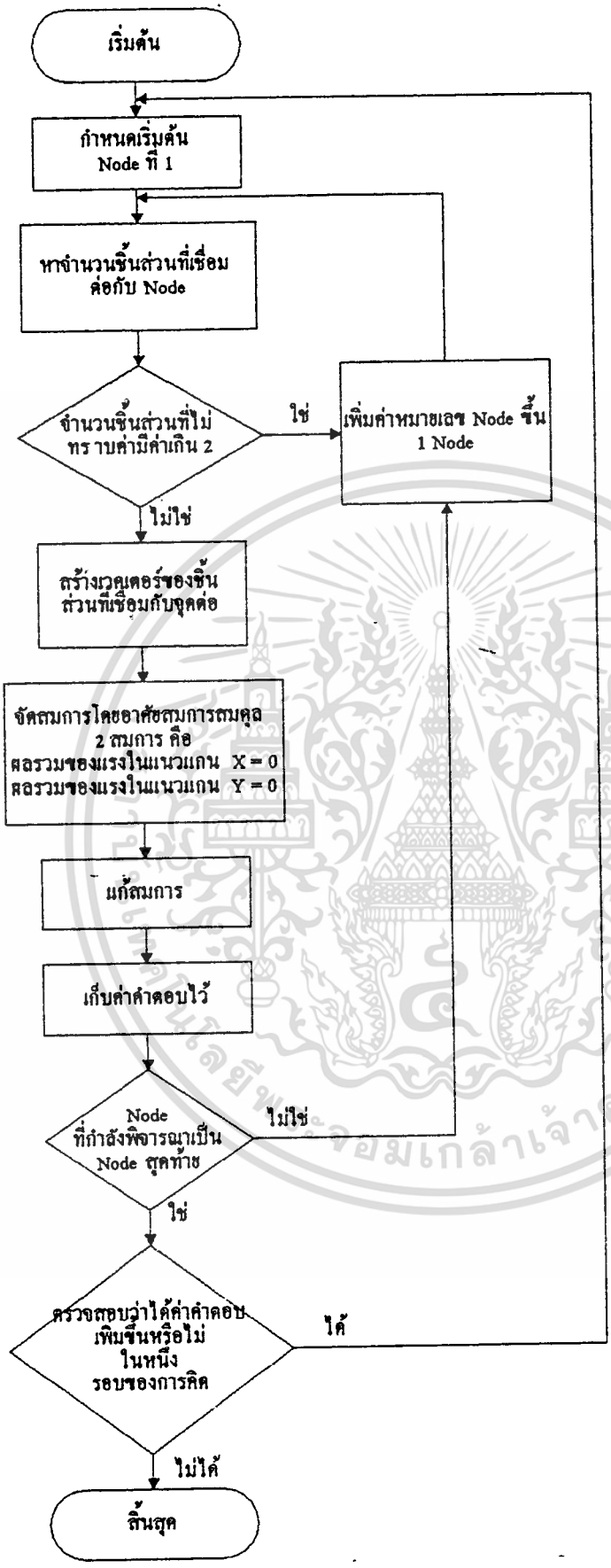
Y-Disp. = ค่าการโก่งตัวของจุดต่อในแนวแกน Y

4. ปริมาตรของวัสดุที่ใช้ทั้งโครงสร้าง

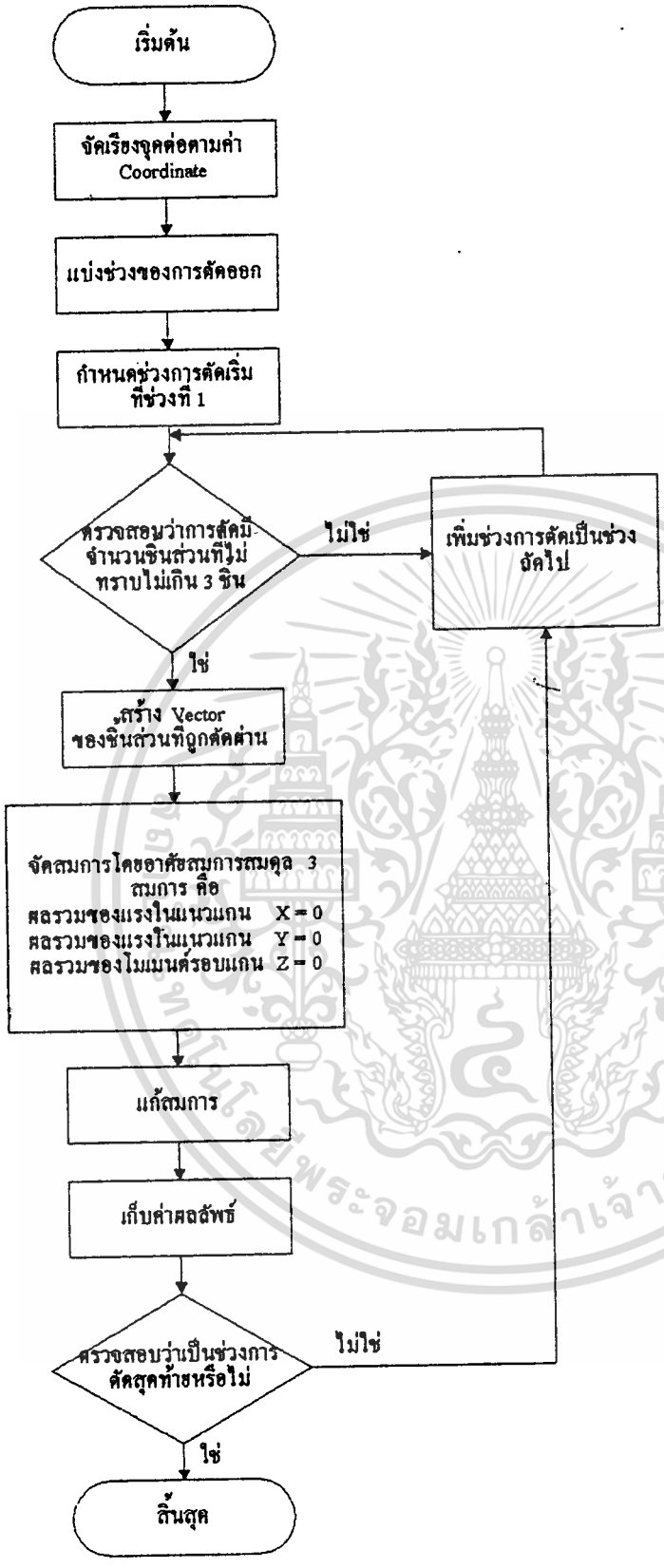
4.2 ลำดับขั้นตอนในการวิเคราะห์ แบ่งออกเป็น 3 ขั้นตอน ดังนี้

- ลำดับขั้นตอนการวิเคราะห์ Vector with Method of Joint
- ลำดับขั้นตอนการวิเคราะห์ Vector with Method of Section
- ลำดับขั้นตอนการวิเคราะห์ Unit Load Method

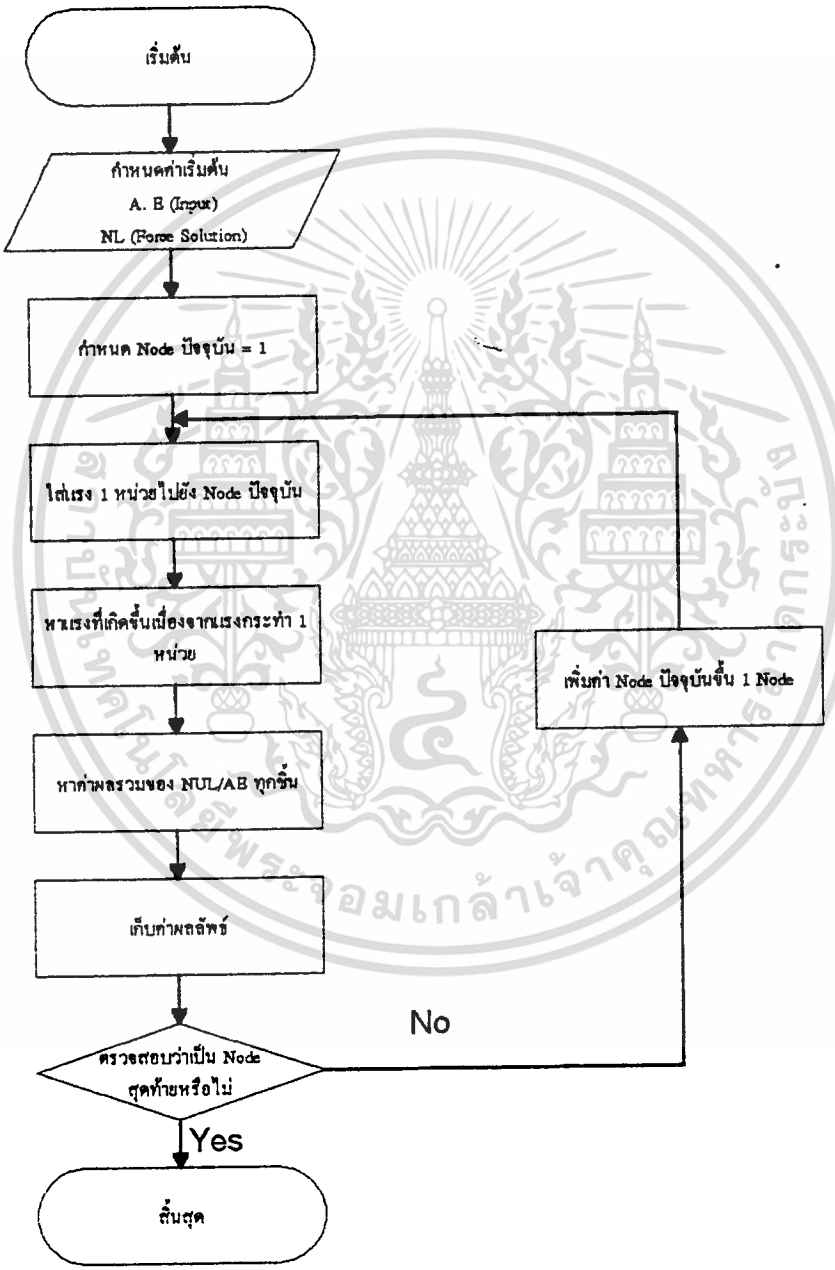
ลำดับขั้นตอนการวิเคราะห์ Vector with Method of Joint



ลำดับขั้นตอนการวิเคราะห์ Vector with Method of Section



ลำดับขั้นตอนการวิเคราะห์ Unit Load Method



บทที่ 5

การใช้งาน PC_TRUSS V 1.0

การใช้งานโปรแกรม PC_TRUSS ในการหาแรงในชิ้นส่วนของโครงข้อหมุนสองมิติแบบ Determinate สำหรับผู้ที่ใช้งานสามารถใช้งานครั้งแรกได้โดยง่าย เพราะผู้เขียนเขียนโปรแกรม ให้มีความสะดวกและรวดเร็วในการใช้งาน เช่น Pulldown Menu , Editor ,Graphics และมี Help ช่วยประกอบในระหว่างการใช้งาน

5.1 รายละเอียดของโปรแกรม

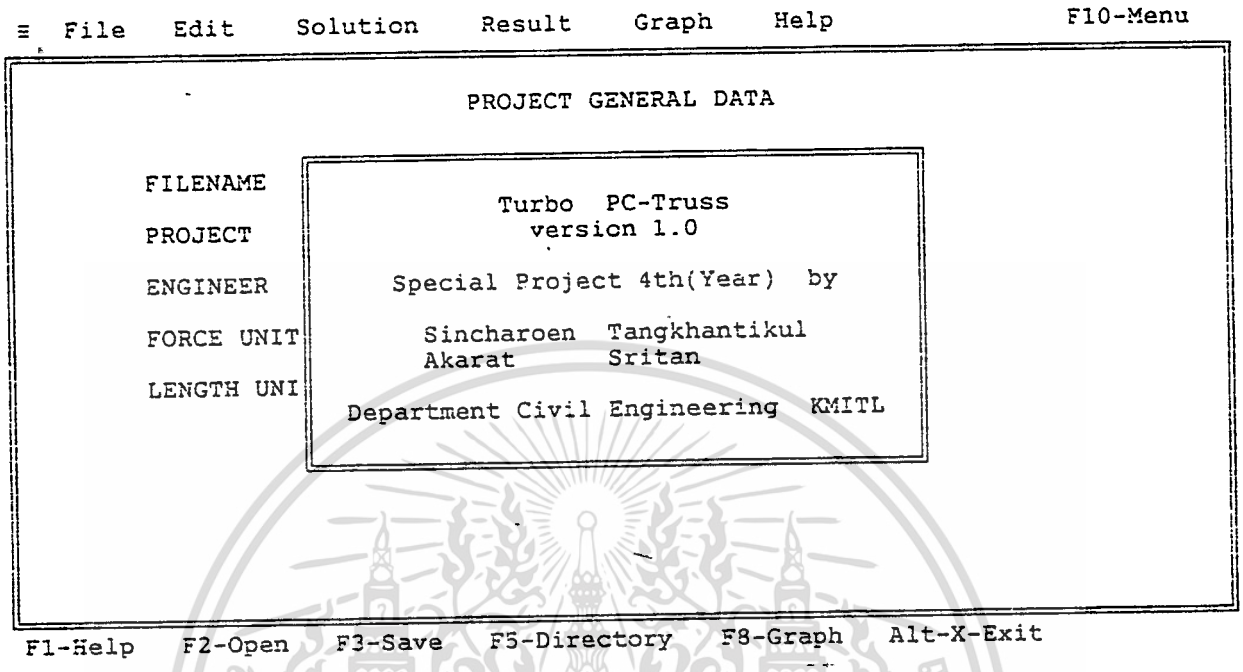
Language Compiler	: Turbo Pascal Version 7.0
Operating System	: Dos Version 2.0 ขึ้นไป
Hardware	: IBM-PC XT , AT or Compatible Ram size 640 kB or - Only one diskette or more - Monochrome Display Adepter With Hercules Graphics Card or CGA or EGA or VGA or Super VGA
Result	: Display on Screen & Hard Copy
Diskette	: Only one diskete , with dos v6.20 , main.exe , file and three Example File

5.2 การเริ่มใช้งานโปรแกรม PC_TRUSS (PT.BAT File)

โปรแกรม PC_TRUSS เป็นโปรแกรมที่สามารถใช้งานได้กับเครื่องไมโครคอมพิวเตอร์ IBM-PC บน Operating System หรือ PC-Dos v2.0 ขึ้นไป โดยที่ตัวเครื่องคอมพิวเตอร์มีขนาดหน่วยความจำอย่างน้อยที่สุด 640 kB

โปรแกรม PC_TRUSS เป็นโปรแกรมที่มีไฟล์เริ่มต้น คือ ไฟล์ PT.BAT ซึ่งเป็น BATFILE ที่ประกอบด้วย การเรียกใช้ไฟล์ MAIN.EXE และ GRAPHICS.COM ในการใช้งานของโปรแกรม ดังนั้น สามารถสั่งการทำงานบน Dos Prompt โดยการพิมพ์ PT.BAT แล้วกด Enter

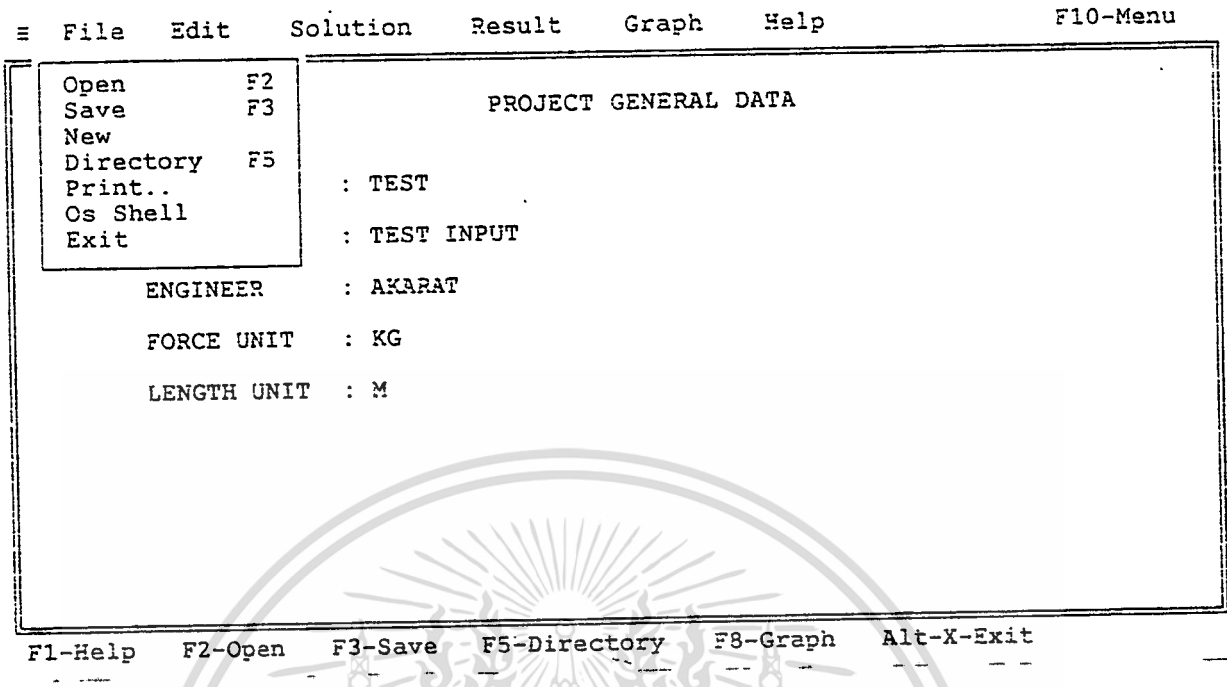
A>**PT.BAT** กด Enter จะปรากฏภาพดังรูปที่ 1 เป็นส่วนที่แสดงรายละเอียดเกี่ยวกับ Program



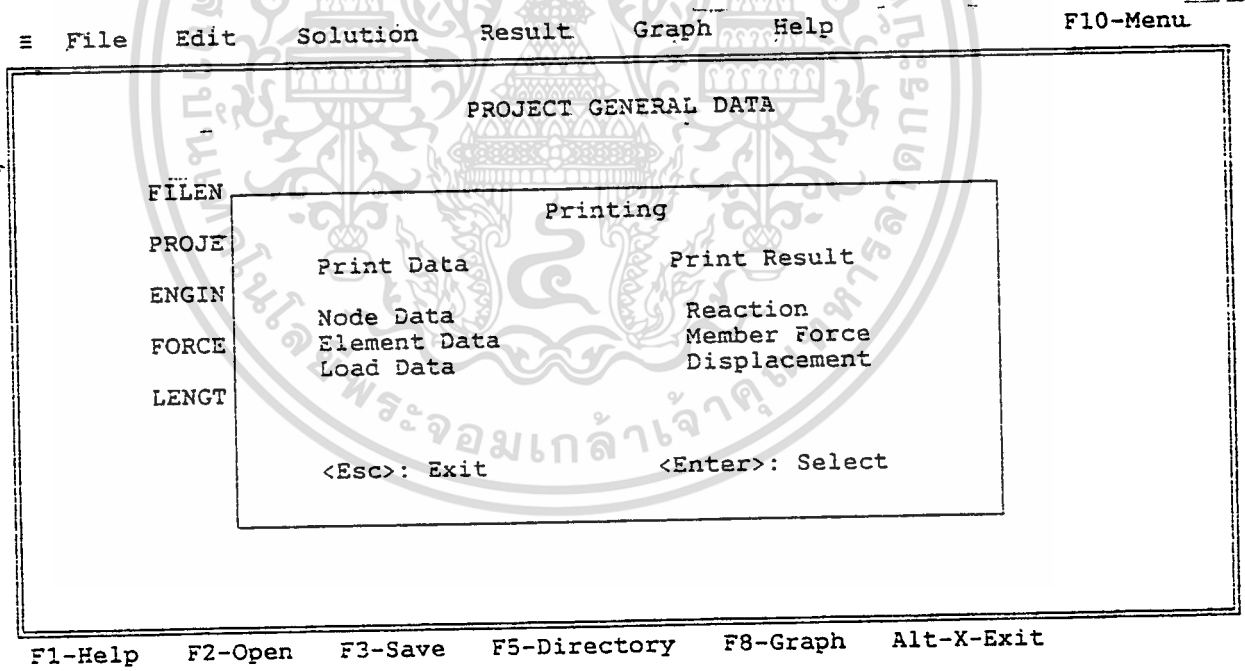
รูปที่ 1

1. เมนู File เป็นส่วนที่ใช้รวบรวมการใช้งานที่เกี่ยวกับการจัดการไฟล์ข้อมูลดังรูปที่ 2.1

- Open หมายถึง การเปิดไฟล์ข้อมูลเก่า เพื่อนำมาใช้งานหรือเพื่อทำการแก้ไข
- Save หมายถึง การเก็บไฟล์ข้อมูล หรือการตั้งชื่อไฟล์ข้อมูลใหม่ โดยปกติเมื่อทำการป้อนข้อมูลและเมื่อออกจากการใช้งานโปรแกรมจะทำการเก็บข้อมูลโดยอัตโนมัติ
- New หมายถึง การเริ่มต้นในการป้อนข้อมูลใหม่
- Directory หมายถึง การกำหนดช่องเก็บข้อมูล
- Print หมายถึง การพิมพ์ข้อมูล ดังรูปที่ 2.2
- Os Shell หมายถึง การออกสู่ Dos ชั่วคราว
- Exit หมายถึง จบการทำงานในการใช้โปรแกรมและออกจากโปรแกรม



รูปที่ 2.1

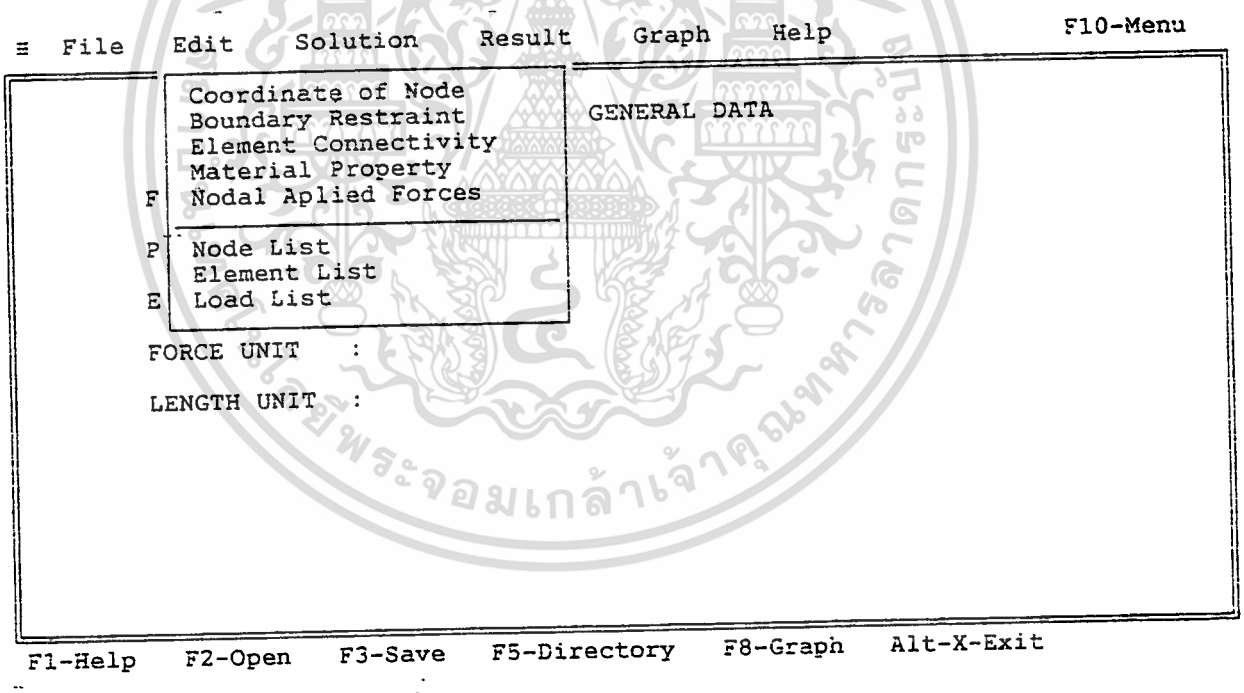


รูปที่ 2.2

2. Edit การแก้ไขข้อมูลหรือการป้อนข้อมูล ดังรูปที่ 3.1 และ 3.2

- Coordinate of Node หมายถึง คำสั่งเลือกการป้อนข้อมูลจุดต่อของโครงสร้าง
- Boundary Restraint หมายถึง คำสั่งเลือกการป้อนจุดรองรับของโครงสร้าง
- Element Connectivity หมายถึง คำสั่งเลือกการป้อนชิ้นส่วนของโครงสร้าง

- Material Property หมายถึง คำสั่งเลือกการป้อนคุณสมบัติของวัสดุที่ใช้ทำเป็นชิ้นส่วนของโครงสร้าง
- Nodel Aplie Force หมายถึง คำสั่งเลือกการป้อนแรงหรือน้ำหนักที่กระทำต่อจุดต่อของโครงสร้าง
- Node List หมายถึง คำสั่งเลือกให้แสดงผลของการป้อนจุดต่อของโครงสร้าง ดังรูปที่ 3.3
- Element List หมายถึง คำสั่งเลือกให้แสดงผลของชิ้นส่วนของโครงสร้าง
- Load List หมายถึง คำสั่งเลือกให้แสดงผลของแรงหรือน้ำหนักที่กระทำต่อจุดต่อของโครงสร้าง



รูปที่ 3.1

≡ File Edit Solution Result Graph Help F10-Menu

FORCE RESULTED <2D-TRUSS SYSTEM> <ESC> : EXIT

ELEM	LENGTH	FORCE	STRESS
1	3.00	250.00	
2	3.00	250.00	
3	3.00	400.00	
4	3.00	400.00	
5	3.00	250.00	
6	3.00	250.00	
7	3.00	-250.00	
8	2.66	-400.00	
9	2.66	-400.00	
10	2.66	-250.00	
11	2.66	150.00	
12	2.66	-25.00	
13	2.66	-50.00	
14	2.66	-25.00	

F1-Help F2-Open F3-Save F5-Directory F8-Graph Alt-X-Exit

รูปที่ 3.2

<< Connectivity Data >>

ELEMENT CONECTIVITY <2D-TRUSS SYSTEM>

ELEM	LENGTH	1-NODE	2-NODE
1	2.00	1	2
2	2.00	2	3
3	2.00	3	4
4	2.00	4	5
5	2.00	6	7
6	2.00	7	8
7	2.00	2	6
8	2.00	3	7
9	2.00	4	8
10	2.33	1	6

Edit Window Keyboard

Node Description
< 0> : Terminate Input
<-1> : Change Total Node
<-2> : List of Node

List Window Keyboard

<ESC> Return to Edit Node
<PgUp> Previous Page
<PgDn> Next Page

No. of total element : [13]

