DATABASE MANAGEMENT SYSTEM (DBMS)
ON MICROSOFT WINDOWS

โดย
น.ส.สุชฎา ศรีงางาม
น.ส.สุวรรณ ศรีหะวรรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2534

ปริญญานิพนธ์ปีการศึกษา 2534

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง DATABASE MANAGEMENT SYSTEM (DBMS)ON MICROSOFT WINDOWS

ผู้จัดทำ น.ส. สุขฟ้า ศรีงางาม
น.ส. สุวรรณ ศรีหะวรรณ์

กฤตวัน เครือตราชู อาจารย์ที่ปรึกษา

(อาจารย์ กฤตวัน เครือตราชู)

จุดประสงค์

1. ทดลองสร้างส่วนหนึ่งของ DBMS(Database Management System) บน Microsoft Windows
2. ศึกษาการทำงานและการเขียนโปรแกรมบน Microsoft Windows

ขอบเขตของโครงงาน (ภาคเรียนที่ 2)

1. เขียนโปรแกรมฟังก์ชั่นที่จำเป็นในงานด้านฐานข้อมูลโดยมีการ interface ผ่าน Windows
2. เขียนโปรแกรมจัดเก็บข้อมูลตาม data structure ของไฟล์ข้อมูล ที่ออกแบบไว้แล้วในเทอมที่ 1

DBMS ON MICROSOFT WINDOWS

น.ส. สุชฟ้า ศรีงางาม

น.ส. สุวรรณ ศรีหะวรรณ์

อาจารย์ที่ปรึกษา

อาจารย์ กฤตวัน เครือตราชู

ปีการศึกษา 2534

บทคัดย่อ

ในปัจจุบันระบบฐานข้อมูล เข้ามามีบทบาทต่องานในหลายสาขาอาชีพ การพัฒนาซอฟต์แวร์ที่เกี่ยวกับฐานข้อมูล จึงมีประโยชน์และมีความสำคัญอย่างยิ่ง คุณสมบัติของฐานข้อมูลที่ดีนอกจากจะสามารถเก็บข้อมูลและความสัมพันธ์ของข้อมูลได้อย่างถูกต้องแล้ว ยังต้องใช้งานได้ง่ายอีกด้วย ดังนั้นโครงงานนี้จึงเลือกเอา Microsoft Windows มาเป็น environment เนื่องจาก Microsoft Windows มีส่วนที่ช่วยผู้ใช้ในด้านการติดต่อกับฐานข้อมูล

โครงงานนี้เป็นการสร้างส่วนหนึ่งของ DBMS (Database Management System) ซึ่งง่ายแก่การนำไปใช้ โดยผู้ใช้สามารถติดต่อกับฐานข้อมูลผ่านทางเมนู DBMS ที่สร้างขึ้นนี้ประกอบด้วยฟังก์ชั่นที่จำเป็นในงานด้านฐานข้อมูล ซึ่งเราจะให้ชื่อโครงงานที่สร้างฐานข้อมูลบน Microsoft Windows นี้ว่า **WinBase** และในตอนต่อจากนี้ไปเราจะกล่าวถึง WinBase ในความหมายที่กล่าวไว้แล้วข้างต้น

ปริญญานิพนธ์ฉบับนี้มีเนื้อหาเกี่ยวกับ

การสร้างส่วนหนึ่งของ DBMS บน Microsoft Windows ซึ่งมีรูปแบบคล้ายกับดีเบสทรีพลัส โดยมีการเลือกและแก้ไขบางฟังก์ชั่นที่เหมาะสมกับงานด้านฐานข้อมูล ส่วนของโครงงานนี้ แบ่งออกเป็น 2 ส่วน คือ

- ส่วน user interface
- ส่วน database functions

ซึ่งในรายงานนี้เป็นส่วนที่เกี่ยวกับ database functions

DBMS ON MICROSOFT WINDOWS

MISS SUKFAH SRINGANGAM

MISS SUWAN SRIHAWAN

ADVISOR

MISS KRITAWAN KRUATRACHUE

ACADEMIC YEAR 1992

ABSTRACT

To day, database is applicable to several job. The development of software that attach database is very important. Above the property of good database is used to store data constain and relation of data, it must have user friendlyness. So Microsoft Windows that help user to be easy to interface database.

The creation of DBMS (Database Manangment System) is purpose of this project. User can interface database by selecting menu. The database that is create is WinBase. It has function that need in database.

This thesis is concerned to create DBMS on Microsoft Windows that is like dbase III plus. But it selectes some function and verify some function that avail to database. This project has two part. One part is interface and other part is database function. This thesis is part of database function.

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญรูป

บทที่ 1

บทนำ

1.1 WinBase คืออะไร

จุดประสงค์ประการหนึ่งของโครงงานนี้คือ เป็นการสร้างส่วนหนึ่งของ DBMS (Database Management System) ให้ง่ายแก่การนำไปใช้ในการติดต่อกับฐานข้อมูล โดยผู้ใช้สามารถติดต่อกับฐานข้อมูลผ่านทาง เมนู DBMS ที่สร้างขึ้นนี้ประกอบด้วยฟังก์ชั่นที่จำเป็นในงานด้านฐานข้อมูล เพื่อความสะดวกในการที่จะอ้างถึงการสร้างฐานข้อมูลบน Microsoft Windows ในตอนต่อ ๆ ไป เราจึงให้ชื่อการสร้างฐานข้อมูลบน Microsoft Windows นี้ว่า " WinBase " และในตอนต่อจากนี้ไปเราจะกล่าวถึง WinBase ในความหมายที่กล่าวไว้แล้วข้างต้น