ELEM 1-NODE 2-NODE NODAL GEN

PREVIOUS ENTRY :
CURRENT ENTRY : -2

F1-Help F2-Open F3-Save F5-Directory F8-Graph Alt-X-Exit

รูปที่ 3.3

3. Coordinate of Node การป้อนข้อมูลจุดต่อของโครงสร้าง ดังรูปที่ 4

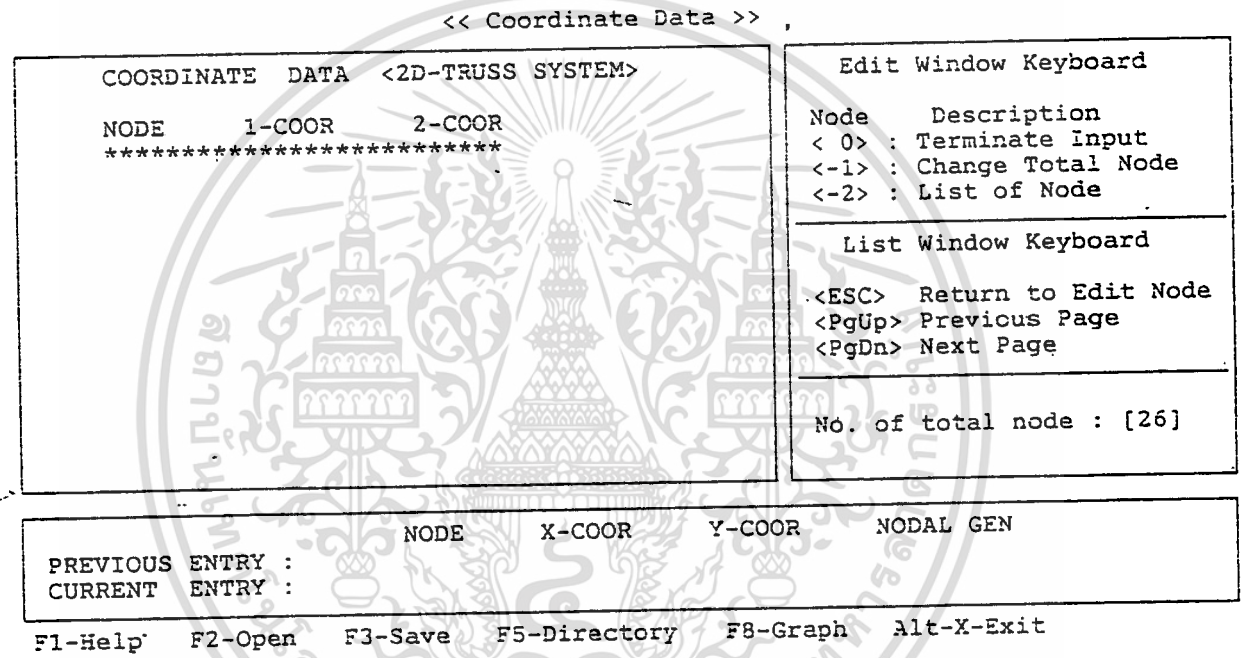
- Node คือ การป้อนจำนวณจุดต่อทั้งหมดของโครงข้อหมุน โดยการใส่ข้อมูลที่เป็นตัวเลข ซึ่งเป็นจำนวณเต็มให้เท่ากับจุดต่อจริงของโครงสร้าง ถ้าใส่มากกว่าหรือน้อยกว่า จะมีผลต่อการวิเคราะห์และการแสดงผลของโครงสร้าง

- X-Coor คือ การป้อนจุด โคออร์ดิเนตของแต่ละจุดต่อ(Node) ในแนวแกน X โดยจะถือตามระบบของ Cate Suin Coordinate โดยจะทำการป้อนใน Quadrant ที่ 1 เพื่อความสะดวก

- Y-Coor คือ การป้อนจุดในแนวแกน Y

ในระหว่างการป้อนจุดต่อของโครงสร้างผู้ใช้สามารถที่ให้โปรแกรมแสดงค่าของจุดต่อที่ได้ทำการป้อนข้อมูล ดังแสดงในรูปที่ 4 ในหัวข้อ Edit Windows keybord และ List Windows keybord

Node Gen คือ การแบ่งข้อมูลของจุดต่อออกเป็นช่วงที่เท่ากัน เพื่อความสะดวกในการป้อนข้อมูล โดยโปรแกรมจะทำการป้อนให้โดยอัตโนมัติ



รูปที่ 4

4. Boundary Restraint การป้อนลักษณะของจุดรองรับ ดังรูปที่ 5

การเขียนจุดรองรับของโครงสร้าง

- First Support <H> ในการป้อนในจุดแรกนี้ควรเป็นการป้อนจุดรองรับที่เป็น Hinge Support คือ ไม่สามารถเคลื่อนที่ได้ทั้งในแนวแกน X และแนวแกน Y

- Second Support <R> จะเป็นการป้อนจุดรองรับที่เป็นลักษณะของ Roller Support คือ จะต้องทำการเลือกเอาว่า จะยอมให้เกิดการเคลื่อนที่ของจุดรองรับ ในแนวแกน X หรือในแนวแกน Y สมมุติในกรณีที่ผู้ใช้ทำการเลือกให้เกิดการเคลื่อนที่ในแนวแกน X โปรแกรมจะไม่ยอมให้เคลื่อนที่ในแนวแกน Y โดยอัตโนมัติ

<< Boundary Condition Data >>

```
BOUNDARY INFORMATION  <2D-TRUSS SYSTEM>

THIS PROGRAM DEFINE DETERMINATE TRUSS
        BY USE 2 SUPPORT

      1)  HINGED SUPPORT <H>
      2)  ROOLER SUPPORT <R>


 HINGED SUPPORT
   DEFINE : LOCK BOTH X-AXIS AND Y-AXIS

 ROOLER SUPPORT
   DEFINE : LOCK ONE AXIS <X-AXIS OR Y-AXIS>
```

```
Edit Window Keyboard

< 0> : Terminate Input

Type of Support

Type    1-BOUN    2-BOUN

<H>       L         L
<R>      F,L       F,L
```

```
                    TYPE      NODE      X-BOUN      Y-BOUN
FIRST  SUPPORT :    <H> :
SECOND SUPPORT :    <R> :
```

```
F1-Help   F2-Open   F3-Save   F5-Directory   F8-Graph   Alt-X-Exit
```

รูปที่ 5

5. Element Connectivity เป็นการป้อนชิ้นส่วนของโครงสร้างที่ทำการเชื่อมระหว่างจุดต่อ (Node) ของโครงสร้าง โดยที่แต่ละชิ้นส่วนจะมีจุดเชื่อมสองจุด ลักษณะการป้อนข้อมูลดังรูปที่ 6

Node 1 คือ การป้อนจุดต่อจุดแรกของชิ้นส่วน หรือจุดเริ่มต้นของการเชื่อมของชิ้นส่วน

Node 2 คือ การป้อนจุดต่อจุดที่สองของชิ้นส่วน หรือจุดสิ้นสุดของการเชื่อมของชิ้นส่วน

Node Gen มีลักษณะการป้อนคล้ายกับการป้อนในเรื่องของ การป้อน Node Gen ของจุดต่อ (Node)

<< Connectivity Data >>

```
ELEMENT CONECTIVITY  <2D-TRUSS SYSTEM>

ELEM      LENGTH      1-NODE      2-NODE
-------------------------------------
   1        2.00          1           2
   2        2.00          2           3
   3        2.00          3           4
   4        2.00          4           5
   5        2.00          6           7
   6        2.00          7           8
   7        2.00          2           6
   8        2.00          3           7
   9        2.00          4           8
  10        2.83          1           6
```

```
Edit Window Keyboard

Node     Description
< 0> : Terminate Input
<-1> : Change Total Node
<-2> : List of Node

List Window Keyboard

<ESC>  Return to Edit Node
<PgUp> Previous Page
<PgDn> Next Page

No. of total element : [13]
```

```
                    ELEM      1-NODE      2-NODE      NODAL GEN
PREVIOUS ENTRY :
CURRENT  ENTRY :      -2
```

```
F1-Help   F2-Open   F3-Save   F5-Directory   F8-Graph   Alt-X-Exit
```

รูปที่ 6

6. Material Property คือการป้อนคุณสมบัติของวัสดุที่นำมาใช้ในการทำโครงสร้างโดยจะต้องทำการป้อนในคุณสมบัติดังต่อไปนี้

- Area คือ การป้อนพื้นที่หน้าตัดของชิ้นส่วนของโครงสร้าง
- Modulas (E) คือ การป้อนค่าความยืดหยุ่นของวัสดุ ที่ใช้ในการทำโครงสร้าง

ในการเลือกชิ้นส่วนนั้นจะต้องทำการเลือกชิ้นส่วนเป็นแบบเดียวกันหมด A = All (การกำหนดให้ชิ้นส่วนทุกกตัวมีคุณสมบัติแบบเดียวกันหมด)

7. Nodal Aplied Force เป็นคำสั่งในการป้อนแรงหรือน้ำหนักที่โครงสร้างต้องรับ โดยจะต้องใส่แรงหรือน้ำหนักที่จุดต่อของโครงสร้างเท่านั้น โดยจะต้องทำการป้อนข้อมูลดังนี้

- Node เป็นการป้อนจุดต่อที่แรงหรือน้ำหนักกระทำต่อจุดต่อของโครงสร้าง
- X คือแรงหรือน้ำหนักที่กระทำในแนวแกน X
- Y คือแรงหรือน้ำหนักที่กระทำในแนวแกน Y
- Node Gen มีลักษณะการป้อนที่คล้ายกับการป้อนจุดต่อของโครงสร้าง

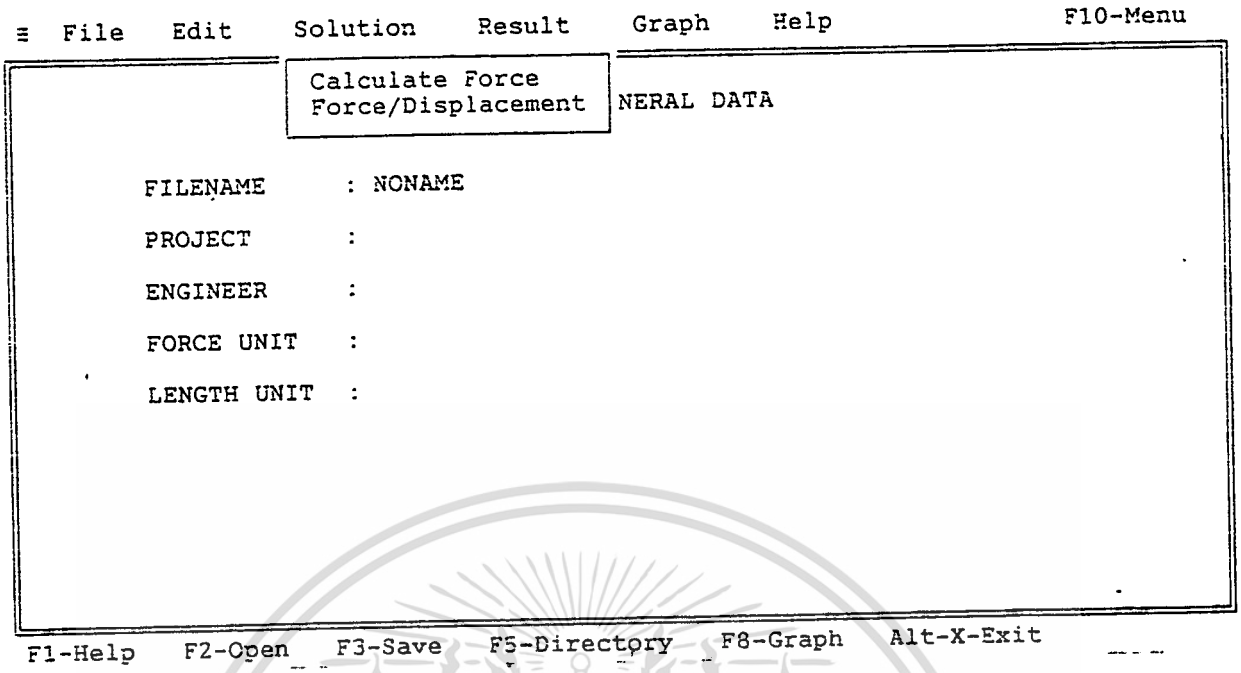
8. Node List การแสดงผลของข้อมูลทั้งหมดของจุดต่อ(Node)ที่ได้ทำการป้อนข้อมูล

9. Element List การแสดงผลของข้อมูลทั้งหมดของชิ้นส่วน(Element)ที่ได้ทำการป้อนข้อมูล

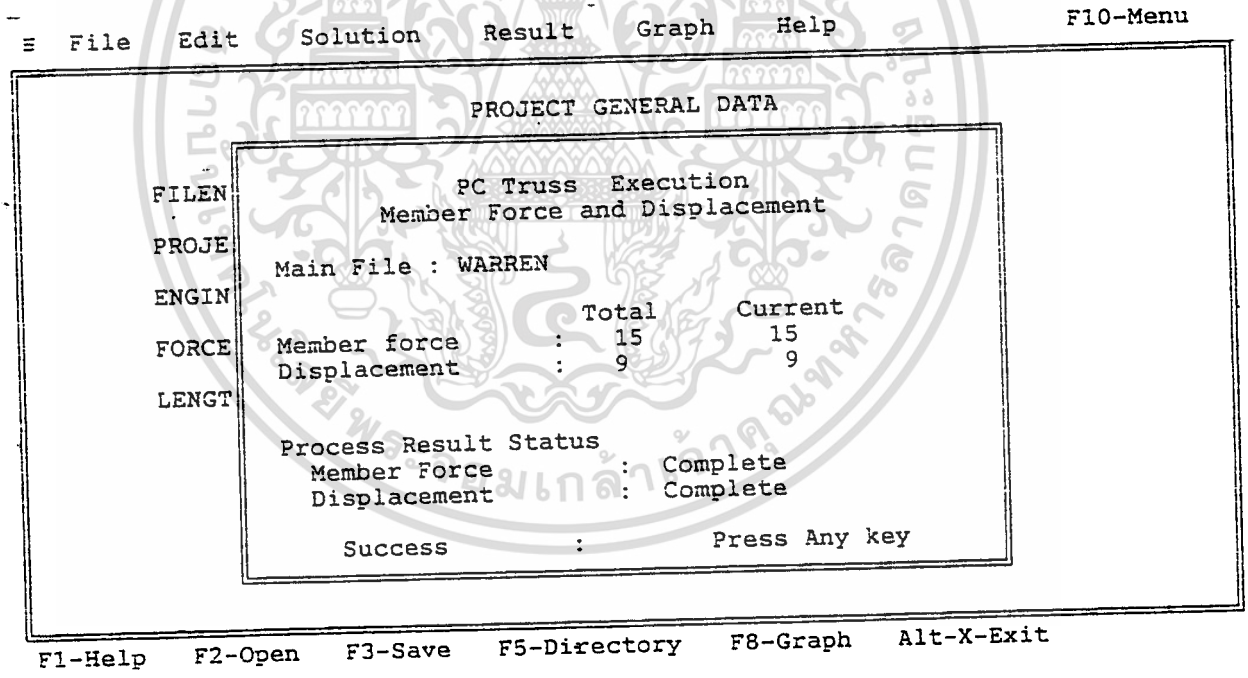
10. Load List การแสดงผลของข้อมูลทั้งหมดของแรงหรือน้ำหนัก(Load)ที่ได้ทำการป้อนข้อมูล

11. Solution หมายถึง คำสั่งที่ใช้ในการเลือกให้โปรแกรมทำการวิเคราะห์ โครงข้อหมุนที่จะทำการออกแบบหรือที่ป้อนข้อมูลในคำสั่ง Edit เรียบร้อยแล้วจะแสดง ดังรูปที่ 7.1 และ 7.2 รายละเอียดในการเลือกรูปแบบการคำนวณดังนี้

- Calculate Force เป็นการเลือกให้โปรแกรมทำการคำนวณเฉพาะแรงในชิ้นส่วนของโครงสร้างเท่านั้น
- Force / Displacment เป็นการเลือกให้โปรแกรมทำการคำนวณทั้งแรง และค่าการโก่งหรือการทรุดตัวของชิ้นส่วนในโครงสร้าง



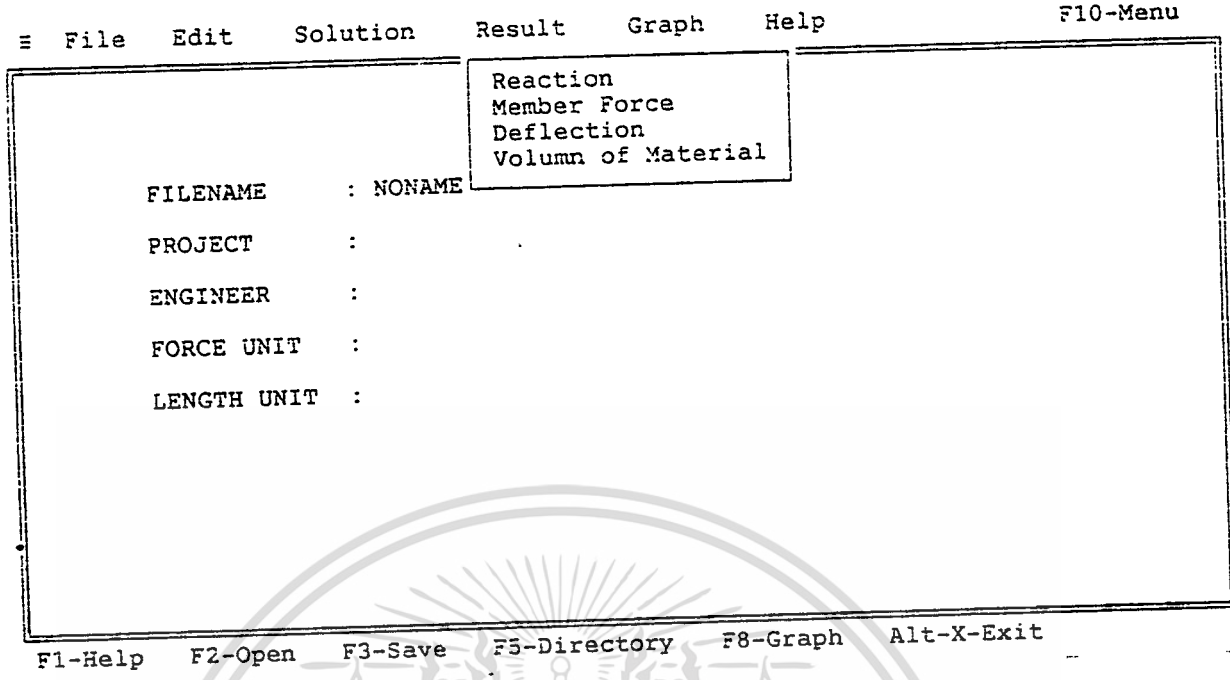
รูปที่ 7.1



รูปที่ 7.2

12. Result เป็นการเลือกคำสั่งในการแสดงผลที่ได้จากการคำนวณ ซึ่งประกอบด้วยส่วนต่างๆ ดังรูปที่ 8 ดังนี้

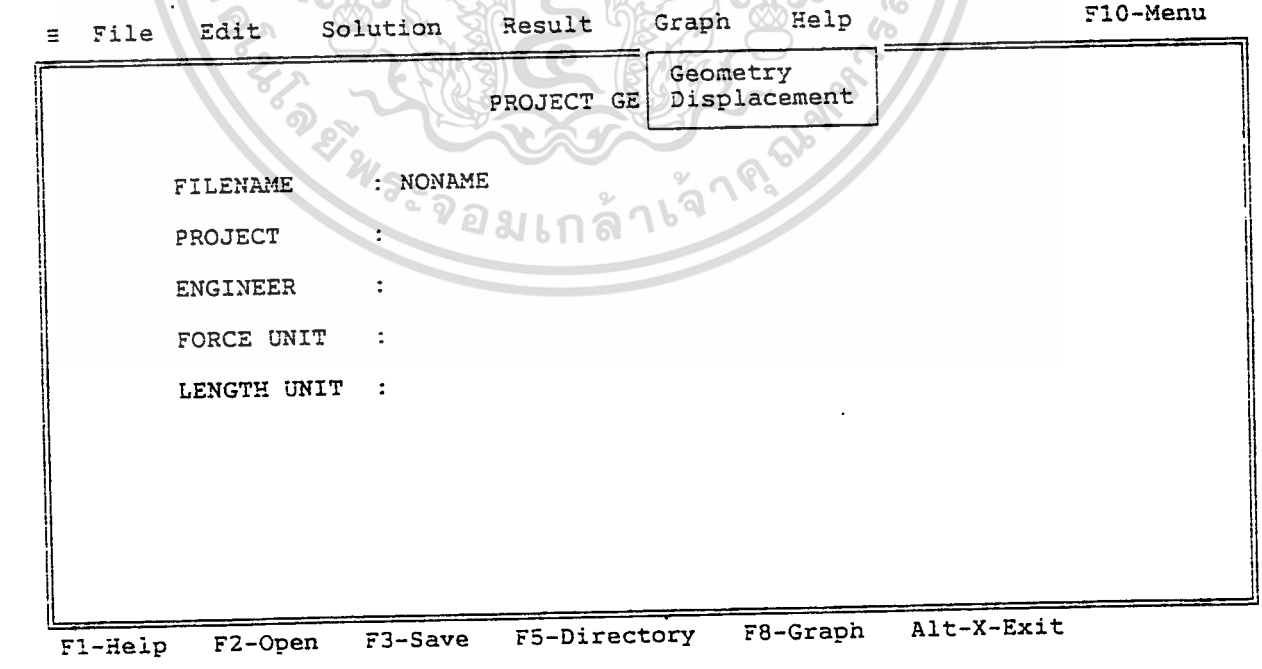
- Reaction คือการแสดงผลของแรงที่จุดรองรับ(Support) ของโครงสร้าง
- Member คือการแสดงผลของแรงที่เกิดขึ้นในแต่ละชิ้นส่วน
- Defection คือการแสดงผลการเคลื่อนที่ของจุดต่อของโครงสร้าง
- Volum of Material คือการแสดงผลรวมของปริมาณวัสดุที่ใช้ในการทำโครงสร้าง



รูปที่ 8

13. Graph คือการแสดงภาพของโครงข้อหมุน ดังรูปที่ 9

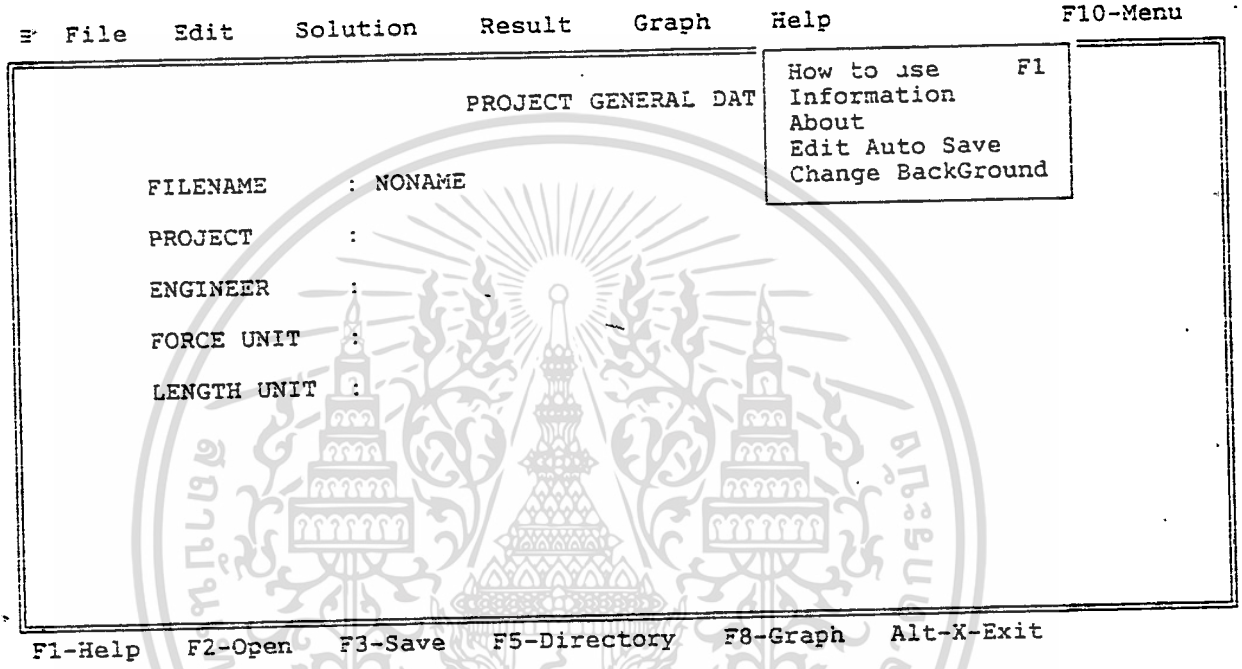
- Geometry เป็นการแสดงภาพของโครงข้อหมุน
- Displacement เป็นการแสดงภาพการทรุดตัวของโครงข้อหมุนหลังจากที่ผ่านการคำนวณแล้ว



รูปที่ 9

14. Help การแสดงรายละเอียดของ Program ดังรูปที่ 10

- How to Use เป็นการแสดงรายละเอียดโปรแกรม
- Information เป็นการแสดงรายละเอียดของข้อแนะนำในการใช้โปรแกรม
- About เกี่ยวกับตัวผู้เขียน Program



รูปที่ 10

การใช้คีย์พิเศษในโปรแกรม

F1 = Help

F2 = Open

F3 = Save

F5 = Directory

F8 = Graph

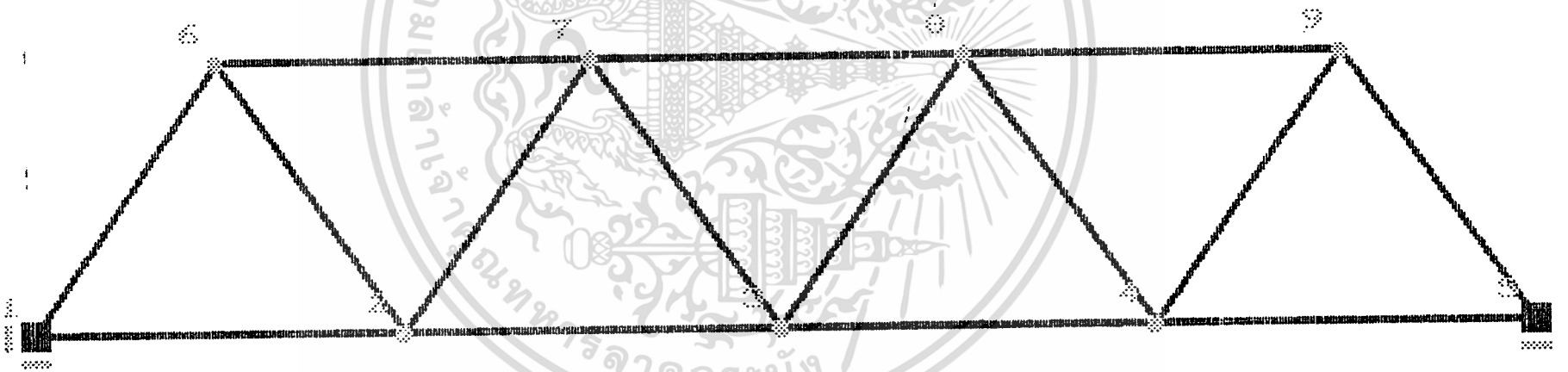
F10 = Menu

Alt-X = Exit

5.3 ตัวอย่างการใช้โปรแกรมในการคำนวณวิเคราะห์โครงสร้าง

- ตัวอย่างที่ 1 Warren Truss (Warren.dat)
- ตัวอย่างที่ 2 K_Truss (K_truss.dat)
- ตัวอย่างที่ 3 Demo.dat

GEOMETRY OF STRUCTURE



Filename : WARREN.dat

==========

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : WARREN
ENGINEER : Akarat

==========

COORDINATE DATA <2D-TRUSS SYSTEM>

NODE	X-COOR	Y-COOR
1	0.000E+00	0.000E+00
2	3.000E+00	0.000E+00
3	6.000E+00	0.000E+00
4	9.000E+00	0.000E+00
5	1.200E+01	0.000E+00
6	1.500E+00	2.200E+00
7	4.500E+00	2.200E+00
8	7.500E+00	2.200E+00
9	1.050E+01	2.200E+00