1.2 ไฟล์ที่จำเป็นต้องใช้ใน WinBase

ไฟล์ที่จำเป็นต้องใช้ในการทำส่วนของ WinBase นี้จะประกอบด้วย 3 ไฟล์ด้วยกัน คือ

- file.dbf
- file.mem
- file.ndx
- file.bt

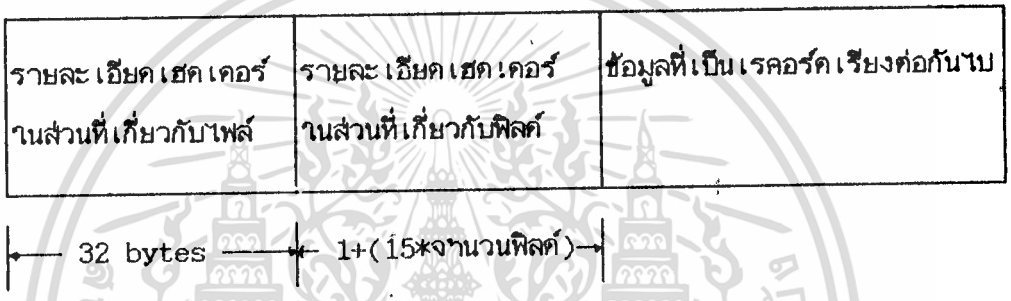
รายละเอียดเกี่ยวกับไฟล์ ทั้ง 4 ชนิดนี้จะได้กล่าวถึงในตอนต่อไป

บทที่ 2

โครงสร้างของไฟล์ข้อมูล(FILE.DBF)

2.1 file.dbf คืออะไร

file.dbf เป็น text file ที่ทำหน้าที่เก็บข้อมูลต่าง ๆ ข้อมูลในที่นี้ก็หมายถึงส่วนที่เป็นเฮดเดอร์(header) และส่วนที่เป็นข้อมูลในฐานข้อมูลจริง ๆ แสดงได้ดังรูปที่ 1



รูปที่ 2.1 โครงสร้างของแฟ้มข้อมูล database(file.dbf)

จากรูปที่ 1 ในส่วนเฮดเดอร์ ของ file.dbf จะแบ่งออกเป็น 2 ส่วนคือ

- เฮดเดอร์ ที่เป็นรายละเอียดเกี่ยวกับไฟล์
- เฮดเดอร์ ที่เป็นรายละเอียดเกี่ยวกับฟิลด์

ส่วนของเฮดเดอร์ที่เป็นรายละเอียดเกี่ยวกับไฟล์ จะทำการเก็บข้อมูลต่าง ๆ เรียงกันไปดังนี้

- โค้ด(code) เป็นตัวเช็คว่า file.dbf ที่ต้องการเปิด เป็น file ที่สร้างจาก WinBase หรือไม่ โดยจะมีการให้โค้ดทุกครั้งที่มีการสร้าง file.dbf จาก WinBase
- วันที่ เป็นวันที่ครั้งสุดท้ายของการแก้ไข file.dbf นั้น
- จำนวนฟิลด์
- จำนวนเรคอร์ด
- ความยาวของเฮดเดอร์ทั้งหมดซึ่งมีขนาดเท่ากับ ผลรวมความยาวของเฮดเดอร์ของไฟล์กับ (15*จำนวนฟิลด์)+1
- ความยาวของเรคอร์ด

ข้อมูลในไบต์ต่าง ๆ ของ เฮด เดอร์ส่วนรายละ เอียด เกี่ยวกับไฟล์เป็นดังตารางที่ 1

ไบต์ที่	ความหมาย
1-3	เก็บโค้ด(CODE)
4-11	2 ไบต์แรก เก็บวันที่แก้ไขครั้งสุดท้าย
	2 ไบต์ต่อมา เก็บเดือนที่แก้ไขครั้งสุดท้าย
	4 ไบต์สุดท้าย เก็บปีที่แก้ไขครั้งสุดท้าย
12-13	จำนวนฟิลด์ 2 ไบต์
14-19	จำนวน เรคอร์ด 6 ไบต์
20-22	ความยาวของ เฮด เดอร์
23-26	ความยาวของ เรคอร์ด
27-32	จำนวน เรคอร์ดของฟิลด์ เมโม

ตารางที่ 2.1 ข้อมูลในไบต์ต่าง ๆ ของ เฮด เดอร์

ส่วนของ เฮด เดอร์ที่เป็นรายละ เอียดเกี่ยวกับฟิลด์ จะทำการ เก็บ เฮด เดอร์รายละ เอียด เกี่ยวกับฟิลด์ ซึ่ง เป็นกลุ่มข้อมูลของแต่ละฟิลด์ เรียงกันไป ซึ่งแต่ละฟิลด์จะประกอบด้วยกลุ่มข้อมูล ขนาด 15 ไบต์ ดังนี้

- ชื่อฟิลด์
- ชนิดของฟิลด์

  มี 4 ชนิด คือ

  C : Character

  N : Numeric

  D : Date การ เก็บข้อมูลในฟิลด์ที่มีชนิดของฟิลด์เป็น Date นี้ จะต้อง เก็บใน รูปแบบดังนี้ YY/MM/DD (YEAR/MONTH/DAY)

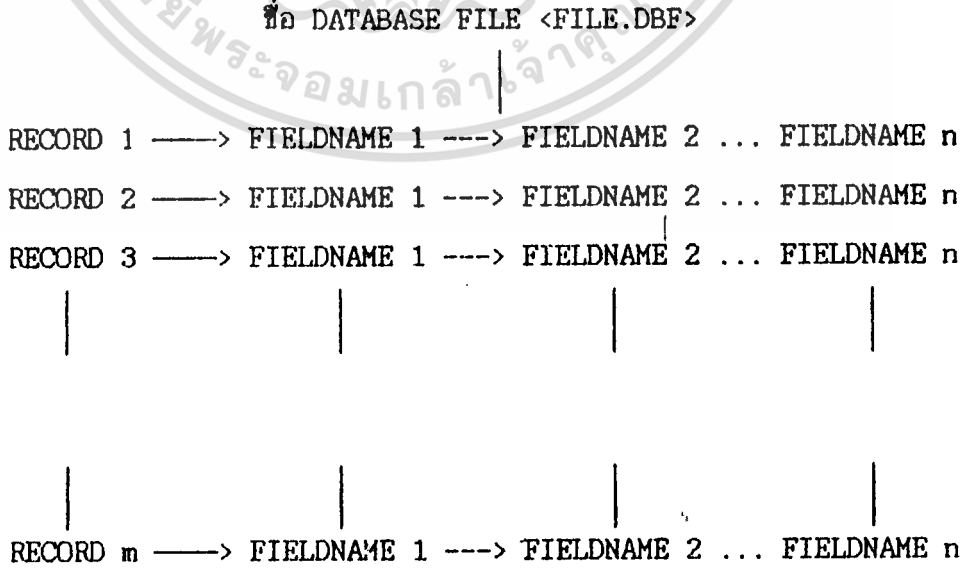
  M : Memo ฟิลด์ที่มีชนิดของฟิลด์เป็นชนิด เมโมนี้จะ เก็บข้อมูลแยกไว้ต่างหากคือไว้ใน file.mem ซึ่งจะกล่าวถึงในตอนต่อไป
- ความกว้างของฟิลด์
- ความกว้างของจุดทศนิยม

ข้อมูลในไบต์ต่าง ๆ ของเฮดเดอร์ส่วนรายละเอียดเกี่ยวกับฟิลด์เป็นดังตารางที่ 1

1-10	ชื่อฟิลด์ ซึ่งเก็บได้ 10 ตัวอักษร
11	ชนิดของฟิลด์ โดยใช้รหัส
	C 43 คือตัวอักษร
	D 44 คือวันที่
	M 4D คือเมโม
	N 4E คือตัวเลข
12-13	ความกว้างของฟิลด์เช่น รหัส 1B คือ
	ตัวเลข 27
14-15	จำนวนตัวเลขจุดทศนิยม ถ้าเป็นฟิลด์ที่
	ไม่ใช่ตัวเลขค่าจะเป็น 0

ตารางที่ 2.2 ข้อมูลในไบต์ต่าง ๆ ต่อจากส่วนที่เป็นเฮดเดอร์

ในไบต์สุดท้ายของเฮดเดอร์ก่อนจะขึ้นตัวข้อมูลคือ 0D 0A 00 ข้อมูลจะถูกเก็บเป็นรหัสเอสกีเรียงกันไปทีละเรคอร์ด ทุก ๆ เรคอร์ดรวมกันเรียกว่า ข้อมูล ซึ่งในแต่ละเรคอร์ดจะประกอบด้วย ฟิลด์ ต่าง ๆ สามารถอธิบายได้ดังรูปที่ 2

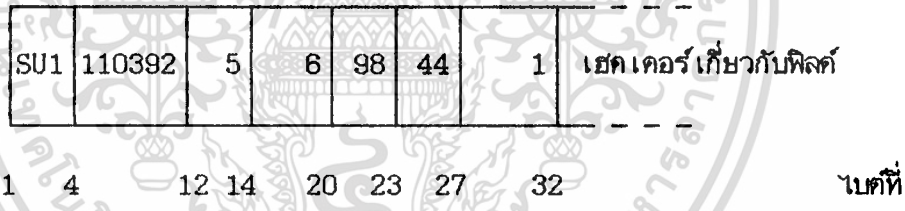


รูปที่ 2.2 แสดงรายละเอียดของข้อมูลใน file.dbf

ตัวอย่างรายละเอียดเกี่ยวกับไฟล์ database(file.dbf)

```
CODE  SU1
แก้ไขไฟล์ database(file.dbf)ครั้งสุดท้ายเมื่อวันที่ 11/03/92
จำนวนฟิลด์ในไฟล์ database(file.dbf) มี 5 ฟิลด์
จำนวนเรคอร์ดในไฟล์ database(file.dbf) มี 3 เรคอร์ด
ความยาวเฮดเดอร์เท่ากับ 98 ไบต์ (32+1+(15*จำนวนฟิลด์))
ความยาวเรคอร์ดเท่ากับ 44 ไบต์
จำนวนเรคอร์ดของฟิลด์เมโม 1 เรคอร์ด
```

รูปที่ 2.3 รายละเอียดเกี่ยวกับไฟล์



รูปที่ 2.4 รูปแสดงรายละเอียดเฮดเดอร์เกี่ยวกับไฟล์

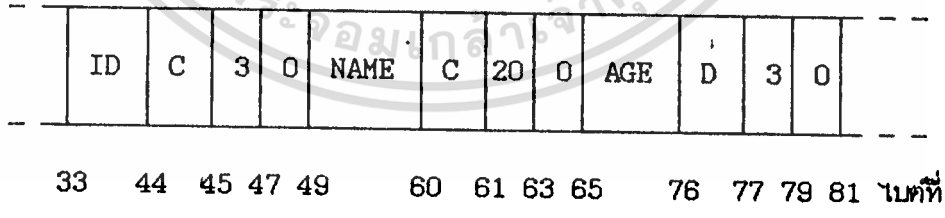
ส่วนที่ต่อจากรายละเอียดเฮดเดอร์เกี่ยวกับไฟล์คือ รายละเอียดเฮดเดอร์เกี่ยวกับฟิลด์ของเรคอรด์ ซึ่งผู้ใช้จะเป็นผู้กำหนดขึ้น โดยการกำหนดจะต้องกำหนดถึง ชื่อฟิลด์ ชนิดของฟิลด์ ความกว้างของฟิลด์ และความกว้างของจุดทศนิยม ดังรูปที่ 5

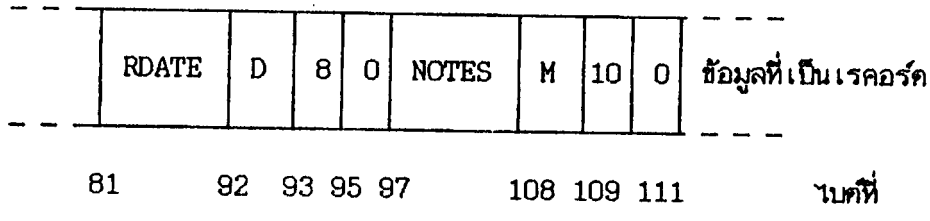
การกำหนดเหล่านี้จะสามารถทำได้หลังจากที่มีการสร้าง file.dbf แล้ว โดยจะเก็บไว้ในส่วนเฮดเดอร์เกี่ยวกับไฟล์และเฮดเดอร์เกี่ยวกับฟิลด์ของ file.dbf

ตัวอย่างรายละเอียดการกำหนดฟิลด์ของเรคอรด์

Field	Field Name	Type	Width	Dec
1	ID	Character	3	0
2	NAME	Character	20	0
3	AGE	Numeric	3	0
4	RDATE	Date	8	0
5	NOTES	Memo	10	0
** Totals **			44	