==========

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : WARREN
ENGINEER : Akarat

==========

ELEMENT CONNECTIVITY DATA <2D-TRUSS SYSTEM>

MEMBER	1-NODE	2-NODE
1	1	2
2	2	3
3	3	4
4	4	5
5	6	7
6	7	8
7	8	9
8	1	6
9	2	7
10	3	8
11	4	9
12	2	6
13	3	7
14	4	8
15	5	9

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : WARREN
ENGINEER : Akarat

LOAD DATA <2D-TRUSS SYSTEM>

NODE	X-COOR	Y-COOR
6	0.000E+00	-5.000E+02
7	0.000E+00	-5.000E+02
8	0.000E+00	-5.000E+02
9	0.000E+00	-5.000E+02

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : WARREN
ENGINEER : Akarat

REACTION RESULTED <2D-TRUSS SYSTEM>

Reaction of Node 1
X-AXIS = 0.00
Y-AXIS = 1000.00

Reaction of Node 5
X-AXIS = 0.00
Y-AXIS = 1000.00

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : WARREN
ENGINEER : Akarat

FORCE RESULTED <2D-TRUSS SYSTEM>

MEMBER	LENGTH	FORCE	STRESS
1	3.00	6.818E+02	1.136E+06
2	3.00	1.364E+03	2.273E+06
3	3.00	1.364E+03	2.273E+06
4	3.00	6.818E+02	1.136E+06
5	3.00	-1.023E+03	-1.705E+06
6	3.00	-1.364E+03	-2.273E+06
7	3.00	-1.023E+03	-1.705E+06
8	2.66	-1.210E+03	-2.017E+06
9	2.66	-6.052E+02	-1.009E+06
10	2.66	-8.266E-10	-1.378E-06
11	2.66	6.052E+02	1.009E+06
12	2.66	6.052E+02	1.009E+06
13	2.66	0.000E+00	0.000E+00
14	2.66	-6.052E+02	-1.009E+06
15	2.66	-1.210E+03	-2.017E+06

==========

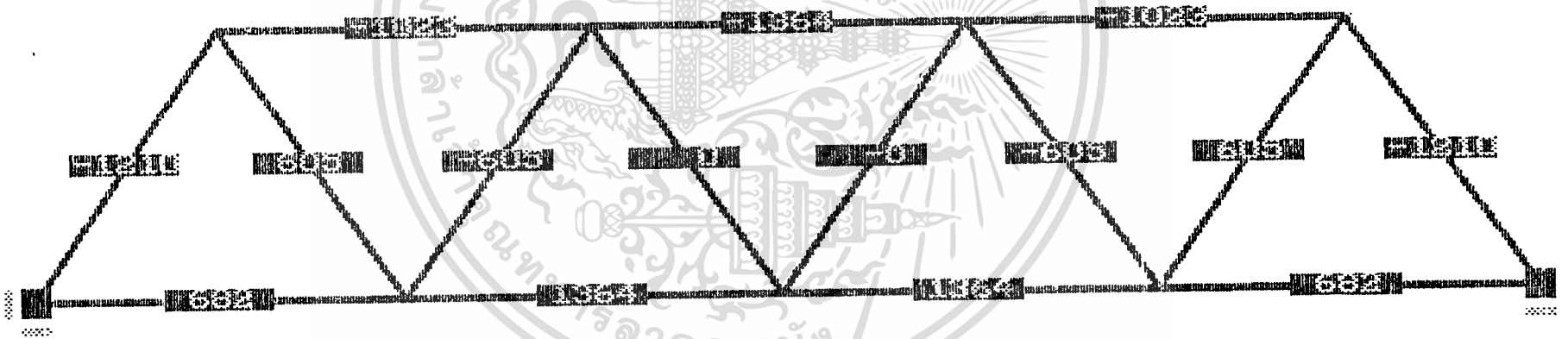
PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : WARREN
ENGINEER : Akarat

==========

DISPLACEMENT RESULTED <2D-TRUSS SYSTEM>

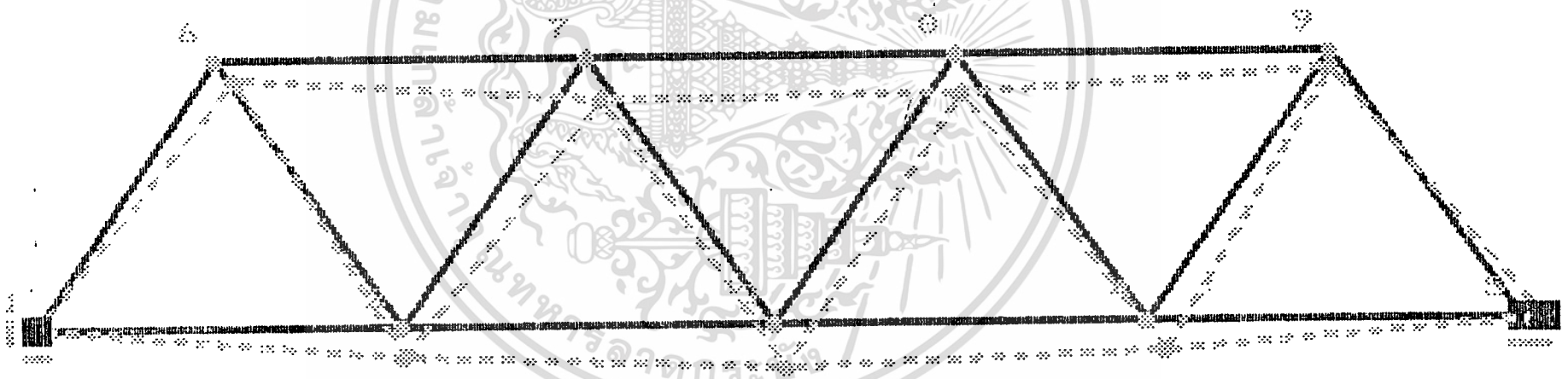
NODE	X-AXIS	Y-AXIS
1	0.000E+00	0.000E+00
2	1.655E-03	-1.326E-02
3	4.965E-03	-1.816E-02
4	8.274E-03	-1.247E-02
5	9.929E-03	0.000E+00
6	8.234E-03	-8.770E-03
7	5.752E-03	-1.763E-02
8	2.442E-03	-1.644E-02
9	-4.039E-05	-6.798E-03

GEOMETRY OF STRUCTURE



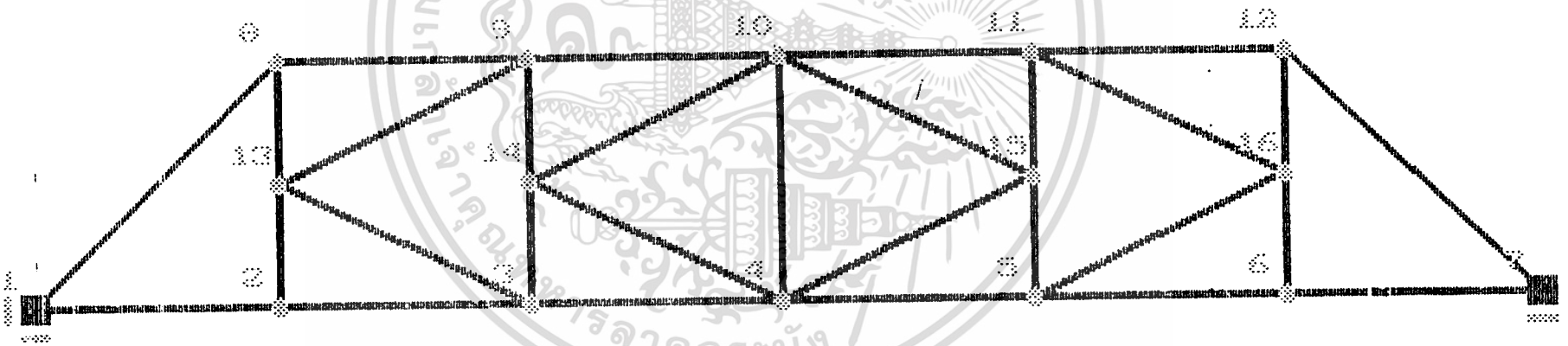
Filename : WARREN.dat

DEFLECTION OF STRUCTURE



Filename : WARREN.dat

GEOMETRY OF STRUCTURE



Filename : K_TRUSS.dat

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : K_TRUSS
ENGINEER : Akarat

COORDINATE DATA <2D-TRUSS SYSTEM>

NODE	X-COOR	Y-COOR
1	0.000E+00	0.000E+00
2	2.000E+00	0.000E+00
3	4.000E+00	0.000E+00
4	6.000E+00	0.000E+00
5	8.000E+00	0.000E+00
6	1.000E+01	0.000E+00
7	1.200E+01	0.000E+00
8	2.000E+00	2.000E+00
9	4.000E+00	2.000E+00
10	6.000E+00	2.000E+00
11	8.000E+00	2.000E+00
12	1.000E+01	2.000E+00
13	2.000E+00	1.000E+00
14	4.000E+00	1.000E+00
15	8.000E+00	1.000E+00
16	1.000E+01	1.000E+00

==========

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : K_TRUSS
ENGINEER : Akarat

==========

ELEMENT CONNECTIVITY DATA <2D-TRUSS SYSTEM>

MEMBER	1-NODE	2-NODE
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6
6	6	7
7	8	9
8	9	10
9	10	11
10	11	12
11	2	13
12	3	14
13	4	10
14	5	15
15	6	16
16	8	13
17	9	14
18	11	15
19	12	16
20	1	8
21	3	13
22	9	13
23	4	14
24	10	14
25	4	15
26	10	15
27	5	16
28	11	16
29	7	12

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : K_TRUSS
ENGINEER : Akarat

LOAD DATA <2D-TRUSS SYSTEM>

NODE	X-COOR	Y-COOR
8	0.000E+00	-2.000E+03
9	0.000E+00	-2.000E+03
10	0.000E+00	-2.000E+03
11	0.000E+00	-2.000E+03
12	0.000E+00	-2.000E+03

```
============================================================

 PLANE TRUSS ANALYSIS                                  PAGE : 1
 PROJECT   : Test Program
 FILENAME  : K_TRUSS
 ENGINEER  : Akarat

============================================================
```

REACTION RESULTED <2D-TRUSS SYSTEM>

Reaction of Node 1
X-AXIS = 0.00
Y-AXIS = 5000.00

Reaction of Node 7
X-AXIS = 0.00
Y-AXIS = 5000.00

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : K_TRUSS
ENGINEER : Akarat

FORCE RESULTED <2D-TRUSS SYSTEM>

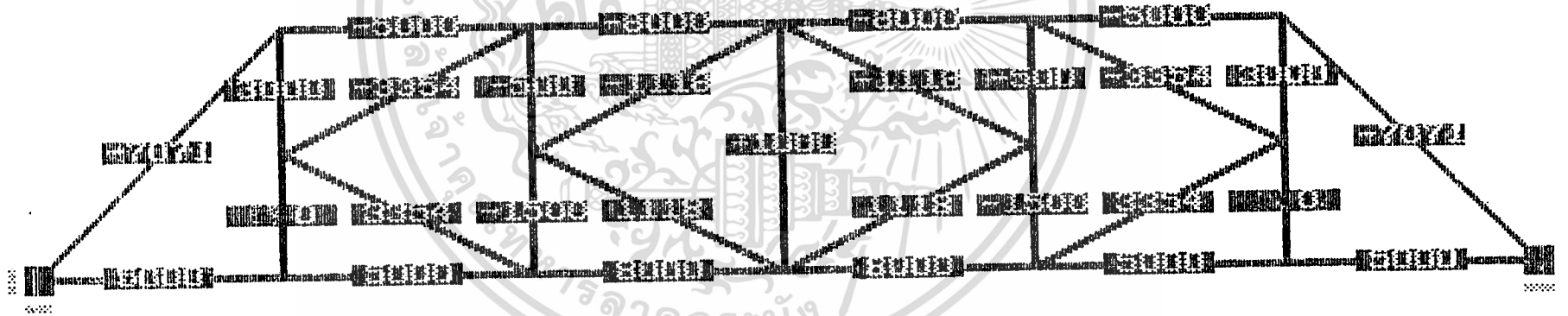
MEMBER	LENGTH	FORCE	STRESS
1	2.00	5.000E+03	8.333E+06
2	2.00	5.000E+03	8.333E+06
3	2.00	8.000E+03	1.333E+07
4	2.00	8.000E+03	1.333E+07
5	2.00	5.000E+03	8.333E+06
6	2.00	5.000E+03	8.333E+06
7	2.00	-5.000E+03	-8.333E+06
8	2.00	-8.000E+03	-1.333E+07
9	2.00	-8.000E+03	-1.333E+07
10	2.00	-5.000E+03	-8.333E+06
11	1.00	0.000E+00	0.000E+00
12	1.00	-1.500E+03	-2.500E+06
13	2.00	-1.000E+03	-1.667E+06
14	1.00	-1.500E+03	-2.500E+06
15	1.00	0.000E+00	0.000E+00
16	1.00	3.000E+03	5.000E+06
17	1.00	-5.000E+02	-8.333E+05
18	1.00	-5.000E+02	-8.333E+05
19	1.00	3.000E+03	5.000E+06
20	2.83	-7.071E+03	-1.179E+07
21	2.24	3.354E+03	5.590E+06
22	2.24	-3.354E+03	-5.590E+06
23	2.24	1.118E+03	1.863E+06
24	2.24	-1.118E+03	-1.863E+06
25	2.24	1.118E+03	1.863E+06
26	2.24	-1.118E+03	-1.863E+06
27	2.24	3.354E+03	5.590E+06
28	2.24	-3.354E+03	-5.590E+06
29	2.83	-7.071E+03	-1.179E+07

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : Test Program
FILENAME : K_TRUSS
ENGINEER : Akarat

DISPLACEMENT RESULTED <2D-TRUSS SYSTEM>

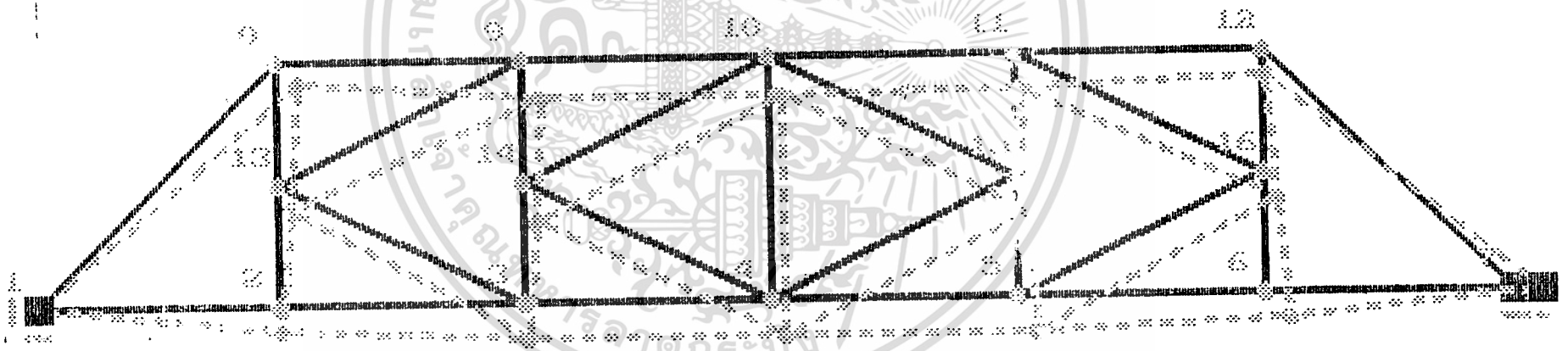
NODE	X-AXIS	Y-AXIS
1	0.000E+00	0.000E+00
2	8.091E-03	-5.218E-02
3	1.618E-02	-7.706E-02
4	2.913E-02	-7.706E-02
5	4.207E-02	-7.625E-02
6	5.016E-02	-5.057E-02
7	5.825E-02	0.000E+00
8	4.976E-02	-4.976E-02
9	4.167E-02	-7.828E-02
10	2.872E-02	-7.868E-02
11	1.578E-02	-7.747E-02
12	7.686E-03	-5.057E-02
13	2.862E-02	-5.218E-02
14	2.852E-02	-7.828E-02
15	2.933E-02	-7.747E-02
16	2.923E-02	-5.057E-02

GEOMETRY OF STRUCTURE



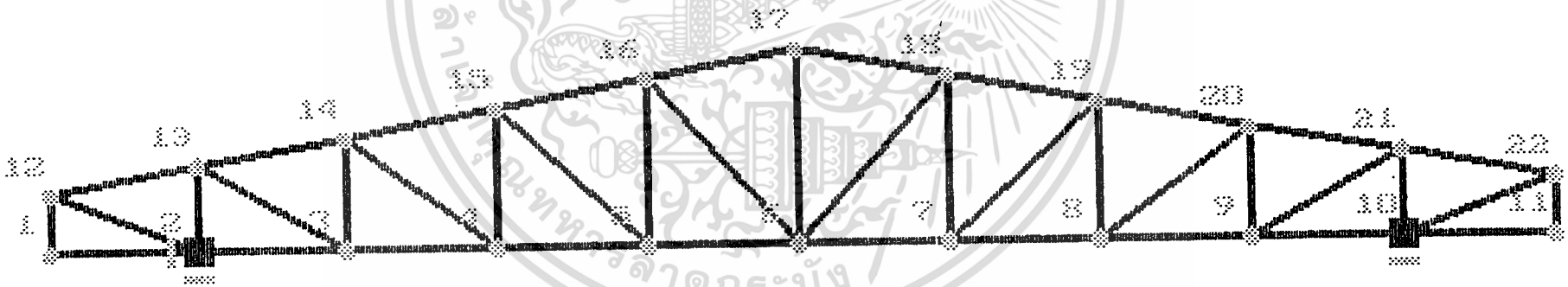
Filename : K_TRUSS.dat

DEFLECTION OF STRUCTURE



Filename : K_TRUSS.dat

GEOMETRY OF STRUCTURE



Filename : DEMO.dat

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : SMALL PLANE TRUSS
FILENAME : DEMO
ENGINEER : SINCHAROEN

COORDINATE DATA <2D-TRUSS SYSTEM>

NODE	X-COOR	Y-COOR
1	0.000E+00	0.000E+00
2	2.000E+02	0.000E+00
3	4.000E+02	0.000E+00
4	6.000E+02	0.000E+00
5	8.000E+02	0.000E+00
6	1.000E+03	0.000E+00
7	1.200E+03	0.000E+00
8	1.400E+03	0.000E+00
9	1.600E+03	0.000E+00
10	1.800E+03	0.000E+00
11	2.000E+03	0.000E+00
12	0.000E+00	8.000E+01
13	2.000E+02	1.160E+02
14	4.000E+02	1.520E+02
15	6.000E+02	1.880E+02
16	8.000E+02	2.240E+02
17	1.000E+03	2.600E+02
18	1.200E+03	2.240E+02
19	1.400E+03	1.880E+02
20	1.600E+03	1.520E+02
21	1.800E+03	1.160E+02
22	2.000E+03	8.000E+01

==========

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : SMALL PLANE TRUSS
FILENAME : DEMO
ENGINEER : SINCHAROEN

==========

ELEMENT CONNECTIVITY DATA <2D-TRUSS SYSTEM>

MEMBER	1-NODE	2-NODE
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6
6	6	7
7	7	8
8	8	9
9	9	10
10	10	11
11	12	13
12	13	14
13	14	15
14	15	16
15	16	17
16	17	18
17	18	19
18	19	20
19	20	21
20	21	22
21	1	12
22	2	13
23	3	14
24	4	15
25	5	16
26	6	17
27	7	18
28	8	19
29	9	20
30	10	21
31	11	22
32	2	12
33	3	13
34	4	14
35	5	15
36	6	16
37	6	18
38	7	19
39	8	20
40	9	21

FILENAME : DEMO
ENGINEER : SINCHAROEN

==========

ELEMENT CONNECTIVITY DATA <2D-TRUSS SYSTEM>

MEMBER	1-NODE	2-NODE
41	10	22

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : SMALL PLANE TRUSS
FILENAME : DEMO
ENGINEER : SINCHAROEN

LOAD DATA <2D-TRUSS SYSTEM>

NODE	X-COOR	Y-COOR
12	0.000E+00	-4.200E+02
13	0.000E+00	-8.400E+02
14	0.000E+00	-8.400E+02
15	0.000E+00	-8.400E+02
16	0.000E+00	-8.400E+02
17	0.000E+00	-8.400E+02
18	0.000E+00	-8.400E+02
19	0.000E+00	-8.400E+02
20	0.000E+00	-8.400E+02
21	0.000E+00	-8.400E+02
22	0.000E+00	-4.200E+02

PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : SMALL PLANE TRUSS
FILENAME : DEMO
ENGINEER : SINCHAROEN

FORCE RESULTED <2D-TRUSS SYSTEM>

MEMBER	LENGTH	FORCE	STRESS
1	200.00	0.000E+00	0.000E+00
2	200.00	-7.241E+02	-7.241E+01
3	200.00	3.316E+03	3.316E+02
4	200.00	4.915E+03	4.915E+02
5	200.00	5.250E+03	5.250E+02
6	200.00	5.250E+03	5.250E+02
7	200.00	4.915E+03	4.915E+02
8	200.00	3.316E+03	3.316E+02
9	200.00	-7.241E+02	-7.241E+01
10	200.00	0.000E+00	0.000E+00
11	203.21	7.358E+02	7.358E+01
12	203.21	-3.369E+03	-3.369E+02
13	203.21	-4.994E+03	-4.994E+02
14	203.21	-5.334E+03	-5.334E+02
15	203.21	-4.924E+03	-4.924E+02
16	203.21	-4.924E+03	-4.924E+02
17	203.21	-5.334E+03	-5.334E+02
18	203.21	-4.994E+03	-4.994E+02
19	203.21	-3.369E+03	-3.369E+02
20	203.21	7.358E+02	7.358E+01
21	80.00	0.000E+00	0.000E+00
22	116.00	-3.910E+03	-3.910E+02
23	152.00	-2.343E+03	-2.343E+02
24	188.00	-1.215E+03	-1.215E+02
25	224.00	-3.150E+02	-3.150E+01
26	260.00	9.046E+02	9.046E+01
27	224.00	-3.150E+02	-3.150E+01
28	188.00	-1.215E+03	-1.215E+02
29	152.00	-2.343E+03	-2.343E+02
30	116.00	-3.910E+03	-3.910E+02
31	80.00	0.000E+00	0.000E+00
32	215.41	-7.799E+02	-7.799E+01
33	231.21	4.670E+03	4.670E+02
34	251.21	2.009E+03	2.009E+02
35	274.49	4.599E+02	4.599E+01
36	300.29	-6.064E+02	-6.064E+01
37	300.29	-6.064E+02	-6.064E+01
38	274.49	4.599E+02	4.599E+01
39	251.21	2.009E+03	2.009E+02
40	231.21	4.670E+03	4.670E+02

==========

PLANE TRUSS ANALYSIS PAGE : 2
PROJECT : SMALL PLANE TRUSS
FILENAME : DEMO
ENGINEER : SINCHAROEN

==========

FORCE RESULTED <2D-TRUSS SYSTEM>

MEMBER	LENGTH	FORCE	STRESS
41	215.41	-7.799E+02	-7.799E+01

==========

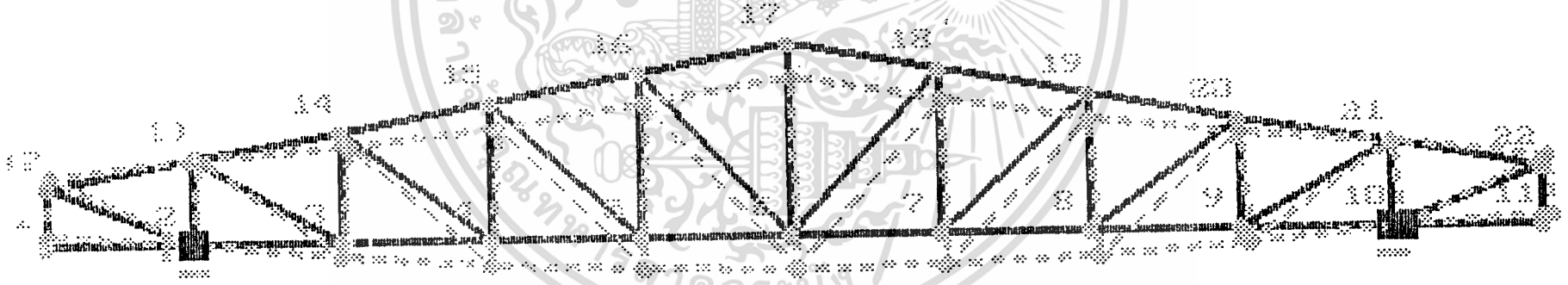
PLANE TRUSS ANALYSIS PAGE : 1
PROJECT : SMALL PLANE TRUSS
FILENAME : DEMO
ENGINEER : SINCHAROEN

==========

DISPLACEMENT RESULTED <2D-TRUSS SYSTEM>

NODE	X-AXIS	Y-AXIS
1	0.000E+00	2.904E-01
2	0.000E+00	0.000E+00
3	-7.099E-03	-3.446E-01
4	2.541E-02	-5.794E-01
5	7.359E-02	-7.091E-01
6	1.251E-01	-7.447E-01
7	1.765E-01	-7.035E-01
8	2.247E-01	-5.683E-01
9	2.572E-01	-3.280E-01
10	2.501E-01	0.000E+00
11	2.501E-01	3.029E-01
12	1.162E-01	2.904E-01
13	1.799E-01	-2.224E-02
14	2.038E-01	-3.446E-01
15	1.955E-01	-5.794E-01
16	1.649E-01	-7.091E-01
17	1.215E-01	-7.447E-01
18	7.901E-02	-7.035E-01
19	4.937E-02	-5.683E-01
20	4.208E-02	-3.280E-01
21	6.701E-02	0.000E+00
22	1.290E-01	3.029E-01

DEFLECTION OF STRUCTURE



Filename : demo.dat

สรุปผลและข้อเสนอแนะ

จากการศึกษาทฤษฎีต่างๆที่ใช้ในการวิเคราะห์และพัฒนาเขียนขึ้นเป็นโปรแกรม พบว่า

1. ทฤษฎีการวิเคราะห์เวคเตอร์เป็นวิธีหนึ่งที่สามารถนำมาใช้ประยุกต์เพื่อวิเคราะห์โครงข้อหมุนได้ดีพอสมควรวิธีหนึ่ง เพราะผลที่ได้จากการคำนวณซึ่งได้ทำการตรวจสอบแล้วพบว่ามีถูกต้องครบถ้วน และหลักการของวิธีนี้สามารถทำการวิเคราะห์ได้รวดเร็ว เพราะหลีกเลี่ยงการแก้สมการหลายตัวแปร โดยอาศัยสมการสมดุล 2 สมการ ใน วิธีสมดุลที่จุดต่อ และ 3 สมการ ในวิธีสมดุลของภาคตัด ซึ่งจะแก้สมการเพียง 2-3 ตัวแปร การคำนวณจึงทำได้รวดเร็ว
2. การใช้เวคเตอร์เข้าช่วยในการเขียนโปรแกรมทำได้ง่ายขึ้น เพราะเวคเตอร์มีทั้งขนาดและทิศทาง เมื่อแทนระบบแรงและโมเมนต์ในรูปของ Catesian Vector แล้วใช้การปฏิบัติของพีชคณิตเวคเตอร์การวิเคราะห์ก็สามารถทำได้ง่าย ทั้งในระบบ 2 มิติ และ 3 มิติ โดยเฉพาะหากนำไปประยุกต์ในระบบ 3 มิติ แล้วจะมีประสิทธิภาพมาก
3. ระบบการเข้าถึงข้อมูลได้ใช้ระบบ MENU ซึ่งเป็นระบบที่คิดว่าง่ายต่อการใช้งานและและง่ายต่อการพัฒนาโปรแกรมต่อไปในอนาคต
4. สำหรับโครงสร้างโครงข้อหมุนชนิดซับซ้อน (Complex Truss) วิธีนี้ไม่สามารถทำการคำนวณได้
5. โปรแกรมนี้ไม่สามารถคิดคำนวณโครงสร้าง Indeterminate ได้เนื่องจากทฤษฎีนี้ยังไม่เพียงพอซึ่งหากมีการพัฒนาต่อไปจะดีทำให้โปรแกรมมีประสิทธิภาพสูงขึ้น

ข้อเสนอแนะ

1. โปรแกรมนี้สามารถวิเคราะห์โครงข้อหมุนได้เพียง 2 มิติ ซึ่งตามทฤษฎีแล้วสามารถนำไปวิเคราะห์กับโครงข้อหมุน 3 มิติได้ ดังนั้นยังสามารถพัฒนาโปรแกรมนี้เพื่อวิเคราะห์โครงข้อหมุน 3 มิติ ต่อไป
2. สามารถพัฒนาโปรแกรมนี้ ให้สามารถออกแบบหน้าตัดของโครงสร้าง เพื่อใช้ในการก่อสร้าง

บรรณานุกรม

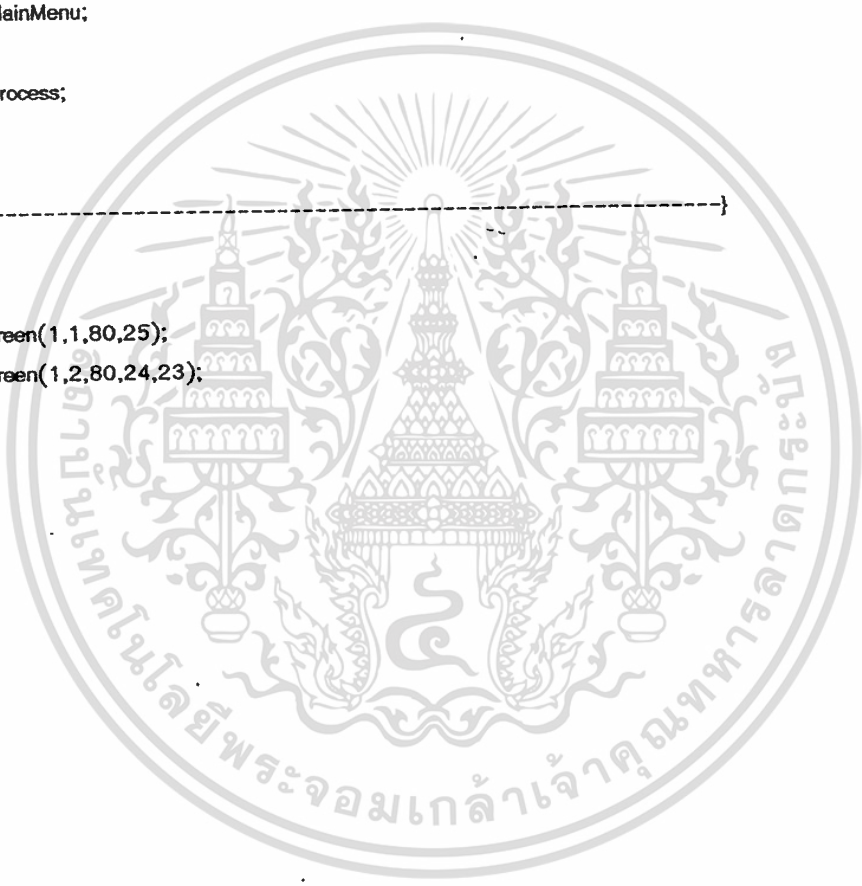
1. J.L. MERIAM AND L.G KRAIGE, ENGINEERING MECHANICS (STATIC) 2nd EDITION , P. 17-22 , 30-32 , 53-63 , 145-163
2. R.C. HIBBELER , ENGINEERING MECHANICS (STATIC) 6th EDITTION , P. 38-53 , 101-114 , 223-258.
3. CHARLES HEAD NORRIS , JOHN BENSON WILBUR , SENOL UTKU ELEMENTARY STRUCTURE ANALYSIS 4th EDITION P. 105-158
4. ผศ. ศิริวัฒน์ ไชยชนะ , การวิเคราะห์โครงสร้าง , พิมพ์ครั้งที่ 2 ,หน้า 7-20 , 36-45
5. อุดม ใยเจริญ , การวิเคราะห์เชิงตัวเลข , พิมพ์ครั้งที่ 3 , หน้า 103-116
6. นุกูล กระจาย,การเขียนโปรแกรมและประมวลผลข้อมูลด้วยเทอร์โบปาสคาล , พิมพ์ครั้งที่ 1
7. สุรศักดิ์ สงวนพงษ์ , การเขียนโปรแกรมขั้นสูง , พิมพ์ครั้งที่ 2