รูปที่ 2.5 รายละเอียดการกำหนดฟิลด์ข้อมูลในแฟ้มข้อมูล





รูปที่ 2.6 รูปแสดงรายละเอียดของเฮดเดอร์ที่เกี่ยวกับฟิลด์

ถ้าข้อมูลที่ต้องการ เก็บมีดังนี้

Record#	ID	Name	Age	Rdate	Notes
1	A01	Dang	26	91/01/01	M
2	A02	Vipa	25	91/02/10	
3	A03	Piti	35	91/04/23	M

Notes

Record# notes

1 He is an expert on WinBase

3 He is a typist

รูปที่ 2.7 รูปแสดงตัวอย่างข้อมูลที่ต้องการเก็บไว้ใน file.dbf

หลังจากที่ทำการรับข้อมูลเป็นเรคอร์ด ๆ แล้ว ก็จะทำการเก็บข้อมูลเหล่านี้ไว้ในส่วนของตัวข้อมูล ของ file.dbf ที่สร้างขึ้นมาแล้ว โดยจะเก็บทุก ๆ เรคอร์ดติดต่อกันไปใน file.dbf ส่วนฟิลด์ที่มีชนิดของฟิลด์เป็นเมโมนั้น file.dbf จะทำการเก็บพอยน์เตอร์ ซึ่งจะชี้ไปยัง file.mem แสดงได้ดังรูป

A01	Dang	26	91/01/01	1	A02	Vipa	25	91/02/10	

A03	Piti	35	91/04/23	3

รูปที่ 2.8 โครงสร้างของตัวข้อมูลหลังจากที่มีการ เก็บข้อมูลแล้ว

บทที่ 3

โครงสร้างของไฟล์เมโม(FILE.MEM)

3.1 file.mem คืออะไร

ไฟล์เมโม(file.mem) เป็น text file ที่ทำหน้าที่เก็บข้อมูลของฟิลด์ที่มีชนิดของฟิลด์เป็นชนิดเมโมใน file.dbf แต่ละเรคอร์ด กล่าวคือ เมื่อมีการกำหนดฟิลด์ชนิดเมโมใน file.dbf และทำการใส่ข้อมูลในฟิลด์นี้ในเรคอร์ดใดเรคอร์ดหนึ่ง เป็นครั้งแรกก็จะไปทำการสร้าง file.mem แยกต่างหากอีกไฟล์หนึ่ง โดยชื่อของ file.mem จะมีชื่อเดียวกับ file.dbf ข้อมูลในฟิลด์เมโมแต่ละเรคอร์ดจะมีความยาวไม่เกิน 99 ตัวอักษร

ใน file.dbf ส่วนของข้อมูลที่มีชนิดของฟิลด์เป็นแบบเมโมนั้น จะเก็บพอยน์เตอร์ที่จะอ้างไปยัง file.mem พอยน์เตอร์ที่ใช้ในการอ้างถึง file.mem จะเป็นลำดับที่ของเรคอร์ดที่มีการเก็บข้อมูลในฟิลด์เมโม และพอยน์เตอร์ที่ใช้ในแต่ละเรคอร์ดนี้ใช้ข้อมูลขนาด 6 ไบต์

He is an expert on WinBase	He is a typist

รูปที่ 3.1 แสดงโครงสร้างของไฟล์เมโม (file.mem)

หลังจากที่มีการเก็บข้อมูลใน file.dbf แล้ว ถ้าผู้ใช้ต้องการจัดการกับข้อมูลแบบ index จะต้องมีการเรียกฟังก์ชัน index ก่อน โดยในฟังก์ชัน index จะทำหน้าที่รับ field index จากผู้ใช้มาสร้างไบนารีทรี ซึ่ง field index นั้นต้อง unique(ค่าของ field index จะต้องไม่ซ้ำกันในแต่ละเรคอร์ด) เมื่อมีการสร้างไบนารีทรีหลังจากที่เรียกฟังก์ชัน index แล้ว จะต้องมีการเก็บไบนารีทรีนั้นไว้ใน file.bt เพื่อประโยชน์ในการโหลดมาไว้ในหน่วยความจำในภายหลัง ส่วน field index ที่ทำการ sort แบบอินออร์เดอร์แล้ว จะเก็บไว้อีกไฟล์หนึ่ง คือ file.ndx ทั้ง file.bt และ file.ndx เราจะได้กล่าวถึงในบทต่อ ๆ ไป

บทที่ 4

โครงสร้างของ FILE.BT

4.1 file.bt คืออะไร

file.bt คือ ไฟล์ที่ทำหน้าที่เก็บไบนารีทรีที่สร้างได้จากการทำฟังก์ชั่น index แล้ว เพื่อที่จะสามารถโหลดไบนารีทรีขึ้นมาในหน่วยความจำได้ในภายหลัง โดย file.bt นี้จะแบ่งเป็น 2 ส่วนคือ ส่วนเฮดเดอร์ ซึ่งจะเก็บค่าระดับสูงสุดของไบนารีทรี และส่วนที่เก็บข้อมูล file.bt จะถูก update เมื่อมีการเปลี่ยนแปลงแก้ไขข้อมูลใน file.dbf

4.2 ส่วนประกอบของโหนดในการสร้างไบนารีทรี

ในการ sort ข้อมูลจะเลือกเฉพาะ field index มาสร้างไบนารีทรี ซึ่งแต่ละโหนดของไบนารีทรี จะประกอบด้วย

-left pointer เป็นพอยน์เตอร์ที่ชี้ไปยังโหนดที่อยู่ทางซ้ายของรูตโหนด

-pointer เป็นพอยน์เตอร์ที่จะอ้างถึงแอดเดรสเริ่มต้นของแต่ละเรคอร์ดใน file.dbf

-level เป็นตัวบอกถึงระดับของ field index ในไบนารีทรี โดยได้ค่าระดับนี้จากการสร้างไบนารีทรี ซึ่งกำหนดให้ระดับสูงสุดของไบนารีทรีเป็น 1

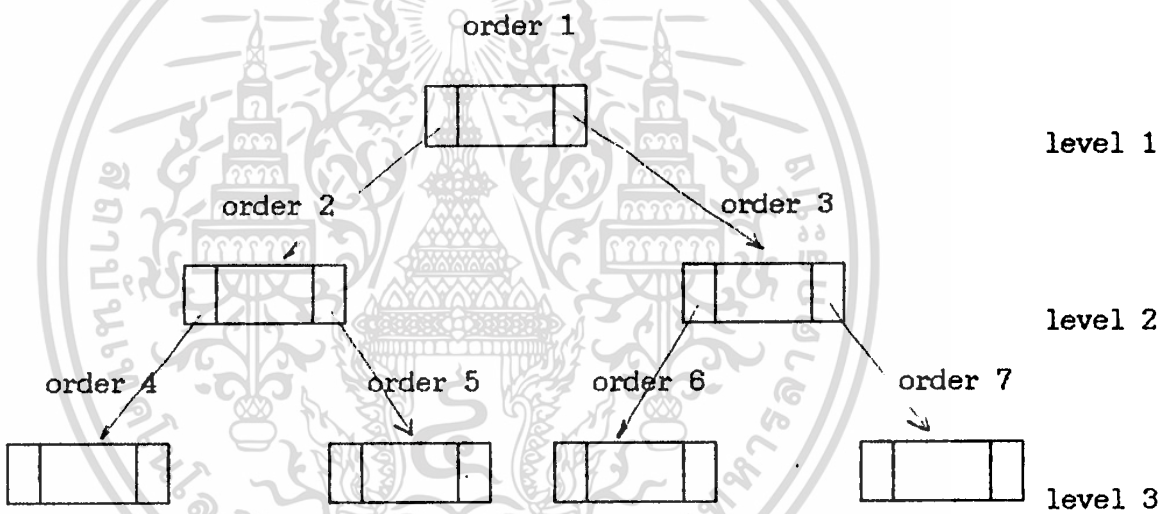
-order เป็นค่าประจำตำแหน่งของ field index ขณะที่อยู่ในไบนารีทรี หลักการการกำหนดค่าประจำตำแหน่งได้อธิบายไว้ในหัวข้อถัดไป

-field index เป็นฟิลด์ใดฟิลด์หนึ่งในเรคอร์ดที่ถูกเลือกให้เป็นคีย์ในการสร้างไบนารีทรี

-right pointer เป็นพอยน์เตอร์ที่ชี้ไปยังโหนดที่อยู่ทางขวาของรูตโหนด

4.3 การกำหนด order ในไบนารีทรี

1. กำหนดให้โหนดเริ่มต้น(รูตโหนด) มีค่าประจำตำแหน่ง(order) เป็น 1
2. ในกรณีที่มี left node ค่าประจำตำแหน่งของ left node นั้น จะหาได้จาก
   ค่าประจำตำแหน่งของ left node = ค่าประจำตำแหน่งของ parent node * 2
   และในกรณีที่ไม่มีค่า left node ก็จะไม่มีการคำนวณหาค่าประจำตำแหน่ง เพราะค่าประจำตำแหน่งเป็นค่าประจำตำแหน่งของโหนดใด ๆ ใน ไบนารีทรี
3. ในกรณีที่มี right node ค่าประจำตำแหน่งของ right node นั้น จะหาได้จาก
   ค่าประจำตำแหน่งของ right node=(ค่าประจำตำแหน่งของ parent node*2)+1
   เช่นเดียวกัน ถ้าในกรณีที่ไม่มีค่า right node ก็ไม่มีการคำนวณหาค่าประจำตำแหน่ง

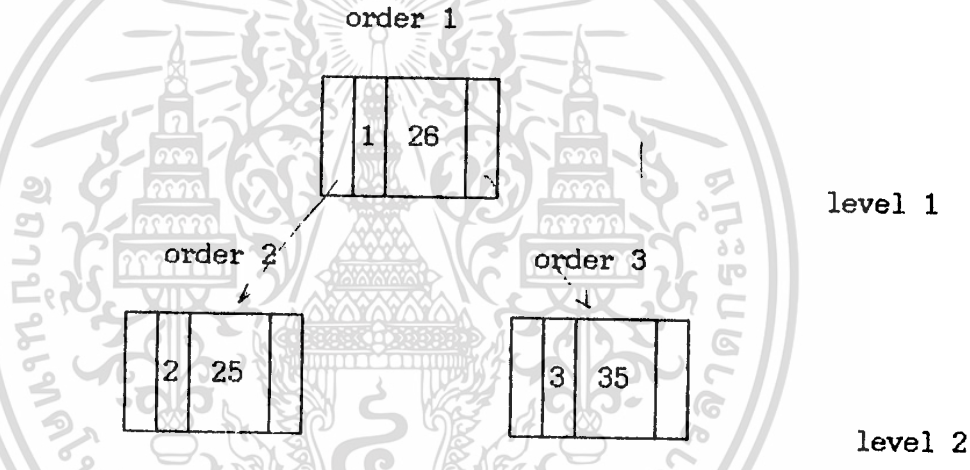


รูปที่ 4.1 รูปแสดงค่าระดับ และ ค่าประจำตำแหน่งของไบนารีทรี

ตัวอย่าง การเก็บข้อมูลใน file.dbf กำหนดให้มีข้อมูลดังนี้

Record#	ID	Name	Age	Rdate	Notes
1	A01	Dang	26	91/01/01	M
2	A02	Vipa	25	91/02/10	
3	A03	Piti	35	91/04/23	M