ภาคผนวก ก

```
{$M $4000,0,307200}

program PT2;

uses
  Crt,Dos,Printer,Crt1,Tool,
  Init,Info,Input,Run,Output;

procedure PopMenu(MenuHead: byte);
var
  a,b,i: Byte;
  Ok: boolean;
  F: array[1..5] of string;
begin
  StatusMenu:=MenuHead;
  GetMenuChoice(MenuHead,a,b);
  CursorOff;
  case a of
    1: case b of
        1: UpdateFile(Filename);      {-Open-}
        2: FileStorage(Filename);     {-Save-}
        3: begin
            GetLegal(Ok);
            if Ok then Status := On
             else
              Status := OFF;
           end;                       {-New-}
        4: Directory;                 {-Change Dir-}
        5: Printing;                  {-Print-}
        6: Os_Shell;                  {-Os Shell-}
        7: if QuitStatus then
            Quit:=True;               {-Exit Program-}
       end;
    2: case b of
        1 : CoordinateForm(Status);   {Edit Coordinate}
        2 : BoundaryForm;             {Edit Boundary}
        3 : ConnectForm(Status);      {Edit Connectivity}
        4 : MaterialForm;             {Edit Material}
        5 : LoadForm(Status);         {Edit Load case}
        7 : NodeList;                 {Node List}
        8 : ElementList;              {Element List}
        9 : LoadList;                 {Load List}
       end;
    3: case b of
        1: Executed(1);
        2: Executed(2);               {Force/Displacement}
```

```
    end;
  4: case b of
      1: Resulting(1);
      2: Resulting(2);          {Force}
      3: Resulting(3);          {Displacement}
      4: Resulting(4);
    end;
  5: case b of
      1: Geometry;
      2: DispGeometry;
    end;
  6: case b of
      1 : Helps;
      2 : Informations;
      3 : About;                {About}
    end;
 end;
end;

procedure ControlProcess;
var
  KeyStroke : char;
  KeyPush   : word;
begin
  Quit:=False;
  while Not Quit do
  begin
    KeyStroke:=Readkey;
    if KeyStroke=#0 then
    begin
      KeyStroke:=ReadKey;
      KeyPush:=$100+Ord(KeyStroke);
    end
    else
      KeyPush:=Ord(Upcase(KeyStroke));
    Case KeyPush of
      $0144 : PopMenu(StatusMenu);  {F10}
      $0121 : PopMenu(1);           {Alt-F}
      $0112 : PopMenu(2);           {Alt-E}
      $011F : PopMenu(3);           {Alt-S}
      $0113 : PopMenu(4);           {Alt-R}
      $0122 : PopMenu(5);           {Alt-G}
      $0123 : PopMenu(6);           {Alt-H}
      $012D : if QuitStatus then
                Quit:=True;         {Alt-X}
      $013C : UpdateFile(Filename); {F2}
```

```
    $013D : FileStorage(Filename);  {F3}
    $013F : Directory;              {F5}
   End;
 end;
 ClearScreen(1,1,80,25);
 CursorOn;
end;

{-------------------------------------------------------------------}

procedure Main;
begin
  DisplayMainMenu;
  About;
  ControlProcess;
end;

{-------------------------------------------------------------------}

begin
  ClearScreen(1,1,80,25);
  ColorScreen(1,2,80,24,23);
  Main;
end.
```



```
unit Init;
interface
uses
  Crt;
type
  Joint = record
    PosX : real;
    PosY : real;
    DisX : real;
    DisY : real;
  end;
  Member = record
    First  : integer;
    Second : integer;
    Force  : real;
    Done   : boolean;
    Length : real;
    Area   : real;
    Modulus : real;
  end;
  LoadSize = record
    SizeX : real;
    SizeY : real;
  end;
  BoundDat = record
    Node : integer;
    Dir  : integer;
    Reac : real;
  end;
  BoundRes = record
    Node : integer;
    X    : real;
    Y    : real;
  end;

  NodeType  = array[1..129] of Joint;
  BarType   = array[1..255] of Member;
  BoundType = array[1..3] of BoundDat;
  BResType  = array[1..2] of BoundRes;
  LoadType  = array[1..129] of LoadSize;
  NodeFile  = File of Joint;
  BarFile   = File of Member;
  LoadFile  = File of LoadSize;
  BoundFile = File of BoundDat;
  cc        = array[1..40,1..40] of real;
  dd        = array[1..40] of real;
```

```
const
  Filename : String = 'NONAME';
  Title    : string = '                    ';
  FUnit    : string = '                    ';
  Lunit    : string = '                    ';
  Engineer : string = '                    ';
var
  Node : NodeType;
  Bar  : BarType;
  Load : LoadType;
  Bound: BoundType;
  AccNode,AccBar : integer;
  BRes       : BResType;
  SLoad      : LoadType;
  SBar       : array[1..255] of real;
  SBound     : array[1..3] of real;
  Quit,Changed : boolean;

type
  Dir = (Hor,Ver);
  Legal = (RX,RY,RZ);
  ForceVector = array[1..10] of real;
  BarActive = array[0..10] of integer;
{-------------------------------------------------------------}
implementation
end. {of unit}
```

```
unit Input;
interface
uses
  Crt,Crt1,Init,Info,Tool;
type
  NewStatus = (ON,OFF);
var
  Status: NewStatus;

procedure NodeList;
procedure ElementList;
procedure LoadList;
procedure CheckRow(z,p1,p2: byte;var c,d:byte);
procedure CheckPage(z,r: byte;Ch: char;var Count: byte);
procedure CoordinateForm(Status: NewStatus);
procedure ConnectForm(Status: NewStatus);
procedure BoundaryForm;
procedure LoadForm(Status: NewStatus);
procedure MaterialForm;
procedure FileStorage(var Filename: string);
procedure UpdateFile(var Filename: String);
procedure ClearInit;


{End of interface Section}

implementation


procedure CheckRow(z,p1,p2: byte;var c,d:byte);
begin
  if p2<=(z div p1) then
  begin
    c:=(p2-1)*p1+1;
    d:=p2*p1;
  end
  else
  begin
    c:=(p2-1)*p1+1;
    d:=(p2-1)*p1+z mod p1;
  end;
end;
```



```
procedure CheckPage(z,r: byte;Ch: char;var Count: byte);
const
  NEXT_PAGE = #81;
```

```
  PREVIOUS_PAGE = #73;
begin
  case Ch of
    PREVIOUS_PAGE : if Count >1 then
                      Dec(Count);
    NEXT_PAGE : if Count < (z div r)+1 then
                  Inc(Count);
  end;
end;

procedure NodeList;
const
  TitleColor = White     + (Blue shl 4);
  NormColor  = LightGray + (Blue shl 4);
  BlinkColor = Yellow    + (Blue shl 4);
var
  Start,Stop,Count,i: byte;
  KeyStroke: char;
  DirCode: boolean;
  OldAttr: byte;
begin
  Count:=1;
  OldAttr := TextAttr;
  OpenWindow(1,2,50,19,2,TitleColor,1);
  TextAttr := NormColor;
  PutString(' COORDINATE DATA <2D-TRUSS SYSTEM>',TitleColor,5,1);
  PutString(' NODE    1-COOR    2-COOR  ',TitleColor,5,3);
  PutString(' ************************  ',TitleColor,5,4);
  Window(3,7,45,17);
  CheckRow(AccNode,10,Count,Start,Stop);
  for i:=Start to Stop do
    WriteLn(i:5,Node[i].PosX:14:2,Node[i].PosY:10:2);
  repeat
    KeyStroke:=ReadKey;
    if KeyStroke=#0 then
    begin
      KeyStroke:=ReadKey;
      DirCode:=True;
    end
    else
      DirCode:=False;
    if DirCode then
    begin
      Clrscr;
      CheckPage(AccNode,10,KeyStroke,Count);
      CheckRow(AccNode,10,Count,Start,Stop);
```

```
    for i:=Start to Stop do
      WriteLn(i:5,Node[i].PosX:14:2,Node[i].PosY:10:2);
   end;
  until (DirCode=False) and (KeyStroke=#27);
  CloseWindow;
  TextAttr := OldAttr;
end;

procedure ElementList;
const
  TitleColor = White     + (Blue shl 4);
  NormColor  = LightGray + (Blue shl 4);
  BlinkColor = Yellow    + (Blue shl 4);
var
  Start,Stop,Count,i : byte;
  Status : string;
  KeyStroke : char;
  DirCode : boolean;
  OldAttr: byte;
begin
  Count:=1;
  OldAttr := TextAttr;
  TextAttr := NormColor;
  OpenWindow(1,2,50,19,2,TitleColor,1);
  PutString('ELEMENT CONECTIVITY  <2D-TRUSS SYSTEM>',TitleColor,3,1);
  PutString('ELEM    LENGTH     1-NODE     2-NODE ',TitleColor,3,3);
  PutString('----------------------------------------',TitleColor,3,4);
  Window(3,7,45,17);
  CheckRow(AccBar,10,Count,Start,Stop);
  for i:=Start to Stop do
    WriteLn(i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  repeat
    KeyStroke:=ReadKey;
    if KeyStroke=#0 then
    begin
      KeyStroke:=ReadKey;
      DirCode:=True;
    end
    else
      DirCode:=False;
    if DirCode then
    begin
      Clrscr;
      CheckPage(AccBar,10,KeyStroke,Count);
      CheckRow(AccBar,10,Count,Start,Stop);
      for i:=Start to Stop do
```

```
      WriteLn(i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
    end;
  until (DirCode=False) and (KeyStroke=#27);
  CloseWindow;
  TextAttr := OldAttr;
end;

procedure LoadList;
const
  TitleColor = White     + (Blue shl 4);
  NormColor  = LightGray + (Blue shl 4);
  BlinkColor = Yellow    + (Blue shl 4);

type
  LoadList= array[1..125] of integer;

var
  Start,Stop,Count,i: byte;
  KeyStroke: char;
  DirCode: boolean;
  OldAttr: byte;
  ListLoad: LoadList;
  AccLoad: integer;

procedure SortLoad(var AccLoad: integer;var ListLoad: LoadList);
var
  i,j: byte;
begin
  AccLoad := 0;  j:=0;
  for i:=1 to AccNode do
  begin
    if (Load[i].SizeX<>0) or (Load[i].SizeY<>0) then
    begin
      Inc(AccLoad);
      Inc(j);
      ListLoad[j] := i;
    end;
  end;
end;

begin
  Count:=1;
  OldAttr := TextAttr;
  OpenWindow(1,2,50,19,2,TitleColor,1);
  TextAttr := NormColor;
  PutString(' NODAL POINT LOAD DATA  <2D-TRUSS SYSTEM>',TitleColor,5,1);
```

```
PutString(' NODE    X-FORCE    Y-FORCE  ',TitleColor,5,3);
PutString(' ***************************  ',TitleColor,5,4);
Window(3,7,45,17);
SortLoad(AccLoad,ListLoad);
CheckRow(AccLoad,10,Count,Start,Stop);
for i:=Start to Stop do
 WriteLn(ListLoad[i]:5,Load[ListLoad[i]].SizeX:14:2,Load[ListLoad[i]].SizeY:10:2);
repeat
 KeyStroke:=ReadKey;
 if KeyStroke=#0 then
 begin
  KeyStroke:=ReadKey;
  DirCode:=True;
 end
 else
  DirCode:=False;
 if DirCode then
 begin
  Clrscr;
  CheckPage(AccLoad,10,KeyStroke,Count);
  CheckRow(AccLoad,10,Count,Start,Stop);
  for i:=Start to Stop do
   WriteLn(i:5,Load[ListLoad[i]].SizeX:14:2,Load[ListLoad[i]].SizeY:10:2);
 end;
until (DirCode=False) and (KeyStroke=#27);
CloseWindow;
TextAttr := OldAttr;
end;

procedure ClearNode;
var
 i: byte;
begin
 for i:=1 to 125 do
 begin
  Node[i].PosX := 0;
  Node[i].PosY := 0;
  Node[i].DisX := 0;
  Node[i].DisY := 0;
 end;
end;

procedure ClearElement;
var
 i: byte;
begin
```

```
 for i:=1 to 255 do
 begin
   Bar[i].First  := 0;
   Bar[i].Second := 0;
   Bar[i].Area   := 0;
   Bar[i].Modulus:= 0;
   Bar[i].Length := 0;
   Bar[i].Force  := 0;
 end;
end;

procedure ClearLoad;
var
  i: byte;
begin
  for i:=1 to 125 do
  begin
    Load[i].SizeX := 0;
    Load[i].SizeY := 0;
  end;
end;

procedure PutHeadForm(s: string);
const
  HeadColor  = Yellow + (Cyan shl 4);
var
  i: byte;
begin
  for i:=1 to 80 do
    Putchar(' ',HeadColor,i,1);
  PutString(s,HeadColor,(80-Length(s)) div 2,1)
end;

procedure HelpEdit(s: string; a: integer);
const
  KBDColor  = LightCyan + (Blue shl 4);
  HeadColor = Yellow    + (Blue shl 4);
  NormColor = LightGray + (Blue shl 4);
var
  OldAttr: byte;
begin
  OldAttr := TextAttr;
  OpenWindow(51,2,80,19,1,NormColor,0);
  PutSfring('Edit Window Keyboard',HeadColor,4,1);
  PutString('Node   Description  ',HeadColor,2,3);
  PutString('< 0> : Terminate Input',KBDColor,2,4);
```

```
  PutString('<-1> : Change Total Node',KBDColor,2,5);
  PutString('<-2> : List of Node',KBDColor,2,6);
  PutString('ฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤ',KBDColor,1,7);
  PutString('List Window Keyboard',HeadColor,4,8);
  PutString('<ESC>  Return to Edit Node',KBDColor,2,10);
  PutString('<PgUp> Previous Page',KBDColor,2,11);
  PutString('<PgDn> Next Page',KBDColor,2,12);
  PutString('ฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤ',KBDColor,1,13);
  TextAttr := NormColor;
  GotoXY(2,15);
  Write('No. of total ',s,' : [',a,']');
  TextAttr := OldAttr;
end;

{---------------------------------------------------------------}

procedure CoordinateForm(Status: NewStatus);
var
  ch: char;
const
  ListColor  = Yellow + (Cyan shl 4);
  FormColor  = White  + (LightGray shl 4);
  EditColor  = Black  + (LightGray shl 4);
  InfoColor  = White  + (Magenta shl 4);
  BlankColor = Yellow + (Cyan shl 4);
  NormColor  = White  + (Blue shl 4);
  BlackColor = Yellow;

 function Initnode: boolean;
 var
   OldAttr: byte;
   xx1,yy1,xx2,yy2: integer;
   NodeTem: integer;
 begin
   OldAttr := TextAttr;
   CursorOn;
   OpenWindow(22,8,58,12,1,InfoColor,1);
   Window(23,9,52,11);
   PutString('  Number of Total Node :',InfoColor,1,2);
   PutString('    ',BlankColor,27,2);
   Textattr := BlankColor;
   repeat
     ReadIntNum(27,2,NodeTem);
   until NodeTem>=0;
   if NodeTem>0 then
     AccNode := NodeTem
```

```
  else
    InitNode := NodeTem<>0;
  xx1 := Lo(WindMin);
  yy1 := Hi(WindMin);
  xx2 := Lo(WindMax);
  yy2 := Hi(WindMax);
  Window(51,3,80,18);
  TextAttr := NormColor;
  GotoXY(2,15);
  Write('No.Of Node :  [',AccNode,']');
  Window(xx1,yy1,xx2,yy2);
  TextAttr := OldAttr;
  CloseWindow;
end;

function IndexNode: integer;
const
  ErrorColor = Yellow + (Brown shl 4);
var
  Index,OldAttr: integer;
begin
  repeat
    PutString('  ',Textattr,26,3);
    ReadIntNum(26,3,Index);
    if Index>AccNode then
      Write(#7);
  until Index<=AccNode;
  IndexNode := Index;
end;

function CheckGen(var NodeGen: integer; Index, LastIndex: integer;
                  var Done: boolean): boolean;
begin
  CheckGen:=False;
  if (NodeGen<>0) and (index>0) and (Index<>LastIndex) then
  begin
    if (((Index-LastIndex) mod NodeGen) <> 0) and ((Index-LastIndex) <0) then
      CheckGen := True
    else
      CheckGen := False;
  end;
  if (Index=LastIndex) then
  begin
    NodeGen:=0;
    Done:=False;
  end;
```

```
end;

function CheckIndex(Index: integer): char;
const
  ExitColor = Yellow + (Brown shl 4);
begin
  CheckIndex := 'N';
  CursorOff;
  case Index of
     0 :  begin
            OpenWindow(23,20,57,23,1,ExitColor,1);
            PutString('   Are you Sure to Terminate',ExitColor,2,1);
            PutString('    Input this data <Y/N>?  ',ExitColor,2,2);
            CheckIndex:=Upcase(ReadKey);
            CloseWindow;
          end;
     -1 : begin
            repeat
            until InitNode;
          end;
     -2 : begin
            NodeList;
          end;
  end;
  CursorOn;
end;

procedure Coordinate;
var
  i,Index,LastIndex,NodeGen: integer;
  AddX,AddY: real;
  MaxX,MaxY: real;
  ch: char;
  Done,GenError: boolean;
begin
  WindowFrame(1,2,50,19,1,NormColor,0);
  PutString(' COORDINATE  DATA  <2D-TRUSS SYSTEM>',NormColor,5,1);
  PutString(' NODE     1-COOR     2-COOR    ',NormColor,5,3);
  PutString(' *************************    ',NormColor,5,4);
  HelpEdit('node',AccNode);
  OpenWindow(1,20,80,24,1,BlackColor,0);
  TextAttr := BlackColor;
  Done := False;
  NodeGen := 0;
  GenError := False;
  LastIndex := 0;
```

```
PutString('NODE    X-COOR    Y-COOR    NODAL GEN',BlackColor,25,1);
PutString('PREVIOUS ENTRY :',BlackColor,2,2);
PutString('CURRENT  ENTRY :',BlackColor,2,3);
repeat
 repeat
  Index := IndexNode;
  GenError := CheckGen(NodeGen,Index,LastIndex,Done);
 until Not GenError;
 if (Index>0) then
 begin
  PutString('  ',Textattr,36,3);
  ReadRealNum(36,3,Node[Index].PosX);
  PutString('  ',Textattr,47,3);
  ReadRealNum(47,3,Node[Index].PosY);
  if Done then
  begin
   AddX:=(Node[Index].PosX-Node[LastIndex].PosX)/(Index-LastIndex);
   AddY:=(Node[Index].PosY-Node[LastIndex].PosY)/(Index-LastIndex);
   for i:=1 to ((Index-LastIndex) div NodeGen) do
   begin
    Node[LastIndex+(i*NodeGen)].PosX:=Node[LastIndex].PosX+(AddX*i*NodeGen);
    Node[LastIndex+(i*NodeGen)].PosY:=Node[LastIndex].PosY+(AddY*i*NodeGen);
   end;
  end;
  LastIndex:=Index;
  repeat
   ReadIntNum(58,3,NodeGen);
  until NodeGen>=0;
  if NodeGen=0 then
   Done:=False
   else
    Done:=True;
  GotoXY(36,2); Delline;
  PutString('PREVIOUS ENTRY :',BlackColor,2,2);
  PutString('CURRENT  ENTRY :',BlackColor,2,3);
 end;
 until CheckIndex(Index) = 'Y' ;
 CloseWindow;
 CloseWindow;
end;

procedure SaveCoordinate;
var
 NF: NodeFile;
 k: byte;
begin
```

```
  Assign(NF,Filename+'.COO');
  Rewrite(NF);
  for k:=1 to AccNode do
    Write(NF,Node[k]);
  Close(NF);
end;

begin
  CursorOn;
  PutHeadForm('<< Coordinate Data >>');
  if Initnode then
  begin
    Coordinate;
    SaveCoordinate;
  end;
  DisplayMainMenu;
  ColorScreen(1,2,80,24,16+7);
  WindowBox(1,1,80,23,2);
  CursorOff;
end;

{-------------------------------------------------------------}

procedure ConnectForm(Status: NewStatus);
var
  ch: char;
const
  ListColor  = Yellow + (Cyan shl 4);
  FormColor  = White  + (LightGray shl 4);
  EditColor  = Black  + (LightGray shl 4);
  InfoColor  = White  + (Magenta shl 4);
  BlankColor = Yellow + (Cyan shl 4);
  NormColor  = White  + (Blue shl 4);
  BlackColor = Yellow;

procedure FixElement;
begin
  Accbar := 2*AccNode-3;
  if AccBar<0 then
    AccBar:=0;
end;

function IndexElem: integer;
const
  ErrorColor = Yellow + (Brown shl 4);
var
```

```
  Index,OldAttr: integer;
begin
  repeat
    PutString('  ',Textattr,26,3);
    ReadIntNum(26,3,Index);
    if Index>AccBar then
      Write(#7);
  until Index<=AccBar;
  IndexElem := Index;
end;

function CheckGen(var NodeGen: integer; Index, LastIndex: integer;
                  var Done: boolean): boolean;
begin
  CheckGen:=False;
  if (Index<=LastIndex) then
  begin
    NodeGen:=0;
    Done:=False;
  end;
end;

procedure InitElem;
const
  ExitColor = Yellow + (Brown shl 4);
var
  ch: char;
begin
  OpenWindow(23,20,57,23,1,ExitColor,1);
  PutString(' Structure must be Determinate',ExitColor,2,1);
  PutString(' So  Element = 2*Node-Reaction',ExitColor,2,2);
  ch := ReadKey;
  CloseWindow;
end;

function CheckIndex(Index: integer): char;
const
  ExitColor = Yellow + (Brown shl 4);
begin
  CheckIndex := 'N';
  CursorOff;
  case Index of
    0 : begin
          OpenWindow(23,20,57,23,1,ExitColor,1);
          PutString('  Are you Sure to Terminate',ExitColor,2,1);
          PutString('  Input this data <Y/N>? ',ExitColor,2,2);
```

```
            CheckIndex:=Upcase(ReadKey);
            CloseWindow;
          end;
      -1 : begin
            InitElem;
          end;
      -2 : begin
            ElementList;
          end;
    end;
    CursorOn;
  end;

  function BarLength(i : byte): real;
  var
    Dif_X,Dif_Y,Pow_XY : real;
  begin
    Dif_X  := Node[Bar[i].First].PosX-Node[Bar[i].Second].PosX;
    Dif_Y  := Node[Bar[i].First].PosY-Node[Bar[i].Second].PosY;
    Pow_XY := Sqr(Dif_X)+Sqr(Dif_Y);
    BarLength := Sqrt(Pow_XY);
  end;

  procedure Connectivity;
  var
    i,Index,LastIndex,NodeGen: integer;
    LastX,LastY: real;
    MaxX,MaxY: real;
    ch: char;
    Done,GenError: boolean;
  begin
    WindowFrame(1,2,50,19,1,NormColor,0);
    PutString(' CONNECTIVITY  DATA  <2D-TRUSS SYSTEM>',NormColor,5,1);
    PutString(' ELEM     1-NODE     2-NODE    ',NormColor,5,3);
    PutString(' ************************   ',NormColor,5,4);
    FixElement;
    HelpEdit('element',Accbar);
    OpenWindow(1,20,80,24,1,BlackColor,0);
    TextAttr := BlackColor;
    Done := False;
    NodeGen := 0;
    GenError := False;
    LastIndex := 0;
    PutString('ELEM     1-NODE     2-NODE     NODAL GEN',BlackColor,25,1);
    PutString('PREVIOUS ENTRY :',BlackColor,2,2);
    PutString('CURRENT  ENTRY :',BlackColor,2,3);
```

```
repeat
 repeat
  Index := IndexElem;
  GenError := CheckGen(NodeGen,Index,LastIndex,Done);
 until Not GenError;
 if (Index>0) then
 begin
  PutString('  ',Textattr,36,3);
  ReadIntNum(36,3,Bar[Index].First);
  PutString('  ',Textattr,47,3);
  ReadIntNum(47,3,Bar[Index].Second);
  if Done then
  begin
   for i:=1 to (Index-LastIndex) do
   begin
    Bar[LastIndex+i].First  := Bar[LastIndex].First+NodeGen*i;
    Bar[LastIndex+i].Second := Bar[LastIndex].Second+NodeGen*i;
   end;
  end;
  for i:=1 to Index do
  begin
   Bar[i].Length := BarLength(i);
  end;
  LastIndex:=Index;
  repeat
   ReadIntNum(58,3,NodeGen);
  until NodeGen>=0;
  if NodeGen=0 then
   Done:=False
  else
   Done:=True;
  GotoXY(36,2); Delline;
  PutString('PREVIOUS ENTRY :',BlackColor,2,2);
  PutString('CURRENT  ENTRY :',BlackColor,2,3);
 end;
 until CheckIndex(Index) = 'Y' ;
 CloseWindow;
 CloseWindow;
end;

procedure SaveConnectivity;
var
 BF: BarFile;
 k: byte;
begin
 Assign(BF,Filename+'.ELM');
```

```
  Rewrite(BF);
  for k:=1 to AccBar do
    Write(BF,Bar[k]);
  Close(BF);
end;

begin
  CursorOn;
  PutHeadForm('<< Connectivity Data >>');
  Connectivity;
  SaveConnectivity;
  DisplayMainMenu;
  ColorScreen(1,2,80,24,16+7);
  WindowBox(1,1,80,23,2);
  CursorOff;
end;

{-------------------------------------------------------------}

procedure BoundaryForm;
const
  ListColor  = Yellow + (Cyan shl 4);
  FormColor  = White  + (LightGray shl 4);
  EditColor  = Black  + (LightGray shl 4);
  InfoColor  = White  + (Magenta shl 4);
  BlankColor = Yellow + (Cyan shl 4);
  NormColor  = White  + (Blue shl 4);
  BlackColor = Yellow;
var
  ch: char;

procedure  HelpEdit;
const
  KBDColor  = LightCyan + (Blue shl 4);
  HeadColor = Yellow    + (Blue shl 4);
  NormColor = LightGray + (Blue shl 4);
var
  OldAttr: byte;
begin
  OldAttr := TextAttr;
  OpenWindow(51,2,80,19,1,NormColor,0);
  PutString('Edit Window Keyboard',HeadColor,4,1);
  PutString('< 0> : Terminate Input',KBDColor,2,3);
  PutString('ฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤ',NormColor,1,4);
  PutString('Type of Support',HeadColor,4,5);
  PutString('  Type   1-BOUN   2-BOUN',KBDColor,2,7);
```

```
  PutString('  <H>     L        L ',KBDColor,2,9);
  PutString('  <R>    F.L      F.L',KBDColor,2,10);
  PutString('ฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤ',NormColor,1,13);
  TextAttr := OldAttr;
 end;

 procedure BoundInfo;
 begin
  WindowFrame(1,2,50,19,1,NormColor,0);
  PutString(' BOUNDARY INFORMATION  <2D-TRUSS SYSTEM>',NormColor,3,1);
  PutString(' THIS PROGRAM DEFINE DETERMINATE TRUSS ',NormColor,3,3);
  PutString('          BY USE 2 SUPPORT ',NormColor,3,4);
  PutString('       1)  HINGED SUPPORT <H> ',NormColor,3,6);
  PutString('       2)  ROOLER SUPPORT <R> ',NormColor,3,7);
  PutString('  HINGED SUPPORT ',NormColor,3,10);
  PutString('    DEFINE : LOCK BOTH X-AXIS AND Y-AXIS',NormColor,3,11);
  PutString('  ROOLER SUPPORT ',NormColor,3,13);
  PutString('    DEFINE : LOCK ONE AXIS <X-AXIS OR Y-AXIS>',NormColor,3,14);
  HelpEdit;
 end;

 function Confirm: boolean;
 const
  ConfirmColor = Yellow + (Magenta shl 4);
 begin
  OpenWindow(23,14,57,17,1,ConfirmColor,1);
  PutString('      Do you want to Input  ',ConfirmColor,2,1);
  PutString('   Boundary data Again <Y/N>?',ConfirmColor,2,2);
  Confirm:=Upcase(ReadKey)='N';
  CloseWindow;
 end;

 procedure Boundary;
 var
  Bound1,Bound2: char;
  Dir,Legal: String;
  aa,bb,cc,Index: integer;
  OK: boolean;
 begin
  BoundInfo;
  OpenWindow(1,20,80,24,1,BlackColor,0);
  TextAttr := BlackColor;
  PutString('TYPE    NODE    X-BOUN    Y-BOUN',BlackColor,25,1);
  PutString('FIRST  SUPPORT :',BlackColor,2,2);
  PutString('SECOND SUPPORT :',BlackColor,2,3);
  PutString('<H> :',BlackColor,25,2);
```

```
PutString('<R> :',BlackColor,25,3);
repeat
 ReadIntNum(35,2,Index);
 PutString('L',TextAttr,45,2);
 PutString('L',TextAttr,56,2);
 Bound[1].Node  := Index;
 Bound[2].Node  := Index;
 Bound[1].Dir   := 0;     { Direction is X Axis }
 Bound[2].Dir   := 1;     { Direction is Y Axis }
 ReadIntNum(35,3,Index);
 repeat
  Dir := '                ';
  ReadString(45,3,OK,Dir)
 until (UpcaseStr(Dir)='L') or (UpcaseStr(Dir)='F');
 if (UpcaseStr(Dir) = 'L') then
 begin
  Bound[3].Node:=Index;
  Bound[3].Dir :=0;
  PutString('F',TextAttr,56,3);
 end;
 if (UpcaseStr(Dir) = 'F') then
 begin
  Bound[3].Node:=Index;
  Bound[3].Dir :=1;
  PutString('L',TextAttr,56,3);
 end;
until Confirm;
CloseWindow;
CloseWindow;
end;

procedure SaveBoundary;
var
 RF: BoundFile;
 k: byte;
begin
 Assign(RF,Filename+'.BOU');
 Rewrite(RF);
 for k:=1 to 3 do
  Write(RF,Bound[k]);
 Close(RF);
end;

begin
 CursorOn;
 PutHeadForm('<< Boundary Condition Data >>');
```

```
 Boundary;
 SaveBoundary;
 DisplayMainMenu;
 ColorScreen(1,2,80,24,16+7);
 WindowBox(1,1,80,23,2);
 CursorOff;
end;

{-------------------------------------------------------------}

procedure LoadForm(Status: NewStatus);
var
 ch: char;
const
 ListColor  = Yellow + (Cyan shl 4);
 FormColor  = White  + (LightGray shl 4);
 EditColor  = Black  + (LightGray shl 4);
 InfoColor  = White  + (Magenta shl 4);
 BlankColor = Yellow + (Cyan shl 4);
 NormColor  = White  + (Blue shl 4);
 BlackColor = Yellow;

function IndexLoad: integer;
const
 ErrorColor = Yellow + (Brown shl 4);
var
 Index,OldAttr: integer;
begin
 repeat
  PutString('  ',Textattr,26,3);
  ReadIntNum(26,3,Index);
  if Index>AccNode then
   Write(#7);
 until Index<=AccNode;
 IndexLoad := Index;
end;

function CheckGen(var NodeGen: integer; Index, LastIndex: integer;
                  var Done: boolean): boolean;
begin
 CheckGen:=False;
 if (Index<=LastIndex) then
 begin
  NodeGen:=0;
  Done:=False;
 end;
```

```
end;

procedure InitLoad;
const
  ExitColor = Yellow + (Brown shl 4);
var
  ch: char;
begin
  OpenWindow(23,20,57,23,1,ExitColor,1);
  ch := ReadKey;
  CloseWindow;
end;

function CheckIndex(Index: integer): char;
const
  ExitColor = Yellow + (Brown shl 4);
begin
  CheckIndex := 'N';
  CursorOff;
  case Index of
     0 :  begin
            OpenWindow(23,20,57,23,1,ExitColor,1);
            PutString('   Are you Sure to Terminate',ExitColor,2,1);
            PutString('    Input this data <Y/N>?  ',ExitColor,2,2);
            CheckIndex:=Upcase(ReadKey);
            CloseWindow;
          end;
     -1 : begin
          end;
     -2 : begin
            LoadList;
          end;
   end;
   CursorOn;
  end;

  procedure PointLoad;
  var
    i,Index,LastIndex,NodeGen: integer;
    LastX,LastY: real;
    MaxX,MaxY: real;
    ch: char;
    Done,GenError: boolean;
  begin
    WindowFrame(1,2,50,19,1,NormColor,0);
    PutString(' NODAL POINT LOAD DATA  <2D-TRUSS SYSTEM>',NormColor,5,1);
```

```
PutString(' NODE    X-FORCE    Y-FORCE  ',NormColor,5,3);
PutString(' ***************************  ',NormColor,5,4);
HelpEdit('Node',AccNode);
OpenWindow(1,20,80,24,1,BlackColor,0);
TextAttr := BlackColor;
Done := False;
NodeGen := 0;
GenError := False;
LastIndex := 0;
PutString('NODE    X-FORCE    Y-FORCE    NODAL GEN',BlackColor,25,1);
PutString('PREVIOUS ENTRY :',BlackColor,2,2);
PutString('CURRENT  ENTRY :',BlackColor,2,3);
repeat
 repeat
  Index := IndexLoad;
  GenError := CheckGen(NodeGen,Index,LastIndex,Done);
 until Not GenError;
 if (Index>0) then
 begin
  PutString('  ',Textattr,36,3);
  ReadRealNum(36,3,Load[Index].SizeX);
  PutString('  ',Textattr,47,3);
  ReadRealNum(47,3,Load[Index].SizeY);
  if Done then
  begin
   for i:=1 to (Index-LastIndex) do
   begin
    Load[LastIndex+NodeGen*i].SizeX := Load[LastIndex].SizeX;
    Load[LastIndex+NodeGen*i].SizeY := Load[LastIndex].SizeY;
   end;
  end;
  LastIndex:=Index;
  repeat
   ReadIntNum(58,3,NodeGen);
  until NodeGen>=0;
  if NodeGen=0 then
   Done:=False
   else
    Done:=True;
  GotoXY(36,2); Delline;
  PutString('PREVIOUS ENTRY :',BlackColor,2,2);
  PutString('CURRENT  ENTRY :',BlackColor,2,3);
 end;
until CheckIndex(Index) = 'Y' ;
CloseWindow;
CloseWindow;
```

```
end;

procedure SaveLoadData;
var
  LF: LoadFile;
  k: byte;
begin
  Assign(LF,Filename+'.NPL');
  Rewrite(LF);
  for k:=1 to AccNode do
    Write(LF,Load[k]);
  Close(LF);
end;

begin
  CursorOn;
  PutHeadForm('<< Nodal Point Load Data >>');
  PointLoad;
  SaveLoadData;
  DisplayMainMenu;
  ColorScreen(1,2,80,24,16+7);
  WindowBox(1,1,80,23,2);
  CursorOff;
end;

{-------------------------------------------------------------------}

procedure MaterialForm;
const
  ListColor  = Yellow + (Cyan shl 4);
  FormColor  = White  + (LightGray shl 4);
  EditColor  = Black  + (LightGray shl 4);
  InfoColor  = White  + (Magenta shl 4);
  BlankColor = Yellow + (Cyan shl 4);
  NormColor  = White  + (Blue shl 4);
  BlackColor = Yellow;
var
  ch: char;

procedure  HelpEdit;
const
  KBDColor  = LightCyan + (Blue shl 4);
  HeadColor = Yellow    + (Blue shl 4);
  NormColor = LightCyan + (Blue shl 4);
var
  OldAttr: byte;
```

```
begin
  OldAttr := TextAttr;
  OpenWindow(51,2,80,19,1,NormColor,0);
  PutString('Edit Window Keyboard',HeadColor,4,1);
  PutString('< 0> : Terminate Input',KBDColor,2,3);
  PutString('ฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤ',NormColor,1,5);
  PutString('Type of Support',HeadColor,4,6);
  PutString(' Type   1-BOUN   2-BOUN',KBDColor,2,8);
  PutString(' <H>     L         L ',KBDColor,2,10);
  PutString(' <R>     F,L       F,L',KBDColor,2,11);
  TextAttr := OldAttr;
end;

procedure BoundInfo;
begin
  WindowFrame(1,2,50,19,1,NormColor,0);
  PutString(' MATERIAL INFORMATION  <2D-TRUSS SYSTEM>',NormColor,3,1);
  PutString(' THIS PROGRAM DEFINE DETERMINATE TRUSS ',NormColor,3,3);
  PutString('         BY USE 2 SUPPORT ',NormColor,3,4);
  PutString('       1)  HINGED SUPPORT <H> ',NormColor,3,6);
  PutString('       2)  ROOLER SUPPORT <R> ',NormColor,3,7);
  PutString(' HINGED SUPPORT ',NormColor,3,10);
  PutString('    DEFINE : LOCK BOTH X-AXIS AND Y-AXIS',NormColor,3,11);
  PutString(' ROOLER SUPPORT ',NormColor,3,13);
  PutString('    DEFINE : LOCK ONE AXIS <X-AXIS OR Y-AXIS>',NormColor,3,14);
  HelpEdit;
end;

function Confirm: boolean;
const
  ConfirmColor = Yellow + (Magenta shl 4);
begin
  OpenWindow(23,14,57,17,1,ConfirmColor,1);
  PutString('      Do you want to Input   ',ConfirmColor,2,1);
  PutString('   Material data Again <Y/N>?',ConfirmColor,2,2);
  Confirm:=Upcase(ReadKey)='N';
  CloseWindow;
end;

procedure Material;
var
  i,Index: integer;
  Area,Modulus: real;
begin
  BoundInfo;
  OpenWindow(1,20,80,24,1,BlackColor,0);
```

```
TextAttr := BlackColor;
PutString(' SET    AXIAL-AREA    E-MODULUS',BlackColor,25,1);
PutString('MATERIAL PROPERTY :',BlackColor,2,2);
{utString('ELEMENT LIST     :',BlackColor,2,3);}
repeat
  ReadIntNum(27,2,Index);
  if Index<>0 then
  begin
    ReadRealNum(38,2,Area);
    ReadRealNum(51,2,Modulus);
  end;
  for i:=1 to Accbar do
  begin
    Bar[i].Area := Area;
    Bar[i].Modulus := Modulus;
  end;
 until Confirm;
 CloseWindow;
 CloseWindow;
end;

procedure SaveMaterial;
var
  BF: BarFile;
  k: byte;
begin
  Assign(BF,Filename+'.ELM');
  Rewrite(BF);
  for k:=1 to AccBar do
    Write(BF,Bar[k]);
  Close(BF);
end;

begin
  CursorOn;
  PutHeadForm('<< Material Property Data >>');
  Material;
  SaveMaterial;
  DisplayMainMenu;
  ColorScreen(1,2,80,24,16+7);
  WindowBox(1,1,80,23,2);
  CursorOff;
end;

{-------------------------------------------------------------}
```

```
procedure FileStorage(var Filename: string);
var
  NF  : NodeFile;
  BF  : BarFile;
  LF  : LoadFile;
  RF  : BoundFile;
  afi : Text;
  k : byte;
begin
  if BlockFile(Filename) then
    if Filename<>#27 then
    begin
      Assign(afi,FileExtend(Filename,'DAT'));
      Rewrite(afi);
      WriteLn(afi,Title,LUnit,FUnit,Engineer);
      Close(afi);

      Assign(NF,FileExtend(Filename,'COO'));
      Rewrite(NF);
      for k:=1 to AccNode do
        Write(NF,Node[k]);
      Close(NF);

      Assign(BF,FileExtend(Filename,'ELM'));
      Rewrite(BF);
      for k:=1 to AccBar do
        Write(BF,Bar[k]);
      Close(BF);

      Assign(LF,FileExtend(Filename,'NPL'));
      Rewrite(LF);
      for k:=1 to AccNode do
        Write(LF,Load[k]);
      Close(LF);

      Assign(RF,FileExtend(Filename,'BOU'));
      Rewrite(RF);
      for k:=1 to 3 do
        Write(RF,Bound[k]);
      Close(RF);
    end;
  end;

  procedure UpdateFile(var Filename: String);
  var
    NF  : NodeFile;
```

```
    BF  : BarFile;
    LF  : LoadFile;
    RF  : BoundFile;
    afi : Text;
    k,i : byte;
    F   : array[1..5] of string;
  begin
   if BlockFile(Filename) then
    if Filename<>#27 then
    begin
      ClearNode;
      ClearElement;
      ClearLoad;
     for i:=1 to 5 do
      F[i] := Filename;
     GroupExtend(F[1],F[2],F[3],F[4],F[5]);
     if FileList(F[1],F[2],F[3],F[4],F[5]) then
     begin
      Assign(afi,FileExtend(Filename,'DAT'));
      Reset(afi);
      ReadLn(afi,Title,LUnit,FUnit,Engineer);
      Close(afi);

      Assign(NF,Filename+'.COO');
      Reset(NF);
      AccNode := FileSize(NF);
      for k:=1 to AccNode do
      Read(NF,Node[k]);
      Close(NF);

      Assign(BF,Filename+'.ELM');
      Reset(BF);
      AccBar := FileSize(BF);
      for k:=1 to AccBar do
      Read(BF,Bar[k]);
      Close(BF);

      Assign(LF,Filename+'.NPL');
      Reset(LF);
      for k:=1 to AccNode do
      Read(LF,Load[k]);
      Close(LF);

      Assign(RF,Filename+'.BOU');
      Reset(RF);
      for k:=1 to 3 do
```

```
        Read(RF,Bound[k]);
      Close(RF);
    end
    else
      FileNotFound;
  end;
end;

procedure ClearInit;
begin
  Status := OFF;
end;

begin
  AccNode := 0;
  AccBar  := 0;
  ClearNode;
  ClearElement;
  ClearLoad;
end. {of Unit}
```

```
unit Run;
interface
uses
  Init,Crt,Crt1,Input,Output;
procedure  Executed(code: byte);

{end of interface section}
implementation

{=============================================================}

          {- Mathematics Operating Function -}


function Max(X1, X2: real): real;
begin
  if X1<X2 then
    Max := X2
  else
    Max := X1;
end;

function Min(X1, X2: real): real;
begin
  if X1>X2 then
    Min := X2
  else
    Min := X1;
end;

function Negative(value: real): real;
begin
  Negative := -value;
end;

procedure SwapRow(n: byte; var a: cc; var b: dd; var status: boolean);
var
  i,j,k: byte;
  Swap: boolean;
  Temp: real;
  Passed: boolean;
begin
  Swap := False;
  Passed := True;
  while Passed do
  begin
```

```
for i:=1 to n do
begin
  if a[i,i]=0 then
    for j:=1 to n do
    begin
      if (a[j,i]<>0) then
      begin
        for k:=1 to n do
        begin
          Temp := a[j,k];
          a[j,k] := a[i,k];
          a[i,k] := Temp;
        end;
        Temp := b[j];
        b[j] := b[i];
        b[i] := Temp;
        j := n;
        Swap := True;
      end;
      if (j=n) and (Not Swap) then
      begin
        i := n;
        Status := False;
        Passed := False;
      end;
    end;
    Swap := False;
  end;
  if Passed then
    for i:=1 to n do
      if a[i,i]=0 then
        Passed := true
        else
        begin
          Passed := False;
          Status := True;
        end;
  end;
end;

procedure Pivat(var k, n: byte; var a: cc; var b: dd; var status: boolean);
var
  Temp: real;
  p: real;
  m,i,j,count: byte;
begin
```

```
  Pivat(k,n,a,b,Status);
 if Status then
 begin
  for k:=1 to n-1 do
  begin
   for i:=k+1 to n do
   begin
    q:=a[i,k]/a[k,k];
    for j:=k+1 to n do
     a[i,j] := a[i,j]-q*a[k,j];
    b[i] :=  b[i]-q*b[k];
   end;
  end;
  x[n] := b[n]/a[n,n];
  for k:=n-1 downto 1 do
  begin
   W:=0;
   for j:=k+1 to n do
    W:=W+a[k,j]*x[j];
   x[k]:=(b[k]-W)/a[k,k];
  end;
  for k:=1 to n do
   b[k] := x[k];
  Gauss := True;
 end
 else
  Gauss := False;
end;

{=============================================================}

            {- Vector Analysis Function -}


function Minode(N1,N2: byte): byte;
begin
 if Node[N1].PosX>Node[N2].PosX then
  Minode:=N2
  else
   Minode:=N1;
end;

function Manode(N1,N2: byte): byte;
begin
 if Node[N1].PosX>Node[N2].PosX then
  Manode:=N1
```

```
    else
      Manode:=N2;
  end;

  function CompatibleNode(i,j : word) : byte;
  begin
   if (Bar[i].First = j) then
     CompatibleNode := Bar[i].Second
     else
       CompatibleNode := Bar[i].First;
  end;

  function GetVectorX(Get1,Get2 : word) : real;
  var
    Dif_X, Dif_Y, Pow_XY : real;
  begin
    Dif_X  := Node[Get1].PosX-Node[Get2].PosX;
    Dif_Y  := Node[Get1].PosY-Node[Get2].PosY;
    Pow_XY := Sqr(Dif_X)+Sqr(Dif_Y);
    GetVectorX := Dif_X/Sqrt(Pow_XY);
  end;

  function GetVectorY(Get1,Get2 : word) : real;
  var
    Dif_X, Dif_Y, Pow_XY : real;
  begin
    Dif_X  := Node[Get1].PosX-Node[Get2].PosX;
    Dif_Y  := Node[Get1].PosY-Node[Get2].PosY;
    Pow_XY := Sqr(Dif_X)+Sqr(Dif_Y);
    GetVectorY := Dif_Y/Sqrt(Pow_XY);
  end;

  procedure CreatVectorMember(IndexNode,RefNode,k: byte;
                        var VecX,VecY: ForceVector);
  begin
    VecX[k] := GetVectorX(IndexNode,RefNode);
    VecY[k] := GetVectorY(IndexNode,RefNode);
  end;

  function JointActive(RefNode : word;var MemActive,
                    MemNotActive : BarActive) : boolean;
  var
    i,j,k : byte;
  begin
    j:=0; k:=0;
    for i:=1 to AccBar do
```

```
begin
  if (Bar[i].First = RefNode) or (Bar[i].Second = RefNode) then
  begin
    if (Bar[i].Done = True) then
    begin
      Inc(j);
      MemActive[j] := i;
    end
    else
    begin
      Inc(k);
      MemNotActive[k] := i;
    end;
  end;
end;
MemActive[0] := j;   MemNotActive[0] := k;
if (MemNotActive[0] <1) or (MemNotActive[0]>2)then
  JointActive := False
  else
    JointActive := True;
end;

function ProcessComplete:boolean;
var
  j : byte;
begin
  ProcessComplete := True;
  for j:=1 to AccBar do
    if Bar[j].Done = False then
      ProcessComplete := False;
end;

{=============================================================}

              {- Set Solution Variable -}


procedure SetInitial;
var
  i : byte;
begin
  for i:=1 to AccBar do
  begin
    Bar[i].Force := 0;
    Bar[i].Done := False;
  end;
```

```
end;

procedure SetInitZero;
var
  i: byte;
begin
  for i:=1 to AccBar do
  begin
    Bar[i].Force := 0;
    Bar[i].Done := False;
  end;
  AccBar:=0; AccNode:=0;
end;

procedure SetZeroLoad;
var
  i : byte;
begin
  for i:=1 to AccNode do
  begin
    Load[i].SizeX := 0;
    Load[i].SizeY := 0;
  end;
end;

procedure SaveInitLoad;
var
  i : byte;
begin
  for i:=1 to AccNode do
  begin
    SLoad[i].SizeX := Load[i].SizeX;
    SLoad[i].SizeY := Load[i].SizeY;
  end;
  for i:=1 to AccBar do
    SBar[i] := Bar[i].Force;
  for i:=1 to 3 do
    SBound[i] := Bound[i].Reac;
end;

procedure RestoreInitLoad;
var
  i : byte;
begin
  for i:=1 to 3 do
    Bound[i].Reac := SBound[i];
```

```
  for i:=1 to AccBar do
    Bar[i].Force:=SBar[i];
  for i:=1 to AccNode do
  begin
    Load[i].SizeX := SLoad[i].SizeX;
    Load[i].SizeY := SLoad[i].SizeY;
  end;
end;

procedure PutBoundRes;
begin
  if (Bound[1].Node <> Bound[2].Node) then
  begin
    BRes[1].Node := Bound[1].Node;
    BRes[2].Node := Bound[2].Node;
    if (Bound[1].Dir=0) then
    begin
      BRes[1].X := Bound[1].Reac;
      BRes[1].Y := 0;
    end
    else
    begin
      BRes[1].X := 0;
      BRes[1].Y := Bound[1].Reac;
    end;
  end
  else
  begin
    begin
      BRes[1].Node := Bound[1].Node;
      BRes[2].Node := Bound[3].Node;
      BRes[1].X := Bound[1].Reac;
      BRes[1].Y := Bound[2].Reac;
      if Bound[3].Dir = 0 then
      begin
        BRes[2].X := Bound[3].Reac;
        BRes[2].Y := 0;
      end
      else
      begin
        BRes[2].X := 0;
        BRes[2].Y := Bound[3].Reac;
      end;
    end;
  end;
end;
```

```
{================================================================}

                {- Function Find Reaction -}


function GetReaction: boolean;
const
  ErrorColor = Yellow + (Magenta shl 4);
var
  a : cc;
  b : dd;
  k : byte;

function  ACC_RF(Code : Legal) : real;
var
  AccLoad : real;
  k : byte;
begin
  AccLoad := 0;
  case Code of
    RX : for k:=1 to AccNode do
           AccLoad := AccLoad+Load[k].SizeX;
    RY : for k:=1 to AccNode do
           AccLoad := AccLoad+Load[k].SizeY;
    RZ : for k:=1 to AccNode do
           AccLoad := AccLoad+(Load[k].SizeX)*(Node[k].PosY)
                     -(Load[k].SizeY)*(Node[k].PosX);
  end;
  ACC_RF := Negative(AccLoad);
end;

procedure EquationActive;
var
  j : byte;
begin
  for j:=1 to 3 do
  begin
    if (Bound[j].Dir = 0) then
    begin
      a[1,j] := 1;
      a[2,j] := 0;
    end;
    if (Bound[j].Dir = 1) then
    begin
      a[1,j] := 0;
```

```
      a[2,j] := 1;
    end;
    a[3,j] := (a[1,j]*Node[Bound[j].Node].PosY)-
              (a[2,j]*Node[Bound[j].Node].PosX);
  end;
  b[1] := ACC_RF(RX);
  b[2] := ACC_RF(RY);
  b[3] := ACC_RF(RZ);
end;

begin
  EquationActive;
  if Not Gauss(3,a,b) then
  begin
    OpenWindow(15,8,65,12,1,ErrorColor,1);
    PutString(' Error Input please cheak Boundary Condition',ErrorColor,1,2);
    Wait;
    CloseWindow;
    GetReaction := False;
  end
  else
  begin
    for k:=1 to 3 do
      Bound[k].Reac := b[k];
    PutBoundRes;
    GetReaction := True;
  end;
end;

{=============================================================}

        {- Vector Analysis with Method of Joint -}


procedure JointProcess(var ErrorCode,DoneCode: boolean);
const
  ErrorColor = Yellow + (Magenta shl 4);
var
  k,j : byte;
  a : cc;
  p : dd;
  MemVecX, MemVecY, ForceVecX, ForceVecY : ForceVector;
  MemActive,MemNotActive : BarActive;
  IndexNode : byte;

procedure SaveChange;
```

```
begin
  Changed := True;
end;

function ProcessNotActive : Boolean;
begin
  if (Changed = True) then
  begin
    ProcessNotActive := True;
    Changed := False;
  end
  else
    ProcessNotActive := False;
end;

function AccLoadX(RefNode: Byte;MemActive: BarActive;
                  ForceVectorX: ForceVector) : real;
var
  Temp : real;
  k : byte;
begin
  Temp := 0;
  for k:=1 to MemActive[0] do
    Temp := Temp+ForceVectorX[k]*Bar[MemActive[k]].Force;
  for k:=1 to 2 do
    if (BRes[k].Node=RefNode) then
      Temp := Temp+BRes[k].X;
  Temp := Temp+Load[RefNode].SizeX;
  AccLoadX := Negative(Temp);
end;

function AccLoadY(RefNode: byte;MemActive : BarActive;
                  ForceVectorY: ForceVector) : real;
var
  Temp : real;
  k : byte;
begin
  Temp := 0;
  for k:=1 to MemActive[0] do
    Temp := Temp+ForceVectorY[k]*Bar[MemActive[k]].Force;
  for k:=1 to 2 do
    if (BRes[k].Node=RefNode) then
      Temp := Temp+BRes[k].Y;
  Temp := Temp+Load[RefNode].SizeY;
  AccLoadY := Negative(Temp);
end;
```

```
procedure FormVectorEquation(RefNode: byte;MemActive,
                        MemNotActive: BarActive);
begin
  case MemNotActive[0] of
    1: begin
         if MemVecX[1]<> 0 then
         begin
           a[1,1] := MemVecX[1];
           p[1] := AccLoadX(RefNode,MemActive,ForceVecX);
         end
         else
         begin
           a[1,1] := MemVecY[1];
           p[1] := AccLoadY(RefNode,MemActive,ForceVecY);
         end;
       end;
    2: begin
         a[1,1] := MemVecX[1];
         a[1,2] := MemVecX[2];
         a[2,1] := MemVecY[1];
         a[2,2] := MemVecY[2];
         p[1] := AccLoadX(RefNode,MemActive,ForceVecX);
         p[2] := AccLoadY(RefNode,MemActive,ForceVecY);
       end;
  end;
end;

procedure PutForceValue(Active : byte);
var
  j : byte;
begin
  for j:=1 to Active do
  begin
    Bar[MemNotActive[j]].Force := p[j];
    Bar[MemNotActive[j]].Done := True;
  end;
end;

begin
  ErrorCode := False;
  DoneCode := False;
  SaveChange;
  while (ProcessNotActive) and (Not ErrorCode) do
  begin
    for k:=1 to AccNode do
```

```
  begin
    if JointActive(k,MemActive,MemNotActive) then
    begin
      for j:=1 to MemNotActive[0] do
      begin
        IndexNode := CompatibleNode(MemNotActive[j],k);
        CreatVectorMember(IndexNode,k,j,MemVecX,MemVecY);
      end;
      for j:=1 to MemActive[0] do
      begin
        IndexNode := CompatibleNode(MemActive[j],k);
        CreatVectorMember(IndexNode,k,j,ForceVecX,ForceVecY);
      end;
      FormVectorEquation(k,MemActive,MemNotActive);
      if Gauss(MemNotActive[0],a,p) then
      begin
        PutForceValue(MemNotActive[0]);
        SaveChange;
        DoneCode := True;
      end
      else
      begin
        ErrorCode := True;
        k := AccNode;
        OpenWindow(15,8,65,15,1,ErrorColor,1);
        PutString(' The Arragement of member truss is not  ',ErrorColor,2,2);
        PutString(' Property, structure will not stability ',ErrorColor,2,3);
        PutString(' Please check data again.....  ',ErrorColor,2,4);
        Beep(2);
        Wait;
        CloseWindow;
      end;
    end;
  end;
end;

{================================================================}

        {- Vector Analysis with Method of Section -}


procedure SectionProcess(var ErrorCode,DoneCode: boolean);
type BarPass = array[0..10] of byte;
const
  ErrorColor = Yellow + (Magenta shl 4);
```

```
var
  Sort,Check : array[0..129] of byte;
  Temp : array[0..129] of real;
  MemPass,MemPassI,MemPassII: BarPass;
  MemVecX, MemVecY, ForceVecX, ForceVecY : ForceVector;
  IndexNode,RefNode : byte;
  a : cc;
  b : dd;
  i,j : byte;

procedure Sorting;
var
  i,j,k,Y : byte;
  X : real;
begin
  for k:=1 to AccNode do
  begin
    Temp[k] := Node[k].PosX;
    Check[k] := k;
  end;
  for i:=2 to AccNode do
  begin
    for j:=AccNode downto i do
      if Temp[j-1] > Temp[j] then
      begin
        X := Temp[j-1];
        Temp[j-1] := Temp[j];
        Temp[j] := X;
        Y := Check[j-1];
        Check[j-1] := Check[j];
        Check[j] := Y;
      end;
  end;
end;

procedure SetRange;
var
  j,k : byte;
begin
  Sort[1] := Check[1]; j:=2;
  for k:=2 to AccNode do
    if Temp[k] > Temp[k-1] then
    begin
      Sort[j] := Check[k];
      Inc(j);
    end;
```

```
  Sort[0] := j-2;
end;

function JointVenture(a,b,c : byte) : boolean;
var
  Pass1,Pass2 : boolean;
begin
  if ((Bar[a].First<>Bar[b].First) or (Bar[a].First<>Bar[b].Second)) and
     ((Bar[a].First<>Bar[c].First) or (Bar[a].First<>Bar[c].Second)) then
    Pass1 := True;
  if ((Bar[a].Second<>Bar[b].First) or (Bar[a].Second<>Bar[b].Second)) and
     ((Bar[a].Second<>Bar[c].First) or (Bar[a].Second<>Bar[c].Second)) then
    Pass2 := True;
  if (Pass1) or (Pass2) then
    JointVenture := False;
end;

function SectionActive(X: byte) : boolean;
var
  i,j,k,m : byte;
  Min1,Max1 : byte;
begin
  i:=0; j:=0; m:=0;
  for k:=1 to AccBar do
  begin
    Min1 := Minode(Bar[k].First,Bar[k].Second);
    Max1 := Manode(Bar[k].First,Bar[k].Second);
    if ((Node[Min1].PosX = Node[Sort[X]].PosX) or
        (Node[Max1].PosX = Node[Sort[X+1]].PosX)) then
      if (Node[Min1].PosX<>Node[Max1].PosX) then
      begin
        if (Bar[k].Done = False) then
        begin
          Inc(i);
          MemPassI[i] := k;
        end
        else
        begin
          Inc(j);
          MemPassII[j] := k;
        end;
        Inc(m);
        MemPass[m]:=k;
      end;
  end;
  MemPass[0]:= m;
```

```
  MemPassI[0] := i;      {-Correct Line 696 col 8 [i=3] -}
  MemPassII[0] := j;
  if ((i<=3) and (i>0)) and
     (Not JointVenture(MemPassI[1],MemPassI[2],MemPassI[3])) then
    SectionActive := True
    else
      SectionActive := False;
end;

function GetMax_X(m : BarPass): real;
var
  Max_X,Tem: real;
  x: byte;
begin
  Max_X:=0;
  for x:=1 to m[0] do
  begin
    Tem := Min(Node[Bar[m[x]].First].PosX,Node[Bar[m[x]].Second].PosX);
    Max_X := Max(Tem,Max_X);
  end;
  GetMax_X := Max_X;
end;

function LastNode(m: BarPass) : byte;
var
  Last,k : byte;
begin
  k:=1; Last:=0;
  while Temp[k] <=GetMax_X(m) do
  begin
    Last:=Check[k];
    Inc(k);
  end;
  LastNode := Last;
end;

function TakeMoment(m : byte;VecX,VecY : real): real;
var
  MinValue : byte;
begin
  if (Node[Bar[m].First].PosX>Node[Bar[m].Second].PosX) then
    MinValue := Bar[m].Second
    else
      MinValue := Bar[m].First;
  TakeMoment := VecX*Node[MinValue].PosY-VecY*Node[MinValue].PosX;
end;
```

```
function ACC_SF(Code : Legal;VecX,VecY : ForceVector): real;
var
  AccLoad: real;
  Last,k: byte;
begin
  AccLoad:=0; k:=1;
  Last := LastNode(MemPass);
  case code of
    RX : begin
          while(Check[k]<>Last) do
          begin
            AccLoad := AccLoad+Load[Check[k]].SizeX;
            Inc(k);
          end;
          AccLoad := AccLoad+Load[Last].SizeX;
          for k:=1 to 3 do
            if (Node[Bound[k].node].PosX <= Node[Last].PosX) and (Bound[k].Dir=0) then
              AccLoad := AccLoad+Bound[k].Reac;
          for k:=1 to MemPassII[0] do
            AccLoad := AccLoad+Bar[MemPassII[k]].Force*VecX[k];
          ACC_SF := Negative(AccLoad);
         end;
    RY : begin
          while(Check[k]<>Last) do
          begin
            AccLoad := AccLoad+Load[Check[k]].SizeY;
            Inc(k);
          end;
          AccLoad := AccLoad+Load[Last].SizeY;
          for k:=1 to 3 do
            if (Node[Bound[k].node].PosX <= Node[Last].PosX) and
               (Bound[k].Dir=1) then
              AccLoad := AccLoad+Bound[k].Reac;
          for k:=1 to MemPassII[0] do
            AccLoad := AccLoad+Bar[MemPassII[k]].Force*VecY[k];
          ACC_SF := Negative(AccLoad);
         end;
    RZ : begin
          while (Check[k]<>Last) do
          begin
            AccLoad := AccLoad-Load[Check[k]].SizeX*(Node[Check[k]].PosY)+
                       Load[Check[k]].SizeY*(Node[Check[k]].PosX) ;
            Inc(k);
          end;
          AccLoad := AccLoad-(Load[Last].SizeX*Node[Last].PosY)+
```

```
                (Load[Last].SizeY*Node[Last].PosX);
        for k:=1 to 3 do
          if (Node[Bound[k].Node].PosX <= Node[Last].PosX) then
            case Bound[k].Dir of
              0 : AccLoad := AccLoad-
                    (Bound[k].Reac)*(Node[Bound[k].Node].PosY);
              1 : AccLoad := AccLoad+
                    (Bound[k].Reac)*(Node[Bound[k].Node].PosX);
            end;
        for k:=1 to MemPassII[0] do
          AccLoad := AccLoad+TakeMoment(MemPassII[k],VecX[k],VecY[k]);
        ACC_SF := AccLoad;
      end;
  end;
end;

procedure SetEquation;
var
  k : byte;
begin
  for k:=1 to 3 do
  begin
    a[1,k] := MemVecX[k];
    a[2,k] := MemVecY[k];
    a[3,k] := TakeMoment(MemPassI[k],MemVecX[k],MemVecY[k]);
  end;
  b[1] := ACC_SF(RX,ForceVecX,ForceVecY);
  b[2] := ACC_SF(RY,ForceVecX,ForceVecY);
  b[3] := ACC_SF(RZ,ForceVecX,ForceVecY);
end;

procedure PutForceRes(Active: byte);
var
  j : byte;
begin
  for j:=1 to active do
  begin
    Bar[MemPassI[j]].Force := b[j];
    Bar[MemPassI[j]].Done := True;
  end;
end;

begin
  Sorting;
  SetRange;
  ErrorCode := False;
```

```
 DoneCode :=False;
 for i:=1 to Sort[0] do
 begin
  if SectionActive(i) then
  begin
   DoneCode := True;
   for j:=1 to MemPassI[0] do
   begin
    refNode  :=Minode(Bar[MemPassI[j]].First,Bar[MemPassI[j]].Second);
    IndexNode:=Manode(Bar[MemPassI[j]].First,Bar[MemPassI[j]].second);
    CreatVectorMember(IndexNode,RefNode,j,MemVecX,MemVecY);
   end;
   for j:=1 to MemPassII[0] do
   begin
    RefNode  :=Minode(bar[MemPassII[j]].first,Bar[MemPassII[j]].Second);
    indexNode:=manode(Bar[MemPassII[j]].first,Bar[Mempassl![j]].Second);
    CreatVectorMember(IndexNode,RefNode,j,ForceVecX,ForceVecY);
   end;
   SetEquation;
   if Gauss(3,a,b) then
    PutForceRes(MemPassI[0])
   else
   begin
    ErrorCode := True;
    i := Sort[0];
    OpenWindow(15,8,65,15,1,ErrorColor,1);
    PutString(' The Arragement of member truss is not  ',ErrorColor,2,2);
    PutString(' Property, structure will not stability ',ErrorColor,2,3);
    PutString(' Please check data again.....  ',ErrorColor,2,4);
    Beep(2);
    Wait;
    CloseWindow;
   end;
  end;
 end;
end;


{=========+============================================================}

      {- Vector Analysis with Simultaneous Equation -}

procedure Simultaneous;
const
  ErrorColor = Yellow + (Magenta shl 4);
var
```

```
  IndexNode: byte;
  i,j,k: integer;
  MemVecX, MemVecY, ForceVecX, ForceVecY: ForceVector;
  MemActive,MemNotActive: BarActive;
  Tem: boolean;
  a: cc;
  b: dd;
begin
  if Accbar<=45 then
  begin
    for i:=1 to AccNode*2 do
    begin
      for j:=1 to AccNode*2 do
        a[i,j]:=0;
      b[i]:=0;
    end;
    for k:=1 to Accnode*2 do
    begin
      Tem := JointActive((k+1) div 2,memActive,MemnotActive);
      for j:=1 to MemNotActive[0] do
      begin
        indexNode := CompatibleNode(MemNotActive[j],(k+1) div 2);
        CreatVectorMember(indexNode,(k+1) div 2,j,MemVecX,MemVecY);
      end;
      for i:=1 to MemNotActive[0] do
      begin
        if ((k+1) mod 2) = 0 then
        begin
          a[k,MemNotActive[i]] := MemVecX[i];
          b[k] := Negative(Load[k].SizeX);
        end
        else
        begin
          a[k,MemNotActive[i]] := MemVecY[i];
          b[k] := Negative(Load[k].SizeY);
        end;
      end;
      for i:=1 to 3 do
        if ((k+1) mod 2 = 0) and ((k+1) div 2 = Bound[i].Node) then
          if Bound[i].Dir = 0 then
            a[k,(AccBar+i)] := 1
          else
            a[k+1,(AccBar+i)] := 1
    end;
    Window(1,1,80,25);
    Clrscr;
```

```
    for i:=1 to AccNode*2 do
    begin
      for j:=1 to AccNode*2 do
      begin
        Write(a[i,j]:5:1)
      end;
    end;
    ReadLn;
    if Gauss(AccNode*2,a,b) then
    begin
      for i:=1 to AccBar do
      begin
        Bar[i].Force:=b[i];
        Bar[i].Done:=True;
      end;
      for i:=1 to 3 do
        Bound[i].Reac:=b[AccBar+i];
      PutBoundRes;
    end
  end
  else
  begin
    OpenWindow(15,8,65,15,1,ErrorColor,1);
    PutString('   Structure is Large Complex Truss   ',ErrorColor,2,2);
    PutString(' Method that applied to Execute in this ',ErrorColor,2,3);
    PutString('   program can not execute this case   ',ErrorColor,2,4);
    Beep(2);
    Wait;
    CloseWindow;
  end;
end;

{=============================================================}

              {- Virtual Work Function -}


procedure PutUnitLoad(RefNode : word; Code : Dir);
var
  i : byte;
begin
  SetZeroLoad;
  case Code of
    Hor : Load[RefNode].SizeX := 1;
    Ver : Load[RefNode].SizeY := 1;
  end;
```

```
end;

function VirtualWork : real;
var
  Disp : real;
  k : byte;
begin
  Disp := 0;
  for k:=1 to AccNode do
    Disp := Disp+(SBar[k]*Bar[k].Force*Bar[k].Length)/
            (Bar[k].Area*Bar[k].Modulus);
  VirtualWork :=Disp;
end;

function GetDisp(RefNode : byte;Code : Dir) : real;
var
  Error1, Error2, Done1, Done2: boolean;
begin
  PutUnitLoad(RefNode,Code);
  SetInitial;
  if GetReaction then
  begin
    JointProcess(Error1,Done1);
    SectionProcess(Error2,Done2);
    GetDisp:= VirtualWork;
  end;
end;

{============================================================}

              {- Execute Process Function -}


function ForceExecute: boolean;
var
  Error1, Error2, Done1, Done2: boolean;
begin
  SetInitial;
  ForceExecute := True;
  Done2 := True;
  Error2 := False;
  if GetReaction then
  begin
    while Done2 do
    begin
      Done2 := False;
```

```
      JointProcess(Error1,Done1);
      if (Not Error1) and (Not ProcessComplete) then
        SectionProcess(Error2,Done2);
      if (Error1) or (Error2) or (Not Done2) then
        Done2 := False;
      if (Not Error1) and (Not Error2) then
      begin
        ForceExecute := True;
      end
      else
        ForceExecute := False;
    end;
    if ((Not ProcessComplete) and (Not Error1) and (Not Error2)) then
    begin
      SetInitial;
      Simultaneous;
    end;
  end
  else
    ForceExecute := False;
  TextAttr := 7*16;
  GotoXY(35,8); Write(AccBar:2);
end;

procedure DispExecute;
var
  h,m,s,t: word;
  Start,Finished: real;
  k:byte;
begin
  for k:=1 to AccNode do
  begin
    Node[k].DisX := GetDisp(k,Hor);
    Node[k].DisY := GetDisp(k,Ver);
    TextAttr := 7*16;
    GotoXY(35,9);
    Write(k:2);
    Beep(1);
    if Keypressed then
      if (Readkey=#27) then
      begin
        SetInitZero;
        CloseWindow;
        Exit;
      end;
  end;
```

```
end;

procedure Executed(code: byte);
const
  ExecuteColor = Black + (LightGray shl 4);
var
  k: byte;
  ch: char;
begin
  OpenWindow(15,5,65,22,2,ExecuteColor,1);
  case code of
    1 : begin
          ExecuteBlock(1);
          CurrentStatus(4);
          if ForceExecute then
          begin
            BlockStatus(3,1);
            StatusBar(1);
          end
          else
          begin
            BlockStatus(3,0);
            StatusBar(2);
          end;
        end;
    2 : begin
          ExecuteBlock(2);
          CurrentStatus(1);
          if ForceExecute then
          begin
            CurrentStatus(3);
            BlockStatus(3,1);
            if ProcessComplete then
            begin
              SaveInitLoad;
              CurrentStatus(2);
              DispExecute;
              CurrentStatus(3);
              BlockStatus(4,1);
              StatusBar(1);
              RestoreInitLoad;
            end
          end
          else
          begin
            CurrentStatus(3);
```

```
            BlockStatus(4,0);
            StatusBar(2);
          end;
        end;
    end;
    CloseWindow;
end;

end.
```

```
unit Output;
interface
uses
  Init,Crt,Crt1,Input,Printer,Tool;

procedure Beep(code: byte);
procedure Wait;
procedure StatusBar(code: byte);
procedure ExecuteBlock(code: byte);
procedure BlockStatus(code,cp: byte);
procedure CurrentStatus(code: byte);
procedure Resulting(code: byte);
procedure Printing;


{End of interface Section}


implementation


{=============================================================================}

procedure Beep(code: byte);
begin
  case code of
    1 : begin
          Sound(30);
          Delay(3);
          Nosound;
        end;
    2 : begin
          Sound(220);
          Delay(300);
          Nosound;
        end;
    3: begin
          Sound(800);
          Delay(500);
          Nosound;
        end;
  end;
end;

procedure Wait;
var
```

```
  ch: char;
begin
  ch := Readkey;
  if ch=#0 then
    ch := Readkey;
end;

procedure StatusBar(Code: byte);
const
  NormColor  = White + (Blue shl 4);
  BlinkColor = White + (LightBlue shl 4);
begin
  case code of
    0: PutString('              <ESC>  Cancel'+
               '              ',NormColor,1,16);
    1: begin
         PutString('    Success       : ',NormColor,1,16);
         PutString('    Press Any key     ',BlinkColor,25,16);
         Beep(3);
         Wait;
       end;
    2: begin
         PutString('    Not Success   : ',NormColor,1,16);
         PutString('    Press Any key     ',BlinkColor,25,16);
         Beep(2);
         Wait;
       end;
  end;
end;

procedure ExecuteBlock(code: byte);
const
  BlockColor = Black + (LightGray shl 4);
var
  OldAttr: byte;
begin
  OldAttr := TextAttr;
  TextAttr := BlockColor;
  case code of
    1 : begin
          WriteLn;
          WriteLn('            PC Truss  Execution            ');
          WriteLn('              Member Force                 ');
          WriteLn;
          WriteLn(' Main File : ',UpcaseStr(Filename));
          WriteLn;
```

```
        WriteLn('                    Total    Current ');
        WriteLn(' Member force     :  ',AccBar);
        WriteLn;
        WriteLn;
        WriteLn;
        WriteLn(' Process Result Status');
        WriteLn('   Member Force       : ');
        Write;
      end;
    2 : begin
        WriteLn;
        WriteLn('            PC Truss  Execution              ');
        WriteLn('        Member Force and Displacement        ');
        WriteLn;
        WriteLn(' Main File : ',UpcaseStr(Filename));
        WriteLn;
        WriteLn('                    Total    Current ');
        WriteLn(' Member force     :  ',AccBar);
        WriteLn(' Displacement     :  ',AccNode);
        WriteLn;
        WriteLn;
        WriteLn(' Process Result Status');
        WriteLn('   Member Force       : ');
        Write('   Displacement       : ');
        StatusBar(0);
      end;
  end;
  TextAttr := OldAttr;
end;

procedure BlockStatus(code,cp: byte);
const
  BlockColor = Black + (LightGray shl 4);
begin
  TextAttr := BlockColor;
  case code of
    1 : begin
        GotoXY(30,8);
        Write(cp);
      end;
    2 : begin
        GotoXY(30,9);
        Write(cp);
      end;
    3 : if cp=0 then
        begin
```

```
        GotoXY(28,13);
        Write('Not Complete');
      end
      else
      begin
        GotoXY(28,13);
        Write('Complete');
      end;
   4 : if cp=0 then
      begin
        GotoXY(28,14);
        Write('Not Complete');
      end
      else
      begin
        GotoXY(28,14);
        Write('Complete');
      end;
  end;
end;

procedure CurrentStatus(code: byte);
const
  NormColor  = Black + (lightGray shl 4);
  BlinkColor  = Black + (White shl 4);
var
  OldAttr: byte;
begin
  OldAttr := TextAttr;
  case code of
    1 : begin
        GotoXY(3,8);
        TextAttr := BlinkColor;
        Write('Member force');
        GotoXY(3,9);
        TextAttr := NormColor;
        Write('Displacement');
      end;
    2 : begin
        GotoXY(3,8);
        TextAttr := NormColor;
        Write('Member force');
        GotoXY(3,9);
        TextAttr := BlinkColor;
        Write('Displacement');
      end;
```

```
end;

procedure ForceResult;
const
  ResColor   = Black   + (LightGray shl 4);
  TitleColor = Magenta + (LightGray shl 4);
var
  Start,Stop,Count,i : byte;
  Status : string;
  KeyStroke : char;
  DirCode : boolean;
begin
  Count:=1;
  OpenWindow(5,3,75,23,1,ResColor,1);
  PutString('FORCE RESULTED  <2D-TRUSS SYSTEM>',TitleColor,3,1);
  PutString('<ESC> : EXIT',TitleColor,54,1);
  PutString('ELEM      LENGTH      FORCE      STRESS  ',TitleColor,3,3);
  PutString('-----------------------------------------',TitleColor,3,4);
  TextAttr := ResColor;
  CheckRow(AccBar,14,Count,Start,Stop);
  Window(8,8,70,22);
  for i:=Start to Stop do
  begin
    WriteLn(i:4,Bar[i].Length:12:2,Bar[i].Force:14:2);
    Delay(25);
  end;
  repeat
    KeyStroke:=ReadKey;
    if KeyStroke=#0 then
    begin
      KeyStroke:=ReadKey;
      DirCode:=True;
    end
    else
      DirCode:=False;
    if DirCode then
    begin
      Clrscr;
      CheckPage(AccBar,14,KeyStroke,Count);
      CheckRow(AccBar,14,Count,Start,Stop);
      for i:=Start to Stop do
      begin
        WriteLn(i:4,Bar[i].Length:12:2,Bar[i].Force:14:2);
        Delay(25);
      end;
    end;
```

```
  until (DirCode=False) and (KeyStroke=#27);
  CloseWindow;
end;

procedure DispResult;
const
  ResColor   = Black  + (LightGray shl 4);
  TitleColor = Magenta + (LightGray shl 4);
var
  Start,Stop,Count,i : byte;
  KeyStroke : char;
  DirCode : boolean;
begin
  Count:=1;
  OpenWindow(5,3,75,23,1,ResColor,1);
  PutString('DISPLACEMENT RESULTED  <2D-TRUSS SYSTEM>',TitleColor,3,1);
  PutString('<ESC> : EXIT',TitleColor,54,1);
  PutString('NODE      X-DISP      Y-DISP  ',TitleColor,3,3);
  PutString('--------------------------------',TitleColor,3,4);
  TextAttr := ResColor;
  CheckRow(AccNode,14,Count,Start,Stop);
  Window(8,8,70,22);
  for i:=Start to Stop do
  begin
    WriteLn(i:3,Node[i].DisX:14:4,Node[i].DisY:14:4);
    Delay(25);
  end;
  repeat
    KeyStroke:=ReadKey;
    if KeyStroke=#0 then
    begin
      KeyStroke:=ReadKey;
      DirCode:=True;
    end
    else
      DirCode:=False;
    if DirCode then
    begin
      Clrscr;
      CheckPage(AccNode,14,KeyStroke,Count);
      CheckRow(AccNode,14,Count,Start,Stop);
      for i:=Start to Stop do
      begin
        WriteLn(i:3,Node[i].DisX:14:4,Node[i].DisY:14:4);
        Delay(25);
      end;
```

```
    end;
  until (DirCode=False) and (KeyStroke=#27);
  CloseWindow;
end;

procedure VolumeResult;
const
  VolColor = Black + (LightGray shl  4);
  TitleColor = Magenta + (LightGray shl 4);
var
  Vol: real;
  ch: char;
  OldAttr,i: byte;
begin
  OldAttr := TextAttr;
  Vol := 0;
  for i:=1 to AccBar do
    Vol := Vol+ Bar[i].Area*Bar[i].Length;
  OpenWindow(5,5,75,10,1,VolColor,1);
  TextAttr := VolColor;
  PutString('VOLUMN OF MATERIAL USED  <2D-TRUSS SYSTEM>',TitleColor,3,1);
  PutString('<ESC> : EXIT',TitleColor,54,1);
  GotoXY(3,3);
  Write('This Truss Used Volumn of Material = ',Vol:6:2);
  repeat
    ch := Readkey
  until ch=#27;
  TextAttr := OldAttr;
  CloseWindow;
end;

procedure Resulting(code: byte);
begin
  case code of
    1 : ReactionResult;
    2 : ForceResult;
    3 : DispResult;
    4 : VolumeResult;
  end;
end;

procedure WaitPaper(code: byte);
const
  Color = Black + (LightGray shl 4);
begin
  Clrscr;
```

```
 PutString(' Adjust New Paper Property <Y/N> ? ',Color,1,2);
 Wait;
 Clrscr;
 case code of
   1 : PutString(' Now Printing Node Data',Color,4,2);
   2 : PutString(' Now Printing Element Data',Color,4,2);
   3 : PutString(' Now Printing Load Data',Color,4,2);
   4 : PutString(' Now Printing Reaction Result',Color,2,2);
   5 : PutString(' Now Printing Force Result',Color,2,2);
   6 : PutString(' Now Printing Displacement Result',Color,2,2);
 end;
end;

procedure PrintNode;
var
  i: integer;
begin
  WaitPaper(1);
  WriteLn(Lst,'  ************************************************');
  WriteLn(Lst,'          COORDINATE DATA <2D-TRUSS SYSTEM>');
  WriteLn(Lst,'  ************************************************');
  WriteLn(Lst,'          NODE        X-COOR       Y-COOR');
  if AccNode<=50 then
    for i:=1 to AccNode do
      WriteLn(Lst,i:15,Node[i].PosX:14:2,Node[i].PosY:10:2);
  if (AccNode>50) and (AccNode<=100) then
  begin
    for i:=1 to 50 do
      WriteLn(Lst,i:15,Node[i].PosX:14:2,Node[i].PosY:10:2);
    WaitPaper(1);
    for i:=51 to AccNode do
      WriteLn(Lst,i:15,Node[i].PosX:14:2,Node[i].PosY:10:2);
  end;
  if (AccNode>100) and (AccNode<=150) then
  begin
    for i:=1 to 50 do
      WriteLn(Lst,i:15,Node[i].PosX:14:2,Node[i].PosY:10:2);
    WaitPaper(1);
    for i:=51 to 100 do
      WriteLn(Lst,i:15,Node[i].PosX:14:2,Node[i].PosY:10:2);
    WaitPaper(1);
    for i:=101 to AccNode do
      WriteLn(Lst,i:15,Node[i].PosX:14:2,Node[i].PosY:10:2);
  end;
 end;
```

```
procedure PrintElement;
var
  i: integer;
begin
  if (AccBar <= 50) then
  begin
    WaitPaper(2);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        1-NODE      2-NODE');
    for i:=1 to AccNode do
      WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  end;
  if (AccBar > 50) and (AccBar<=100) then
  begin
    WaitPaper(2);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        1-NODE      2-NODE');
    for i:=1 to 50 do
      WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
    WaitPaper(2);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        1-NODE      2-NODE');
    for i:=51 to 100 do
      WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  end;
  if (AccBar > 101) and (AccBar<=150) then
  begin
    WaitPaper(2);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        1-NODE      2-NODE');
    for i:=1 to 50 do
      WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
    WaitPaper(2);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        1-NODE      2-NODE');
    for i:=51 to 100 do
```

```
  WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 WaitPaper(2);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM         1-NODE       2-NODE');
 for i:=101 to AccNode do
  WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
end;
if (AccBar > 151) and (AccBar<=200) then
begin
 WaitPaper(2);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM         1-NODE       2-NODE');
 for i:=1 to 50 do
  WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 WaitPaper(2);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM         1-NODE       2-NODE');
 for i:=51 to 100 do
  WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 WaitPaper(2);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM         1-NODE       2-NODE');
 for i:=101 to 150 do
  WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 WaitPaper(2);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM         1-NODE       2-NODE');
 for i:=151 to 200 do
  WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
end;
if (AccBar>151) and (AccBar<=200) then
begin
 WaitPaper(2);
 for i:=1 to 50 do
  WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 WaitPaper(2);
```

```
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'           ELEM          1-NODE        2-NODE');
  for i:=51 to 100 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  WaitPaper(2);
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'           ELEM          1-NODE        2-NODE');
  for i:=101 to 150 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  WaitPaper(2);
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'           ELEM          1-NODE        2-NODE');
  for i:=151 to 200 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  WaitPaper(2);
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'          ELEMENT  DATA    <2D-TRUSS SYSTEM>');
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'           ELEM          1-NODE        2-NODE');
  for i:=201 to 250 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 end;
end;

procedure PrintLoad;
type
 LoadList= array[1..125] of integer;
var
 ListLoad: LoadList;
 AccLoad,i: integer;

procedure SortLoad(var AccLoad: integer;var ListLoad: LoadList);
var
 i,j: byte;
begin
 AccLoad := 0; j:=0;
 for i:=1 to AccNode do
 begin
  if (Load[i].SizeX<>0) or (Load[i].SizeY<>0) then
  begin
```

```
      Inc(AccLoad);
      Inc(j);
      ListLoad[j] := i;
    end;
  end;
end;

begin
  SortLoad(AccLoad,ListLoad);
  if AccLoad<=50 then
  begin
    WaitPaper(3);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          LOAD DATA       <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'            NODE         X-COOR       Y-COOR');
    for i:=1 to 50 do
      WriteLn(ListLoad[i]:5,Load[ListLoad[i]].SizeX:14:2,Load[ListLoad[i]].SizeY:10:2);
  end;
  if (AccLoad>51) and (AccLoad<=100) then
  begin
    WaitPaper(3);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          LOAD DATA       <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'            NODE         X-COOR       Y-COOR');
    for i:=1 to 50 do
      WriteLn(ListLoad[i]:5,Load[ListLoad[i]].SizeX:14:2,Load[ListLoad[i]].SizeY:10:2);
    WaitPaper(3);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          LOAD DATA       <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'            NODE         X-COOR       Y-COOR');
    for i:=51 to AccLoad do
      WriteLn(ListLoad[i]:5,Load[ListLoad[i]].SizeX:14:2,Load[ListLoad[i]].SizeY:10:2);
  end;
  if (AccLoad<101) and (AccLoad<=150) then
  begin
    WaitPaper(3);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          LOAD DATA       <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'            NODE         X-COOR       Y-COOR');
    for i:=1 to 50 do
      WriteLn(ListLoad[i]:5,Load[ListLoad[i]].SizeX:14:2,Load[ListLoad[i]].SizeY:10:2);
    WaitPaper(3);
```

```
procedure PrintForce;
var
  i: integer;
begin
  if (AccBar <= 50) then
  begin
    WaitPaper(2);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          FORCE  RESULTED   <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        LENGTH       FORCE');
    for i:=1 to AccNode do
      WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  end;
  if (AccBar > 50) and (AccBar<=100) then
  begin
    WaitPaper(2);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          FORCE  RESULTED   <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        LENGTH       FORCE');
    for i:=1 to 50 do
      WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
    WaitPaper(5);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          FORCE  RESULTED   <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        LENGTH       FORCE');
    for i:=51 to 100 do
      WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  end;
  if (AccBar > 101) and (AccBar<=150) then
  begin
    WaitPaper(5);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          FORCE  RESULTED   <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        LENGTH       FORCE');
    for i:=1 to 50 do
      WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
    WaitPaper(5);
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'          FORCE  RESULTED   <2D-TRUSS SYSTEM>');
    WriteLn(Lst,'  ***************************************************');
    WriteLn(Lst,'           ELEM        LENGTH       FORCE');
```

```
 for i:=51 to 100 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 WaitPaper(5);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          FORCE  RESULTED    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM          LENGTH        FORCE');
 for i:=101 to AccNode do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
end;
if (AccBar > 151) and (AccBar<=200) then
begin
 WaitPaper(2);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          FORCE  RESULTED    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM          LENGTH        FORCE');
 for i:=1 to 50 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 WaitPaper(5);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          FORCE  RESULTED    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM          LENGTH        FORCE');
 for i:=51 to 100 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 WaitPaper(5);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          FORCE  RESULTED    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM          LENGTH        FORCE');
 for i:=101 to 150 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 WaitPaper(5);
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'          FORCE  RESULTED    <2D-TRUSS SYSTEM>');
 WriteLn(Lst,'  ***************************************************');
 WriteLn(Lst,'           ELEM          LENGTH        FORCE');
 for i:=151 to 200 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
end;
if (AccBar>151) and (AccBar<=200) then
begin
 WaitPaper(2);
 for i:=1 to 50 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
```

```
  WaitPaper(5);
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'          FORCE RESULTED    <2D-TRUSS SYSTEM>');
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'           ELEM        LENGTH       FORCE');
  for i:=51 to 100 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  WaitPaper(5);
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'          FORCE RESULTED    <2D-TRUSS SYSTEM>');
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'           ELEM        LENGTH       FORCE');
  for i:=101 to 150 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  WaitPaper(5);
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'          FORCE RESULTED    <2D-TRUSS SYSTEM>');
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'           ELEM        LENGTH       FORCE');
  for i:=151 to 200 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
  WaitPaper(5);
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'          FORCE RESULTED    <2D-TRUSS SYSTEM>');
  WriteLn(Lst,'  ***************************************************');
  WriteLn(Lst,'           ELEM        LENGTH       FORCE');
  for i:=201 to 250 do
   WriteLn(Lst,i:5,Bar[i].Length:14:2,Bar[i].First:10,Bar[i].Second:10);
 end;
end;

procedure BlockPrinted;
type
 Head = array[1..2,1..3] of string;
const
 Color     = Black + (LightGray shl 4);
 BlockColor = White + (Cyan shl 4);
 BarColor   = White + (Blue shl 4);
 Headed : Head = (('  Node Data    ',
                   ' Element Data ',
                   ' Load Data    '),
                  ('  Reaction     ',
                   ' Member Force ',
                   ' Displacement '
                   ));
var
```

```
  cn1,cn2,col,row,cd: byte;
  ch: char;
  Doned,Cr: boolean;
begin
  PutString(' Printing ',BlockColor,20,1);
  PutString(' Print Data   ',BlockColor,5,3);
  PutString(' Print Result ',BlockColor,28,3);
  Doned := False;
  col:=1; row:=1;
  repeat
    for cn1:=1 to 2 do
      for cn2:=1 to 3 do
        PutString(Headed[cn1,cn2],BlockColor,5+24*(cn1-1),4+cn2);
    PutString(Headed[col,row],BarColor,5+24*(col-1),4+row);
    PutString('    <Esc>: Exit          <Enter>: Select',BlockColor,2,11);
    ReadFunckey(ch);
    if (FuncKey) then
    case ch of
      #72 : begin
            if (row=2) or (row=3) then
              Dec(row)
            else
            begin
              if col=1 then
              begin
                col:=2;
                row:=3;
              end
              else
              begin
                col:=1;
                row:=3;
              end;
            end;
          end;
      #80 : begin
            if (row=1) or (row=2) then
              Inc(row)
            else
            begin
              if col=1 then
              begin
                col:=2;
                row:=1;
              end
              else
```

```
          begin
            col:=1;
            row:=1;
          end;
        end;
      end;
  #75 : begin
        if col=1 then
          col:=2
        else
          col:=1;
      end;
  #77 : begin
        if col=2 then
          col:=1
        else
          col:=2;
      end;
end
else
begin
  if ch=#27 then
  begin
    Doned := True;
    cd:=0;
    CloseWindow;
  end;
  if ch=#13 then
  begin
    if col =1 then
      cd:=row
    else
      cd:=row+3;
    Cr:=True;
  end;
end;
if Cr then
begin
  OpenWindow(25,10,55,14,1,Color,1);
  case cd of
    1 : PrintNode;
    2 : PrintElement;
    3 : PrintLoad;
    4 : PrintReaction;
    5 : PrintForce;
    6 : PrintDisp;
```

```
      end;
      CloseWindow;
      Cr:=False;
    end;
  until Doned;
end;


procedure Printing;
const
  BlockColor = White + (Cyan shl 4);
var
  cd: byte;
begin
  OpenWindow(15,7,65,20,1,BlockColor,1);
  BlockPrinted;
end;

end. {of unit}
```

```
unit Info;
interface
uses
  Crt,Crt1,Init,Tool,Dos;

procedure About;
procedure GetLegal(var Ok: boolean);
procedure Directory;
procedure Os_Shell;
function  BlockFile(var s: string): boolean;
function  FileExist(F: string): boolean;
function  FileList(F1,F2,F3,F4,F5: string): boolean;
function  FileExtend(s1,s2: string): string;
procedure GroupExtend(var s1,s2,s3,s4,s5: string);
procedure FileNotFound;
function  QuitStatus: boolean;

implementation

procedure About;
const
  AboutColor = Yellow + (Cyan shl 4);
  CharColor  = White  + (Cyan shl 4);
var
 ch: char;
begin
  OpenWindow(20,6,60,18,2,AboutColor,1);
  CursorOff;
  PutString('          Turbo  PC-Truss          ',CharColor,1,2);
  PutString('            version 1.0            ',CharColor,1,3);
  PutString('    Special Project 4th(Year)  by    ',CharColor,1,5);
  PutString('      Sincharoen  Tangkhantikul      ',CharColor,1,7);
  PutString('      Akarat      Sritan             ',CharColor,1,8);
  PutString(' Department Civil Engineering  KMITL ',CharColor,1,10);
  ch:=Readkey;
  CursorOn;
  CloseWindow;
end;

function BlockFile(var s: string): boolean;
const
  BlockColor = Black + (LightGray shl 4);
var
  OldAttr: byte;
  aa: boolean;
begin
```

```
  OldAttr := TextAttr;
  CursorOn;
  OpenWindow(15,6,55,8,1,BlockColor,1);
  TextAttr := BlockColor;
  GotoXY(3,1); Write('Enter Filename : ');
  ReadString(20,1,aa,s);
  CloseWindow;
  CursorOff;
  BlockFile := aa;
  TextAttr := OldAttr;
end;

function FileExist(F: string): Boolean;
var
  Sr: SearchRec;
begin
  FindFirst(F,AnyFile,Sr);
  FileExist := DosError = 0;
end;

function FileList(F1,F2,F3,F4,F5: string): boolean;
begin
  if (FileExist(F1)) and (FileExist(F2)) and (FileExist(F3)) and
     (FileExist(F4)) and (FileExist(F5)) then
    FileList := True
  else
    FileList := False;
end;

function  FileExtend(s1,s2: string): string;
var
  post: integer;
begin
  post := Length(s1)-3;
  if s1[post]='.' then
  begin
    s1 := Copy(s1,1,post-1);
    s1 := Concat(s1,'.',s2);
  end
  else
    s1 := Concat(s1,'.',s2);
  FileExtend := s1;
end;

procedure GroupExtend(var s1,s2,s3,s4,s5: string);
  begin
```

```
  s1 := FileExtend(s1,'dat');
  s2 := FileExtend(s2,'coo');
  s3 := FileExtend(s3,'bou');
  s4 := FileExtend(s4,'elm');
  s5 := FileExtend(s5,'npl');
end;

procedure FileNotFound;
const
  BlockColor = White  + (Red shl 4);
  EscColor   = Yellow + (Red shl 4);
var
  ch: char;
begin
  OpenWindow(25,6,55,8,1,BlockColor,1);
  PutString(' File Not Found. Press ',BlockColor,2,1);
  PutString('ESC ',EscColor,25,1);
  repeat
    ch := ReadKey;
  until ch = #27;
  CloseWindow;
end;

function QuitStatus:boolean;
const
  QuitColor = Yellow +(LightGray shl 4);
var
  ch:char;
begin
  CurSorOff;
  OpenWindow(28,12,52,14,1,QuitColor,1);
  repeat
    PutString(' Are you sure <Y/N>? ',QuitColor,2,1);
    ch:=Upcase(Readkey);
  until (ch='Y') or (ch='N');
  CloseWindow;
  if ch='Y' then
    QuitStatus:=True
  else
    QuitStatus:=False;
  CursorOn;
end;

procedure Directory;
const
  DirectColor = White   + (Cyan shl 4);
```

```
  ErrorColor  = Yellow   + (Magenta shl 4);
  BlankColor  = LightCyan + (Blue shl 4);
var
  D1,D2: String;
  i: byte;
  ch: char;
  Done: boolean;
  OldAttr: byte;
begin
  OldAttr := TextAttr;
  OpenWindow(15,8,65,10,1,DirectColor,1);
  Window(16,9,62,9);
  GetDir(0,D1);
  D2:=UpcaseStr(D1);
  CursorOn;
  TextAttr := DirectColor;
  Write('Enter Data Directory : ');
  TextAttr := BlankColor;
  GotoXY(24,1);  ClrEol;
  ReadString(WhereX,WhereY,Done,D2);
  CursorOff;
  CloseWindow;
  if (Done) and (D2<>#27) then
  begin
    {$I-} ChDir(D2); {$I+}
    if IoResult <> 0 then
    begin
      OpenWindow(24,8,56,10,1,ErrorColor,1);
      TextAttr := ErrorColor;
      Write('   Directory Not Found     '+#7);
      ChDir(D1);
      ch:=ReadKey;
      CloseWindow;
    end
    else
    begin
      OpenWindow(24,8,56,10,1,DirectColor,1);
      TextAttr := DirectColor;
      Write(' Now Directory Changed Already'+#7);
      ch:=ReadKey;
      CloseWindow;
    end;
  end;
  TextAttr := OldAttr;
end;
```

```
procedure Os_Shell;begin
 Window(1,1,80,25);
 TextAttr:=7; Clrscr;
 CursorOn;
 WriteLn(' Type EXIT to return to PC Truss...');
 SwapVectors;
 Exec(GetEnv('COMSPEC'), '');
 SwapVectors;
 if DosError <> 0 then
  WriteLn('Could not execute COMMAND.COM');
 WriteLn;
 ColorScreen(1,2,80,24,16+7);
 DisplayMainMenu;
 CursorOff;
end;

procedure GetLegal(var ok: boolean);
const
 LegalColor  = LightCyan+ (Blue shl 4);
 CharColor   = Black    + (Cyan shl 4);
 NormColor   = Black    + (LightGrey shl 4);
 BottonColor = White    + (Green shl 4);
 Shadow      = Black    + (Cyan shl 4);
var
 OldAttr,Select: byte;
 Okk: boolean;
 Done: boolean;
 ch: char;

 procedure LegalIn;
 var
  afi: Text;
 begin
  Assign(afi,FileExtend(Filename,'dat'));
  Rewrite(afi);
  WriteLn(afi,Title);
  WriteLn(afi,LUnit);
  WriteLn(afi,FUnit);
  WriteLn(afi,Engineer);
  Close(afi);
 end;

 procedure DrawBotton;
 begin
  PutString(' <OK> ',NormColor,10,13);
  PutString('',Shadow,18,13);
```

```
  PutString('ßßßßßßßß',Shadow,11,14);
  PutString(' AGAIN ',NormColor,23,13);
  PutString('',Shadow,30,13);
  PutString('ßßßßßßß',Shadow,24,14);
  PutString(' CANCEL ',NormColor,35,13);
  PutString('',Shadow,43,13);
  PutString('ßßßßßßßß',Shadow,36,14);
end;

procedure SelectBotton(code: byte);
begin
  case code of
    1: begin
         PutString(' <OK> ',BottonColor,10,13);
         PutString('',Shadow,18,14);
         PutString('ßßßßßßßß',Shadow,11,14);
       end;
    2: begin
         PutString(' AGAIN ',BottonColor,23,13);
         PutString('',Shadow,30,13);
         PutString('ßßßßßßß',Shadow,24,14);
       end;
    3: begin
         PutString(' CANCEL ',BottonColor,35,13);
         PutString('',Shadow,43,13);
         PutString('ßßßßßßßß',Shadow,36,14);
       end;
  end;
end;

begin
  OldAttr := TextAttr;
  OpenWindow(15,6,65,21,2,CharColor,1);
  Window(16,7,60,19);
  TextAttr := CharColor;
  GotoXY(5,2); Write('Master Filename : ');
  GotoXY(5,4); Write(' Project Title : ');
  GotoXY(5,6); Write('  Length Unit : ');
  GotoXY(5,8); Write('   Force Unit : ');
  GotoXY(5,10); Write('     Engineer : ');
  TextAttr := LegalColor;
  GotoXY(24,2); ClrEol;
  GotoXY(24,4); ClrEol;
  GotoXY(24,6); ClrEol;
  GotoXY(24,8); ClrEol;
  GotoXY(24,10); ClrEol;
```

```
DrawBotton;
repeat
 CursorOn;
 repeat
  ReadString(24,2,Okk,Filename);
 until (Length(Filename)<=8) and (Filename[1]<>#27);
 ReadString(24,4,Okk,Title);
 ReadString(24,6,Okk,LUnit);
 ReadString(24,8,Okk,FUnit);
 ReadString(24,10,Okk,Engineer);
 Select := 1;
 Done := False;
 CursorOff;
 repeat
  DrawBotton;
  SelectBotton(Select);
  ReadFuncKey(ch);
  if FuncKey then
   case ch of
    #77 : if Select<3 then
            Inc(Select)
          else
           Select := 1;
    #75 : if Select>1 then
            Dec(Select)
          else
           Select := 3;
   end
   else
    if (ch=#13) then
     Done:=True;
   DrawBotton;
   SelectBotton(Select)
  until Done;
 until (Select=1) or (Select=3);
 if (Select=1) then
 begin
  Ok := True;
  LegalIn;
 end
 else
  Ok := False;
 CloseWindow;
 TextAttr := OldAttr;
end;
```