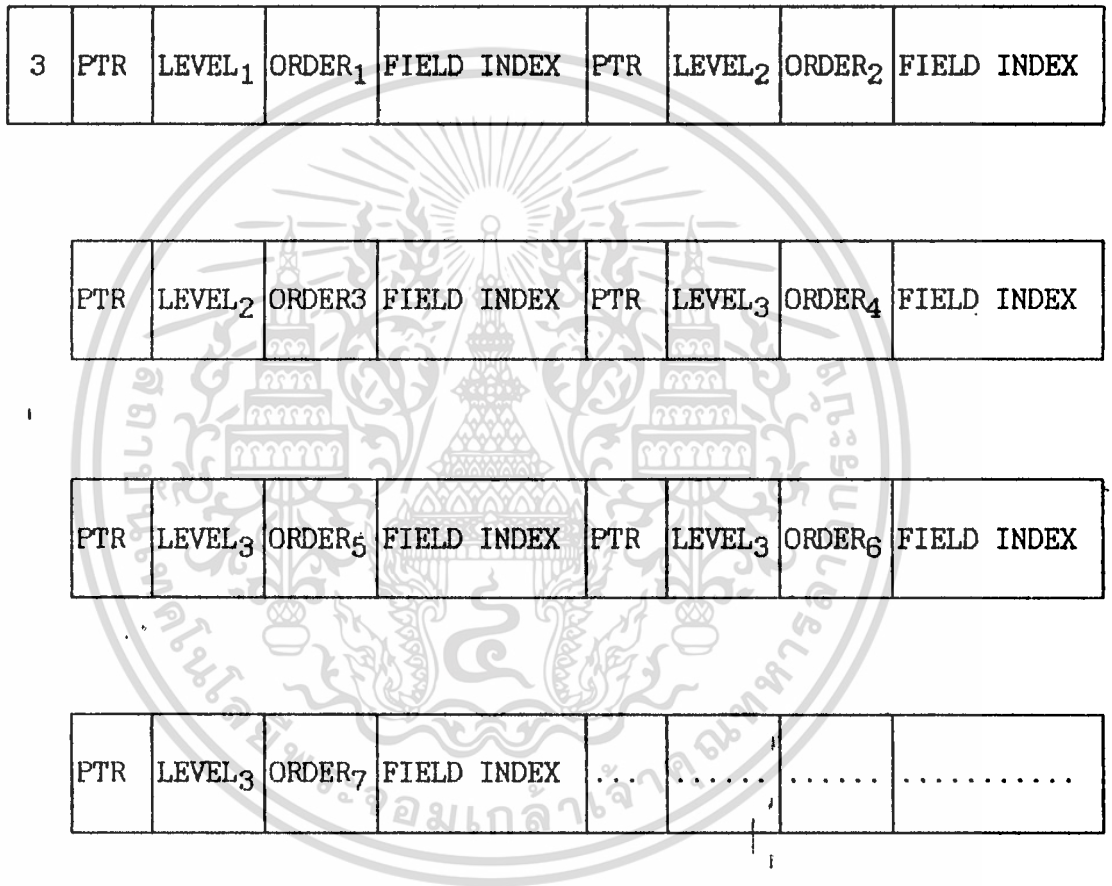
นำข้อมูลที่ได้เหล่านี้มาสร้างไบนารีทรี โดยใช้ Age เป็น field index จะได้ไบนารีทรีมีลักษณะดังรูปที่ 11



รูปที่ 4.2 รูปแสดงพอยน์เตอร์,ค่าระดับ และ ค่าประจำตำแหน่งของไบนารีทรี

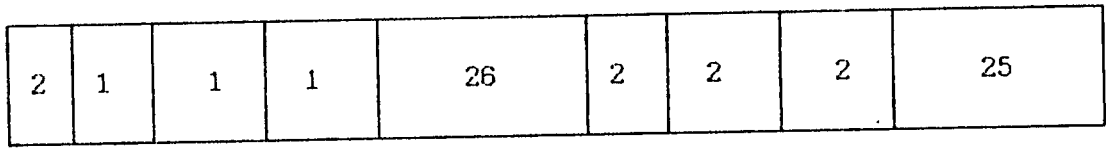
4.4 Data Structure ของ file.bt

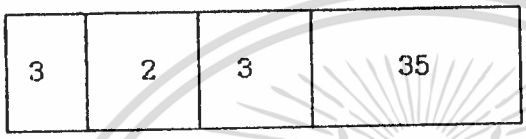
data structure ของ file.bt จะเก็บค่าระดับสูงสุดที่ต้น file.bt และต่อด้วยการเก็บค่าในโหนดของไบนารีทรี แต่ละโหนดจะเก็บเฉพาะค่า พอยน์เตอร์(ptr),ค่าระดับในไบนารีทรี(level),ค่าประจำตำแหน่ง(order) และ field index เท่านั้น แสดงได้ดังรูปที่ 12



รูปที่ 4.3 โครงสร้างของ file.bt

จากรูปที่ 11 ไบนารีทรีที่สร้างได้แล้วมาเก็บใน file.bt ตาม data structure ที่ได้อธิบายไว้แล้วข้างต้น จะได้โครงสร้างของ file.bt ดังรูปที่ 13





รูปที่ 4.4 รูปแสดงการเก็บข้อมูลใน file.bt

บทที่ 5

โครงสร้างของไฟล์อินเด็กซ์(FILE.NDX)

5.1 file.ndx คืออะไร

file.ndx คือ ไฟล์ที่ทำหน้าที่เก็บเฉพาะ field index ที่ทำการ sort แล้ว แบบอินออร์เดอร์ และพอยน์เตอร์ที่อ้างถึงแอดเดรสเริ่มต้นเรคอร์ดใน file.dbf ที่มีค่าในฟิลด์เท่ากับ ค่าใน field index การจะสร้าง file.ndx นี้ จะสร้างได้ก็ต่อเมื่อ มีการเปิด file.dbf แล้ว

5.2 Data Structure ของ file.ndx

Data structure ของ file.ndx จะมี 2 ส่วนคือ ส่วนที่เป็นเฮดเดอร์ ซึ่งจะเก็บลำดับที่(โดยให้ฟิลด์แรกเป็นฟิลด์ที่ 0),ชนิด และความกว้างของ field index ส่วนที่สองคือ ส่วนที่ทำหน้าที่เก็บพอยน์เตอร์ และข้อมูลของ field index ซึ่งสามารถแสดงได้ดังรูปที่ 14

NO. TYPE WIDTH	PTR	FIELD INDEX$_1$	PTR	FIELD INDEX$_2$	...

รูปที่ 5.1 โครงสร้างของ file.ndx

จากตัวอย่างข้างบน และจากรูปที่ 11 หลังจากที่ sort ข้อมูลแบบอินออร์เดอร์แล้ว จะได้โครงสร้างของ file.ndx จะมีลักษณะดังรูปที่ 15

2 C 3	2	25	1	26	3	35

รูปที่ 5.2 โครงสร้างของ file.ndx หลังจาก sort ข้อมูลแล้ว

file.ndx จะถูก update ตาม file.dbf ถ้าผู้ใช้เลือกที่จะจัดการกับข้อมูลแบบ index นั่นคือ เป็นการกระทำการใด ๆ กับข้อมูลบนไบนารีทรี ซึ่งได้แก่ การเพิ่ม หรือ ลบ เรคอร์ด เพราะเมื่อผู้ใช้ทำการเพิ่มโหนด field index ของเรคอร์ดจะต้อง traverse ไปบนไบนารีทรี และทำการเพิ่มโหนดนั้นลงในไบนารีทรีให้ถูกตำแหน่ง เพื่อว่าเมื่อให้ไบนารีทรี ผ่านการทำอินออร์เดอร์ แล้วจึงจะได้ข้อมูลที่ sort อย่างถูกต้อง และ file.ndx ก็จะถูก update อย่างถูกต้องด้วย ในทำนองเดียวกัน การลบเรคอร์ดก็จำเป็นต้องมีการปรับเปลี่ยน ไบนารีทรี หลังจากที่ทำการลบโหนดนั้น ไปแล้วเช่นกัน

ในทางตรงกันข้าม ถ้าผู้ใช้ไม่เลือกที่จะจัดการกับข้อมูลแบบ index โดยที่ผู้ใช้ทำการเปิด file.dbf ซึ่งผู้ใช้เคยทำการสร้าง file.ndx ไว้แล้ว ถ้าหากว่า ผู้ใช้มีการกระทำการใด ๆ กับข้อมูล ไม่ว่าจะเป็นการเพิ่ม หรือ ลบเรคอร์ด การกระทำดังกล่าวก็จะไปทำการ update ข้อมูลเฉพาะใน file.dbf เท่านั้น โดยที่ file.ndx จะไม่ถูก update ไปด้วย ซึ่งในขณะนี้ ก็จะถือว่า file.ndx ไม่ match กับ file.dbf แล้ว กล่าวคือ จะถือว่า file.ndx นั้นไม่ ได้เป็นการ sort ข้อมูลของ file.dbf นั้นต่อไป และ ถ้าต้องการจัดการกับข้อมูลใน file.dbf นั้นก็จะต้องทำการเลือกทำฟังก์ชั่น index ซึ่งเปรียบเสมือนว่าเป็นการสร้าง file.ndx ใหม่เลย

บทที่ 6

ฟังก์ชั่นการทำงานของ WinBase

6.1 ฟังก์ชั่นการจัดการเกี่ยวกับไฟล์ข้อมูล (file.dbf)

-open เป็นการที่ผู้ใช้ระบุไฟล์ข้อมูล (file.dbf) ที่ต้องการติดต่อ และ file.dbf นั้นจะต้องมีการสร้างไว้แล้ว หลังจากที่ฟังก์ชัน open ได้รับชื่อ file.dbf แล้วฟังก์ชันนี้ก็จะทำการโหลดเฮดเดอร์ทั้ง เฮดเดอร์ที่เป็นรายละเอียดเกี่ยวกับไฟล์ข้อมูล และเฮดเดอร์ที่เป็นรายละเอียดเกี่ยวกับฟิลด์มาไว้ในหน่วยความจำ รวมถึงการเตรียมพร้อมในการติดต่อกับไฟล์เมโมด้วย ถ้า file.dbf นั้นมีการใส่ข้อมูลที่เกี่ยวกับฟิลด์เมโม แต่ถ้าใน file.dbf นั้นไม่มีใส่ข้อมูลในฟิลด์เมโมแล้วก็จะทำการโหลดเฮดเดอร์ของ file.dbf เท่านั้น นอกจากนี้อาจมีการเปิด file.bt และ file.ndx ด้วย ถ้าผู้ใช้ต้องการติดต่อกับข้อมูลใน file.dbf แบบ index การเปิด file.bt ก็จะทำการโหลดไบนารีทรีที่เก็บไว้ใน file.bt มาไว้ในหน่วยความจำด้วย ซึ่งจะสามารถทราบ ชื่อ,ชนิด และ ความกว้างของ field index ได้จาก file.ndx

-close เป็นฟังก์ชั่นที่ผู้ใช้ทำการยกเลิกการติดต่อกับไฟล์ทั้งหมดที่เปิดอยู่ในขณะนั้น แล้วแต่ว่าการเปิด file.dbf ในขณะนั้นผู้ใช้ทำการติดต่อกับไฟล์อะไรบ้าง โดยฟังก์ชั่นนี้จะรับเฉพาะชื่อ file.dbf ที่ต้องการปิดเท่านั้น ในส่วนของฟังก์ชั่น close นี้จะทำการ clear ค่าตัวแปรทั้งหมดในหน่วยความจำด้วย

-print เป็นฟังก์ชั่นการพิมพ์ข้อมูลทั้งหมดใน file.dbf ที่เปิดอยู่ในปัจจุบัน โดยพิมพ์ออกทางพรินเตอร์

-quit เป็นฟังก์ชั่นการยกเลิกการทำงานของ WinBase โดยจะกลับเข้าสู่เมนูของ window

6.2 ฟังก์ชั่นการสร้างไฟล์ข้อมูล (file.dbf)

-create เป็นฟังกชั่นการสร้างไฟล์ข้อมูล(file.dbf)ใหม่ โดยฟังก์ชั่นนี้จะทำการรับชื่อไฟล์ข้อมูล แล้วจะทำการเก็บเฮดเดอร์ที่เป็นรายละเอียดเกี่ยวกับไฟล์ข้อมูลนั้น และถ้ามีการกำหนดรายละเอียดเกี่ยวกับฟิลด์ของเรคอร์ดใน file.dbf นั้น ก็จะทำการเก็บเฮดเดอร์ที่เป็นรายละเอียดเกี่ยวกับฟิลด์ต่อท้ายเฮดเดอร์ในส่วนแรก และถ้ามีการใส่ข้อมูลที่เป็นเรคอร์ด ก็จะทำการเก็บข้อมูลเหล่านั้นต่อท้ายเฮดเดอร์ในส่วนที่สอง