end. {of unit}

```
implementation

var
  regs: registers;

procedure XYtoRC(x, y: byte; var r, c: byte);
var
  x1,y1: byte;
begin
  x1 := Lo(WindMin);
  y1 := Hi(WindMin);
  r  := y1+y;
  c  := x1+x;
end;

procedure PutChar(ch: char; attr, x, y: byte);
var
  r,c: byte;
begin
  XYtoRC(x,y,r,c);
  ScreenPtr^[r,c] := Ord(ch)+attr*$100;
end;

procedure PutString(s: string; attr, x, y: byte);
var
  r,c,i,k: byte;
begin
  XYtoRC(x,y,r,c);
  i := 0;
  for k:=c to (c+Length(S)-1) do
  begin
    Inc(i);
    ScreenPtr^[r,k] := Ord(s[i])+attr*$100;
  end;
end;

function GetChar(x,y: byte): char;
var
  r,c: byte;
begin
  XYtoRc(x,y,r,c);
  GetChar := chr(ScreenPtr^[r,c]-TextAttr*$100);
end;

procedure SetCursor(top, bottom: byte);
```

```
begin
  Regs.AH := 1;
  Regs.CH := top;
  Regs.CL := bottom;
  intr($10,Regs);
end;

procedure CursorOn;
begin
  Regs.AH := 1;
  Regs.CH := 12;
  Regs.CL := 13;
  intr($10,regs);
end;

procedure CursorOff;
begin
  Regs.AH := 1;
  Regs.CH := 20;
  Regs.CL := 0;
  intr($10,regs);
end;

procedure ClearScreen(x1, y1, x2, y2: byte);
begin
  Window(x1, y1, x2, y2);
  TextAttr := 7;
  Clrscr;
end;

procedure BlockScreen(x1, y1, x2, y2, attr: byte);
var
  col,row: byte;
begin
  for col:=x1 to x2 do
    for row:=y1 to y2 do
      putchar(#219,Attr,col,row);
end;

procedure ColorScreen(x1, y1, x2, y2, attr: byte);
var
  xx1,yy1,xx2,yy2: byte;
begin
  xx1:=Lo(WindMin);
  yy1:=Hi(WindMin);
  xx2:=Lo(WindMax);
```

```
  yy2:=Hi(WindMax);
  Window(x1,y1,x2,y2);
  TextAttr:=attr;
  Clrscr;
  Window(xx1,yy1,xx2,yy2);
end;

procedure CenterStr(s: string; y1, attr: byte);
var
  x1,col,row: byte;
begin
  x1 := ((Lo(WindMax)-Lo(WindMin))-Length(s)) div 2 +1;
  PutString(s,attr,x1,y1);
end;

function UpcaseStr(s: string): string;
var
  i: integer;
begin
  for i:=1 to Length(s) do
    s[i] := Upcase(s[i]);
  UpcaseStr:=s;
end;

procedure WindowBox(x1, y1, x2, y2, style: byte);
const
  BoxStyle : array[1..3,1..8] of char =
          ((#218,#191,#192,#217,#196,#196,#179,#179),
           (#201,#187,#200,#188,#205,#205,#186,#186),
           (#219,#219,#219,#219,#223,#220,#219,#219));
var
  col,row,attr : byte;
begin
  attr := TextAttr;
  PutChar(BoxStyle[style,1],attr,x1,y1);
  PutChar(BoxStyle[style,2],attr,x2,y1);
  PutChar(BoxStyle[style,3],attr,x1,y2);
  PutChar(BoxStyle[style,4],attr,x2,y2);
  for col:=x1+1 to x2-1 do
  begin
    PutChar(BoxStyle[style,5],attr,col,y1);
    PutChar(BoxStyle[style,6],attr,col,y2);
  end;
  for row:=y1+1 to y2-1 do
  begin
    PutChar(BoxStyle[style,7],attr,x1,row);
```

```
    PutChar(BoxStyle[style,8],attr,x2,row);
  end;
end;

procedure WindowFrame(x1, y1, x2, y2, style, winAttr, code: byte);
var
  OldAttr,i : byte;
begin
  OldAttr := TextAttr;
  Window(x1,y1,x2,y2);
  TextAttr := WinAttr;
  Clrscr;
  Window(1,1,80,25);
  WindowBox(x1,y1,x2,y2,style);
  case code of
    1: begin
         for i:=x1+2 to x2+2 do
           PutChar(GetChar(i,y2+1),8,i,y2+1);
         for i:=y1+1 to y2+1 do
         begin
           PutChar(Getchar(x2+1,i),8,x2+1,i);
           PutChar(GetChar(x2+2,i),8,x2+2,i);
         end;
       end;
  end;
  TextAttr:=OldAttr;
  Window(x1+1,y1+1,x2-1,y2-1);
end;

procedure OpenWindow(x1,y1,x2,y2,style,winattr,code: byte);
var
  Block: WindowLink;
  LineLength,WindowSize,I: integer;
  row: byte;
begin
  case code of
    0 : begin
          LineLength := x2-x1+1;
          WindowSize := LineLength*(y2-y1+1)*2+FixedSize;
          if (x2>80) or (y2>25)  or (x2-x1<2) or (y2-y1<2) then
            ErrorWindow:=1
          else if (abs(MemAvail) < WindowSize) then
            ErrorWindow:=2;
          if ErrorWindow = 0 then
          begin
            GetMem(Block,WindowSize);
```

```
    Block^.X1 := x1;
    Block^.X2 := x2;
    Block^.Y1 := y1;
    Block^.Y2 := y2;
    Block^.X  := WhereX;
    Block^.Y  := WhereY;
    Block^.BackLink := ActiveWindow;
    ActiveWindow := Block;
    Inc(WindowCount);
    Block^.ID := WindowCount;
    I:=1;
    for row:=y1 to y2 do
    begin
      Move(ScreenPtr^[Row,x1],Block^.ScreenContents[I],LineLength*2);
      I := I+LineLength;
    end;
  end;
end;
1 : begin
    LineLength := x2-x1+3;
    WindowSize := LineLength*(y2-y1+2)*2+FixedSize;
    if (x2>78) or (y2>24)  or (x2-x1<2) or (y2-y1<2) then
      ErrorWindow:=1
    else if (abs(MemAvail) < WindowSize) then
      ErrorWindow:=2;
    if ErrorWindow = 0 then
    begin
      GetMem(Block,WindowSize);
      Block^.X1 := x1;
      Block^.X2 := x2+2;
      Block^.Y1 := y1;
      Block^.Y2 := y2+1;
      Block^.X  := WhereX;
      Block^.Y  := WhereY;
      Block^.BackLink := ActiveWindow;
      ActiveWindow := Block;
      Inc(WindowCount);
      Block^.ID := WindowCount;
      I:=1;
      for Row:=y1 to y2+1 do
      begin
        Move(ScreenPtr^[Row,x1],Block^.ScreenContents[I],LineLength*2);
        I := I+LineLength;
      end;
    end;
  end;
```

```
  end;
  WindowFrame(x1,y1,x2,y2,style,winAttr,code);
end;

procedure CloseWindow;
var
  Block: WindowLink;
  LineLength,WindowSize,I: integer;
  row: byte;
begin
  if ActiveWindow <> nil then
  begin
    Block := ActiveWindow;
    LineLength := Block^.X2-Block^.X1+1;
    WindowSize := LineLength*(Block^.Y2-Block^.Y1+1)*2+FixedSize;
    Dec(WindowCount);
    I:=1;
    for row:=Block^.Y1 to Block^.Y2 do
    begin
      Move(Block^.ScreenContents[I],ScreenPtr^[Row,Block^.X1],LineLength*2);
      I:=I+LineLength;
    end;
    ActiveWindow := Block^.BackLink;
    if ActiveWindow = nil then
      Window(1,1,80,25)
    else
      with ActiveWindow^ do
        Window(x1+1,y1+1,x2-1,y2-1);
    GotoXY(Block^.X,Block^.Y);
    FreeMem(Block,WindowSize);
  end;
end;

procedure IdentifyCrt;
begin
  case CrtType of
    0..3 : VideoSeg := ColorSeg;
       7 : VideoSeg := MonoSeg;
  end;
end;

procedure Initial;
begin
  IdentifyCrt;
  ActiveWindow := nil;
  FixedSize := Sizeof(WindowControlBlock)-Sizeof(ScreenBlock);
```

```
unit Crt1;
interface
uses
  Crt,Dos;
type
  ScreenArray = array[1..25,1..80] of integer;
  ScreenBlock = array[1..2000] of integer;
  WindowLink  = ^WindowControlBlock;
  WindowControlBlock = record
    x1,y1,x2,y2: integer;
    x,y: integer;
    ID: byte;
    BackLink: WindowLink;
    ScreenContents: ScreenBlock;
  end;
const
  ColorSeg = $B800;
  MonoSeg = $B000;
var
  VideoSeg: word;
  CrtType: byte Absolute $0040:$0049;
  CursorMode: word Absolute $0040:$0060;
  Vport: word Absolute $0040:$0063;
  ScreenPtr: ^ScreenArray;
  ActiveWindow: WindowLink;
  FixedSize: integer;
  WindowCount: byte;
  ErrorWindow: byte;

procedure XYtoRC(x, y: byte; var r, c: byte);
function  GetChar(x,y: byte): char;
procedure PutChar(ch: char; attr, x, y: byte);
procedure PutString(s: string; attr, x, y: byte);
procedure SetCursor(top, bottom: byte);
procedure CursorOn;
procedure CursorOff;
procedure ClearScreen(x1, y1, x2, y2: byte);
procedure BlockScreen(x1, y1, x2, y2, attr: byte);
procedure ColorScreen(x1, y1, x2, y2, attr: byte);
procedure CenterStr(s: string; y1, attr: byte);
function  UpcaseStr(s: string): string;
procedure WindowBox(x1, y1, x2, y2, style: byte);
procedure WindowFrame(x1, y1, x2, y2, style, winattr, code: byte);
procedure OpenWindow(x1, y1, x2, y2, style, winattr, code: byte);
procedure CloseWindow;
```

```
  ScreenPtr := Ptr(VideoSeg,0);
  Window(1,1,80,25);
  WindowCount:=0;
  ErrorWindow := 0;
end;

begin
  Initial;
end.
```

```
unit Tool;
interface
uses
  Crt,Crt1;
var
  Key: char;
  FuncKey,QuitMenu,EScPress: boolean;
  CurrentChoice: byte;
const
  Return_Key = #13;
  Esc_Key    = #27;
  F1  = #59;  F2  = #60;
  F3  = #61;  F4  = #62;
  F5  = #63;  F6  = #64;
  F7  = #65;  F8  = #66;
  F9  = #67;  F10 = #68;
  Home_Key = #71;  Up_Key   = #72;
  PgUp_Key = #73;  Lt_Key   = #75;
  Rt_Key   = #77;  End_Key  = #79;
  Dn_Key   = #80;  PgDn_Key = #81;
  Ins_Key  = #82;  Del_Key  = #83;

const
  MaxChoice = 10;
  MaxMenu = 6;

type
  String14  = String[14];
  String40  = String[40];
  StructureMenu = record
    Win : array[1..4] of byte;
    Col : array[0..MaxChoice] of byte;
    Row : array[0..MaxChoice] of byte;
    Msg : array[0..MaxChoice] of String40;
    LastChoice : byte;
  end;
  StrucMenu = array[1..MaxMenu] of StructureMenu;
  FuncMenu = record
    Col : array[1..7] of byte;
    Row : array[1..7] of byte;
    Msg : array[1..7] of String14;
  end;

{---------------------------- Menu  Messages ---------------------------}
```

```
const
  Menu : StrucMenu=(
            (Win :(03,02,20,10);
             Col :(04,01,01,01,01,01,01,01,01,01,01);
             Row :(01,01,02,03,04,05,06,07,08,09,10);
             Msg :(' File ',
                 ' Open       F2 ',
                 ' Save       F3 ',
                 ' New          ',
                 ' Directory  F5 ',
                 ' Print..      ',
                 ' Os Shell     ',
                 ' Exit         ',
                 '','','');
            LastChoice:7),
            (Win :(11,02,37,12);
            Col :(11,01,01,01,01,01,01,01,01,01,01,01);
            Row :(01,01,02,03,04,05,06,07,08,09,10);
            Msg :(' Edit ',
                 ' Coordinate of Node     ',
                 ' Boundary Restraint     ',
                 ' Element Connectivity   ',
                 ' Material Property      ',
                 ' Nodal Aplied Forces    ',
                 'ฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤฤ',
                 ' Node List              ',
                 ' Element List           ',
                 ' Load List              ',
                 '');
            LastChoice:9),
            (Win :(19,02,40,05);
            Col :(19,01,01,01,01,01,01,01,01,01,01,01);
            Row :(01,01,02,03,04,05,06,07,08,09,10);
            Msg :(' Solution ',
                 ' Calculate Force    ',
                 ' Force/Displacement ',
                 '','','','','','','','');
            LastChoice:2),
            (Win :(31,02,52,07);
            Col :(31,01,01,01,01,01,01,01,01,01,01,01);
            Row :(01,01,02,03,04,05,06,07,08,09,10);
            Msg :(' Result ',
                 ' Reaction           ',
                 ' Member Force       ',
                 ' Deflection         ',
                 ' Volumn of Material ',
```

```
                    '','','','','','');
                LastChoice:4),
                (Win :(41,02,56,05);
                Col :(41,01,01,01,01,01,01,01,01,01,01,01);
                Row :(01,01,02,03,04,05,06,07,08,09,10);
                Msg :(' Graph ',
                    ' Geometry    ',
                    ' Displacement ',
                    '','','','','','','','');
                LastChoice:2),
                (Win :(50,02,70,08);
                Col :(50,01,01,01,01,01,01,01,01,01,01,01);
                Row :(01,01,02,03,04,05,06,07,08,09,10);
                Msg :(' Help ',
                    ' How to use    F1 ',
                    ' Information      ',
                    ' About            ',
                    ' Edit Auto Save   ',
                    ' Change BackGround ',
                    '','','','','');
                LastChoice:5)
               );

{------------------------ Funtion Key Messages ---------------------------}

Fkey : FuncMenu = (Col: (02,12,22,32,47,58,64);
                   Row: (25,25,25,25,25,25,25);
                   Msg: ('F1-Help',
                         'F2-Open',
                         'F3-Save',
                         'F5-Directory',
                         'F8-Graph',
                         'Alt-X-Exit','')
                  );


const
  ChoiceMenu : array[1..MaxMenu] of 1..MaxChoice = (1,1,1,1,1,1);
  StatusMenu : 1..MaxMenu = 1;
  MenuFinish : boolean = False;

procedure DisplayMainMenu;
procedure Readfunckey(var Key: Char);
procedure GetMenuChoice(code: byte;var m,c: byte);
procedure ReadString(X,Y: byte;var Done: boolean;var St:String);
procedure ReadRealNum(X,Y: byte;var Num: real);
```

```
procedure ReadIntNum(X,Y: byte;var Num: integer);

implementation

const
  MainMenuColor    = Black + (LightGray shl 4);
  FunctionColor    = Black + (LightGray shl 4);
  MenuHeadColor    = Black + (Cyan shl 4);
  FirstLetterColor = Red   + (LightGray shl 4);
  ChoiceColor      = Black + (LightGray shl 4);
  SelectColor      = Black + (Cyan shl 4);

procedure Readfunckey(var Key: Char);
begin
  Key := ReadKey;
  if Key = #0 then
  begin
    FuncKey := true;
    Key := ReadKey;
  end
  else
    FuncKey := false;
end;

procedure ColorEol(X,Y,Color: byte);
var
  Old: byte;
begin
  Old:=TextAttr;
  GotoXY(X,Y); TextAttr:=Color;
  ClrEol;
  Textattr:=Old;
end;

procedure DisplayFuncKey;
var
  i: byte;
begin
  with FKey do
  begin
    for i:=1 to 6 do
    begin
      PutString(Msg[i],FunctionColor,Col[i],Row[i]);
      PutChar(Msg[i,1],FirstLetterColor,Col[i],Row[i]);
      PutChar(Msg[i,2],FirstLetterColor,Col[i]+1,Row[i]);
      if i=6 then
```

```
      begin
        PutChar(Msg[i,3],FirstLetterColor,Col[i]+2,Row[i]);
        PutChar(Msg[i,4],FirstLetterColor,Col[i]+3,Row[i]);
        PutChar(Msg[i,5],FirstLetterColor,Col[i]+4,Row[i]);
      end;
    end;
  end;
end;

procedure DisplayMainMenu;
var
  I: byte;
begin
  Window(1,1,80,25);
  ColorEol(1,1,MainMenuColor);
  PutChar('o',MainMenuColor,2,1);
  for I:=1 to MaxMenu do
  with Menu[I] do
  begin
    PutString(Msg[0],MainMenuColor,Col[0],Row[0]);
    PutChar(Msg[0,2],FirstLetterColor,Col[0]+1,Row[0]);
  end;
  PutString('F10-Menu',MainMenuColor,70,1);
  ColorEol(1,25,FunctionColor);
  DisplayFuncKey;
end;

procedure ChoiceActive(old,new: byte);
begin
  with Menu[StatusMenu] do
  begin
    PutString(Msg[old],ChoiceColor,Col[old],Row[old]);
    PutString(Msg[new],SelectColor,Col[new],Row[new]);
  end;
end;

procedure MenuActive(new: byte);
var
  OldAttr,i: byte;
begin
  DisplayMainmenu;
  with Menu[new] do
  begin
    PutString(Msg[0],MenuHeadColor,Col[0],Row[0]);
    OldAttr:=TextAttr;
    OpenWindow(Win[1],Win[2],Win[3],Win[4],1,ChoiceColor,1);
```

```
    for i:=1 to LastChoice do
      PutString(Msg[i],ChoiceColor,Col[i],Row[i]);
  end;
  ChoiceActive(1,ChoiceMenu[New]);
end;

procedure MoveUp;
begin
  CurrentChoice:=ChoiceMenu[StatusMenu];
  if CurrentChoice=1 then
    ChoiceMenu[StatusMenu] := Menu[StatusMenu].LastChoice
  else
    if (StatusMenu=2) and (CurrentChoice=7) then
      ChoiceMenu[StatusMenu]:=CurrentChoice-2
      else
        ChoiceMenu[StatusMenu] := CurrentChoice-1;
  ChoiceActive(CurrentChoice,ChoiceMenu[StatusMenu]);
end;

procedure MoveDown;
begin
  CurrentChoice:=ChoiceMenu[StatusMenu];
  if CurrentChoice=Menu[StatusMenu].LastChoice then
    ChoiceMenu[StatusMenu] := 1
  else
    if (StatusMenu=2) and (CurrentChoice=5) then
      ChoiceMenu[StatusMenu]:=CurrentChoice+2
      else
        ChoiceMenu[StatusMenu] := CurrentChoice+1;
  ChoiceActive(CurrentChoice,ChoiceMenu[StatusMenu]);
end;

procedure MoveForward;
begin
  CloseWindow;
  if StatusMenu+1 > MaxMenu then
    StatusMenu:=1
  else
    Inc(StatusMenu);
  MenuActive(StatusMenu);
end;

procedure MoveBack;
begin
  CloseWindow;
  if StatusMenu =1 then
```

```
    StatusMenu := MaxMenu
  else
    Dec(StatusMenu);
  MenuActive(StatusMenu);
end;

procedure TestKey(Key: Char);
begin
  if FuncKey then
    case (Key) of
      #72 : MoveUp;
      #75 : MoveBack;
      #77 : MoveForward;
      #80 : MoveDown;
    end
  else
    case (Key) of
      #13      : QuitMenu:=True;
      #27      : begin
                   QuitMenu:=True;
                   EScPress:=True;
                 end;
      #110     : begin
                   if StatusMenu=1 then
                     ChoiceMenu[StatusMenu]:=1;
                   QuitMenu:=True;
                 end;
      #111,#79 : begin
                   if StatusMenu=1 then
                     ChoiceMenu[StatusMenu]:=2;
                   QuitMenu:=True;
                 end;
    end;
end;

procedure GetMenuChoice(code: byte;var m,c: byte);
var
  i: byte;
begin
  QuitMenu:=False;
  EScPress:=False;
  DisplayMainMenu;
  MenuActive(code);
  repeat
    CursorOff;
    ReadFunckey(Key);
```

```
    TestKey(Key);
  until QuitMenu;
  if EScPress then
  begin
    m:=0;
    c:=0;
  end
  else
  begin
    m:=StatusMenu;
    c:=ChoiceMenu[m];
  end;
  CloseWindow;
  DisplayMainMenu;
end;

procedure MapKey(var Key:char);
begin
  if FuncKey then
    case Key of
      Home_Key : Key := ^A;
      End_Key  : Key := ^Z;
      Lt_Key   : Key := ^S;
      Rt_Key   : Key := ^D;
      Del_Key  : Key := ^G
      else
        Key := #00; {NULL}
    end;
end;

procedure BackSpace(var St: String; var Ptr: byte);
var
  Tail: String;
begin
  if Ptr <> 1 then
  begin
    Tail := Copy(St,Ptr,Length(St)-Ptr+1);
    Delete(St,Ptr-1,1);
    Ptr := Ptr-1;
    Write(^H);
    ClrEol;
    Write(Tail);
    GotoXY(WhereX-Length(Tail),WhereY);
  end;
end;
```

```
procedure LeftArrow(var Ptr:byte);
begin
 if Ptr <> 1 then
 begin
  Ptr := Ptr-1;
  Write(^H);
 end;
end;

procedure RightArrow(St: String;var Ptr:byte);
var
 Len: byte;
begin
 Len:=Length(St);
 if (Ptr<=Len) and (Len<>0) then
 begin
  Ptr := Ptr+1;
  GotoXY(WhereX+1,WhereY);
 end;
end;

procedure ToHome(var Ptr: byte);
begin
 GotoXY(WhereX-Ptr+1,WhereY);
 Ptr:=1;
end;

procedure ToEnd(St: String;var Ptr: byte);
var
 Len: byte;
begin
 Len:=Length(St);
 if Ptr<=Len then
 begin
  ToHome(Ptr);
  GotoXy(WhereX+Len,WhereY);
  Ptr := Len+1;
 end;
end;

procedure Truncate(var St:String; var Ptr: byte);
begin
 St:=COpy(St,1,Ptr-1);
 Ptr:=Length(St)+1;
 if Ptr=1 then
  St:='';
```

```
  ClrEol;
end;

procedure Del(var St: String;var Ptr: byte);
var
  Tail: String;
  Len: byte;
begin
  Len:=Length(St);
  if (Ptr<=Len) and (Len<>0) then
  begin
    Tail:=Copy(St,Ptr+1,Length(St)-Ptr);
    Delete(St,Ptr,1);
    ClrEol;
    Write(Tail);
    GotoXY(WhereX-Length(tail),WhereY);
  end;
end;

procedure Clear(var St: String;var Ptr:byte);
begin
  ToHome(Ptr);
  ClrEol;
  St:='';
end;

procedure Character(var St: String; Ch: char;var Ptr: byte);
var
  P,X,Len: byte;
begin
  Len:=Length(St);
  if Ptr > Len then
  begin
    if (WhereX <= (Lo(WindMax)-Lo(windMin))) then
    begin
      St := St+Ch;
      Ptr:= Ptr+1;
      Write(Ch);
    end;
  end
  else
    if (Len < (Lo(WindMax)-Lo(WindMin))) then
    begin
      X:=WhereX;
      P:=Ptr;
      Insert(Ch,St,Ptr);
```

```
    Ptr:=Ptr+1;
    ToHome(P);
    Write(St);
    GotoXY(X+1,WhereY);
   end;
end;

procedure ProcessFuncKey(Key: char;
                         var St: String;
                         var Ptr: byte;
                         var Quit: boolean);
begin
  Case Key of
   ^M : begin
        if St='' then
          St:=#13;
          Quit:=True;
        end;
   ^[ : begin
        St:=#27;
        Quit:=True;
       end;
   ^H : BackSpace(St,Ptr);
   ^G : Del(St,Ptr);
   ^S : LeftArrow(Ptr);
   ^D : RightArrow(St,Ptr);
   ^T : Truncate(St,Ptr);
   ^A : ToHome(Ptr);
   ^Z : ToEnd(St,Ptr);
   ^Y : Clear(St,Ptr);
   else
     if Key>=#32 then
       Character(St,Key,Ptr);
  End;
end;

procedure ReadString(x,y: byte;var Done: boolean;var St:String);
var
  Ch: char;
  Ptr,k: byte;
  Quit: boolean;
begin
  Quit:=False;
  Ptr:=1;
  GotoXY(X,Y); Write(St);
  GotoXY(X,Y);
```

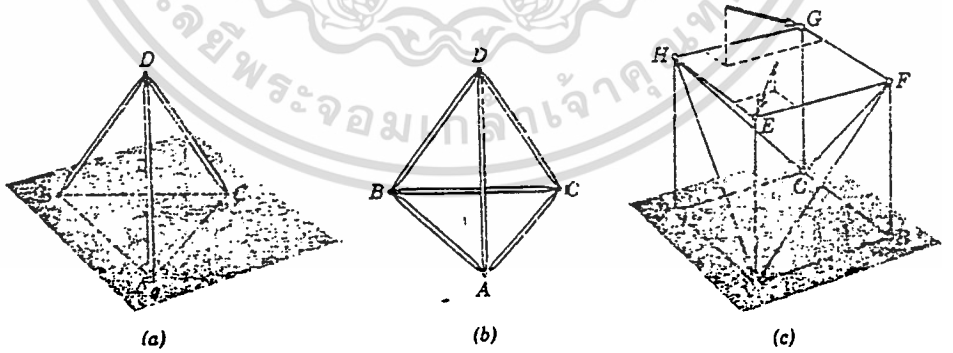
```
ReadfuncKey(Key);
MapKey(Key);
if (Key<#32) then
 ProcessFuncKey(Key,St,Ptr,Quit)
 else
 begin
  St:=Key;
  Ptr:=2;
  GotoXY(X,Y);
  ClrEol;
  Write(St);
 end;
 while Not(Quit) do
 begin
  ReadFuncKey(Key);
  MapKey(Key);
  if Key=#27 then
  begin
   Done:=False;
   GotoXY(X,Y);
   for k:=1 to Length(St) do
    Write(' ');
   GotoXY(X,Y);
  end
  else
  begin
   Done:=True;
   ProcessFuncKey(Key,St,Ptr,Quit);
  end;
 end;
end;

procedure ReadRealNum(X,Y: byte;var Num:real);
var
 Done:boolean;
 a,b,c,Ptr:byte;
 St: String;
 Code: integer;
begin
 St:=' ';
 while(St[Length(st)]='0') do
  St[0]:=Chr(Ord(St[0])-1);
 repeat
  ReadString(X,Y,Done,St);
  Val(St,Num,Code);
  if Code<>0 then
```

```
    begin
      GotoXY(X,Y); ClrEol;
      GotoXY(X,Y); St:='';
    end;
  until Code=0;
end;

procedure ReadIntNum(X,Y: byte;var Num: integer);
var
  Done: boolean;
  Ptr: Byte;
  St: String;
  Code: integer;
begin
  St:='';
  repeat
    ReadString(X,Y,Done,St);
    Val(St,Num,Code);
    if Code<>0 then
    begin
      GotoXY(X,Y); ClrEol;
      GotoXY(X,Y); St:='';
    end;
  until Code=0;
end;

end. {of unit}
```

ภาคผนวก ข

โครงข้อหมุนสามมิติ

Space Truss คือ Truss สามมิติภาพความคิดของ Space Truss คือ ชิ้นส่วนที่แข็งแรงซึ่งถูกนำมาเชื่อมต่อกันทีปลาย ด้วยระบบต่อด้วย Ball,Socket Connection และ Space Truss ขั้นมูลฐานที่สุดประกอบด้วยชิ้นส่วน 6 ชิ้นเชื่อมต่อกันที่ปลายก่อให้เกิดเป็นรูป Tetrahedron ทำให้เกิดเป็นโครงสร้างที่แข็งแรงไม่โยกคลอน ดังภาพ ๖ การต่อเติมโครงสร้างสามมิติจะต้องใช้ชิ้นส่วนทีละสามชิ้นเพิ่มเข้าไปจึงจะได้โครงสร้างที่แข็งแรง โดยชิ้นส่วนทั้งสามที่นำไปเติมจะพบกันที่จุดหนึ่งใน Space ส่วนปลายทั้งสามที่เหลือต่ออยู่กับจุดคงที่ใดๆ บน Truss เดิมดังในภาพที่ 1 ชิ้นส่วน AF BF และ CF ต่อติดกับฐานแล้วมาเชื่อมด้วยกันที่จุด F ใน Space ทำนองเดียวกับจุด H เป็นจุดคงที่ในอวกาศที่ชิ้นส่วน AH,DH และ CH มาพบกัน ชิ้นส่วน CG ,FG และ HG ต่อติดอยู่กับจุดคงที่ C,F และ H ปลายที่เหลือมาพบกันที่จุด G ในอวกาศ จุด E ถูกสร้างขึ้นจากหลักเกณฑ์เดียวกัน โครงสร้างทั้งหมดแข็งแรง เมื่อได้รับ Load มันจะถ่ายทอดแรงที่ได้รับให้แก่ชิ้นส่วนทุกชิ้น ตามแนวความคิดนั้น Truss จะต้องมีจุดที่รองรับ และมักใช้ Ball Scotket Joint ในการต่อชิ้นส่วนต่างๆ ของ Space Truss เพื่อป้องกันไม่ให้ชิ้นส่วนบิดตัว ถ้าที่จุดต่อชิ้นส่วนของ Space Truss มีการเชื่อมประสานหรือขันนอตยึด และถ้าแนวกึ่งกลางของชิ้นส่วนทุกชิ้นนั้นพบกันที่จุดเชื่อมพอดี ข้อสมมุติฐานเกี่ยวกับชิ้นส่วนรับแรงสองแรงภายใต้แรงกดอัด และแรงดึงสามารถนำมาใช้กับ Space Truss ด้วยเช่นเดียวกัน กับที่เคยใช้กับ Plane Truss มาแล้ว



รูปที่ 1

สำหรับ Space Truss ใดที่ถูกรองรับหรือยึดตรึงด้วย Support ภายนอกในวิถีทางที่จะทำให้ Truss เป็น Statically Determinate ทั้งระบบจะเกิดความสัมพันธ์ระหว่างจำนวนจุด Joint กับจำนวนชิ้นส่วนที่จำเป็น สำหรับเสถียรภาพสำหรับโครงสร้าง เป็นค่าจำกัดค่าหนึ่ง เนื่องจากการสมดุลย์ของแรงที่ Joint หนึ่งๆ สอดข้องกับสมการสมดุลย์ 3 สมการ ดังนั้นจะมีสมการ

3j สมการสำหรับ Truss สามมิติอย่างง่ายที่มีจำนวน Joint ทั้งหมด j จุด ถ้า Truss ทั้งหมดประกอบด้วยชิ้นส่วน m ชิ้น จะมีตัวที่ไม่ทราบค่าทั้งหมด m + 6 ตัว ทั้งนี้เพราะสำหรับ Space Truss ทั่วไปจะมีปฏิกิริยาจากที่รองรับถึง 6 ตัว อยู่ก่อนแล้วที่จะต่อเติม Truss ดังนั้น สำหรับ Space Truss ทั่วไปที่ประกอบขึ้นจากรูป Tetrahedron จะสอดข้องกับสมการ m + 6 = 3j ถ้า Truss เป็นปัญหา Statically Determinate ความสัมพันธ์อันนี้เป็นเงื่อนไขจำเป็นสำหรับเสถียรภาพของโครงสร้าง แต่ไม่ใช่เงื่อนไขที่พอเพียง เพราะชิ้นส่วนหนึ่งหรือหลายชิ้นใน m ชิ้น สามารถจัดวางให้อยู่ในลักษณะที่ไม่ช่วยให้โครงสร้างมีเสถียรภาพ ถ้า m + 6 > 3j Truss นั้นจะมีจำนวนชิ้นที่มากกว่าจำนวนสมการอิสระ Truss เป็น Statically Indeterminate ภายใน โดยมีชิ้นส่วนเกิน แต่ถ้า m + 6 < 3j Truss มีชิ้นส่วนภายในเพียงพอแต่ไม่เสถียรภาพอาจโยกเยกได้เมื่อรับแรง

ความสัมพันธ์ระหว่างชิ้นส่วนและระหว่าง Joint มีความสำคัญสำหรับ Space Truss และช่วยในการออกแบบ Space Truss เป็นอย่างมาก เพราะรูปร่างและโครงสร้างสามมิติ เมื่อตรวจสอบด้วยสายตา ไม่อาจจะประมาณได้ว่ามีเสถียรภาพหรือไม่ เหมือนกับ Plane Truss ซึ่งตรวจสอบได้ง่ายๆ

การวิเคราะห์ Truss สามมิติอาจจะใช้วิธีการของ Joint หรือจะใช้วิธีการของ Section ในการใช้วิธีการ Joint จะต้องใช้สมการสมดุลย์ ของแรงที่จุด Joint หนึ่งๆ ทั้งสามสมการคือ $\sum FX = 0$, $\sum FY = 0$, $\sum FZ = 0$ ซึ่งเขียนรวมกันเป็นสมการเวคเตอร์ว่า $\sum F = 0$

การวิเคราะห์จะเริ่มที่จุด Joint ใดๆ ซึ่งทราบแรงแล้วอย่างน้อยที่สุด 1 แรงและมีแรงที่ไม่ทราบค่าไม่เกินสามแรง ส่วนวิธีการของ สมการสมดุลย์ทั้ง 6 สมการ ซึ่งรวมเข้าด้วยกันเป็นสมการเวคเตอร์ได้ 2 สมการคือ

$$\sum F = 0 \quad \text{และ} \quad \sum M = 0$$

Truss สามมิติทุกชิ้นที่ถูกตัดออกมาจะสอดคล้องกับสมการทั้งสองนี้ การตัด Space Truss จะต้องเลือกแนวทางที่ตัดไม่ให้ผ่านชิ้นส่วนเกิน 6 ชิ้น ที่ยังไม่ทราบแรง เพราะว่ามีสมการสมดุลย์อิสระเพียง 6 สมการ และวิธีการของ Section มักไม่นิยมใช้ในการวิเคราะห์ Space Truss เพราะการตั้งแกนโมเมนต์ที่จุดใด ๆ มักจะจำกัดแรงที่ไม่ทราบค่าไปเสียทั้งหมด ไม่เหลือไว้ให้วิเคราะห์เพียงแรงเดียว โดยทั่วไปจึงนิยมใช้วิธีการของ Joint และใช้วิธีการทางด้านเวคเตอร์เข้ามาช่วย เพราะปัญหาสามมิติ ถ้าใช้เรขาคณิต และตรีโกณมิติอย่างเดียวในการคำนวณจะประสบความสำเร็จได้ยากลำบากมาก