6.3 ฟังก์ชั่นการอัพเดท (update) file.dbf

-append เป็นฟังก์ชั่นการเพิ่มเรคอร์ด (insert) ใน file.dbf โดยฟังก์ชั่นนี้จะทำการเอาเรคอร์ดที่เพิ่มเข้าไปใหม่นั้นไปต่อท้าย file.dbf และทำการเพิ่มจำนวนเรคอร์ดในส่วนเฮดเดอร์ที่เป็นรายละเอียดเกี่ยวกับไฟล์

-replace เป็นฟังก์ชั่นการเปลี่ยนข้อมูลเก่าในฟิลด์ที่ระบุ ให้เป็นข้อมูลใหม่ในฟิลด์นั้นตามที่ต้องการ โดยสามารถทำการ replace ข้อมูลใน field index ได้ด้วย และจะมีการ update ไบนารีทรีตาม (นั่นคือ จะมีการ update file.ndx และ file.bt) ถ้าผู้ใช้เลือกที่จะจัดการกับข้อมูลใน file.dbf แบบ index

ในส่วนของฟังก์ชั่นนี้ เมื่อผู้ใช้ไม่จัดการกับข้อมูลใน file.dbf แบบ index ผู้ใช้สามารถเลือกทำฟังก์ชั่นนี้ได้ 3 กรณีคือ

1. ทำการ replace ข้อมูลในฟิลด์ของเรคอร์ดที่พอยน์เตอร์ชี้อยู่ปัจจุบัน

เนื่องจากพอยน์เตอร์สามารถชี้ได้ทีละ 1 เรคอร์ดเท่านั้น ดังนั้นถ้ามีการเลือกทำการ replace ตามกรณีนี้แล้วจะสามารถ replace ข้อมูลของฟิลด์ได้ทีละ 1 เรคอร์ดเท่านั้น โดยจะให้พอยน์เตอร์ seek ไปยังฟิลด์ที่ต้องการ replace ในเรคอร์ดที่พอยน์เตอร์ปัจจุบันชี้อยู่ จากนั้นทำการลบข้อมูลเก่าที่อยู่ในฟิลด์นั้นก่อนที่จะทำการเขียนข้อมูลใหม่ ทั้งนี้เพื่อป้องกันการมีข้อมูลเก่าหลงเหลืออยู่ในฟิลด์นั้น ถ้าข้อมูลใหม่สั้นกว่าข้อมูลเก่า

2. ทำการ replace ข้อมูลในฟิลด์ของเรคอร์ดที่กำหนดให้

หลักการในการทำฟังก์ชั่น replace ในกรณีนี้ คือ ต้องให้พอยน์เตอร์ชี้ไปยังเรคอร์ดที่กำหนดให้ก่อน จากนั้นจึงให้พอยน์เตอร์ seek ไปยังฟิลด์ที่ต้องการ replace แล้วทำเช่นเดียวกับกรณีแรก

3. ทำการ replace ข้อมูลในฟิลด์ของเรคอร์ดที่ตรงตามเงื่อนไข (condition) ที่ผู้ใช้เป็นผู้กำหนดให้

ในการทำฟังก์ชั่น replace ในกรณีนี้จะเป็นประโยชน์ในกรณีที่ผู้ใช้อาจจำลำดับที่ของเรคอร์ดใน file.dbf ไม่ได้ หรือผู้ใช้ต้องการ replace ฟิลด์ข้อมูลมากกว่า 1 เรคอร์ด เพียงแต่ผู้ใช้กำหนด query ในการทำการฟังก์ชั่น replace เท่านั้น แล้วในส่วนของฟังก์ชั่นนี้จะทำการหาพอยน์เตอร์ของเรคอร์ดที่ตรงตาม query ที่ผู้ใช้กำหนดมาให้ก่อน โดยอาจได้พอยน์เตอร์หนึ่ง หรือมากกว่าหนึ่งเรคอร์ด นั่นคือ ผู้ใช้สามารถทำการ replace ข้อมูลในฟิลด์ได้มากกว่า 1 เรคอร์ดนั่นเอง หลักการในการทำฟังก์ชั่นในกรณีนี้ ก็เช่นเดียวกับกรณีที่ 2 เพียงแต่ในกรณีนี้สามารถการทำการ replace ข้อมูลในฟิลด์ได้มากกว่า 1 เรคอร์ด

หลักการในการทำฟังก์ชั่นนี้ เมื่อผู้ใช้เลือกที่จะจัดการข้อมูลใน file.dbf แบบ index สามารถแบ่งได้เป็น 3 กรณีและมีหลักการในการทำฟังก์ชั่นนี้ เช่นเดียวกับที่ได้กล่าวมาแล้วข้างต้น แต่จะแตกต่างกันคือ ก่อนที่จะทำการ replace ข้อมูลเก่าของ field index ด้วยข้อมูลใหม่ของ field index จะต้องมีการเช็คก่อนว่าข้อมูลใหม่ที่ต้องการจะทำการ replace นั้นจะต้องไม่ซ้ำกับข้อมูลใน field index ของเรคอร์ดอื่น (นั่นคือจะต้องเช็คก่อนว่าถ้ามีการเปลี่ยนแปลงข้อมูลใน field index แล้ว field index นั้นต้อง unique เหมือนเดิม)

-delete เป็นฟังก์ชั่นที่ทำการ mark เครื่องหมายดอกจันไว้หน้าเรคอร์ดที่ต้องการลบ แต่จะยังไม่มีการลบเรคอร์ดออกจริง ๆ จนกว่าจะได้รับการ confirm จากการทำฟังก์ชั่น pack

ในส่วนของฟังก์ชั่นนี้สามารถแบ่งได้เป็น 3 กรณี คือ

1. ทำการ delete ข้อมูลในฟิลด์ของเรคอร์ดที่พอยน์เตอร์ชี้อยู่ปัจจุบัน

เนื่องจากพอยน์เตอร์สามารถชี้ได้ทีละ 1เรคอร์ดเท่านั้น ดังนั้นถ้ามีการเลือกทำฟังก์ชั่น delete ตามกรณีนี้แล้ว จะสามารถ delete ข้อมูลได้ทีละ 1 เรคอร์ดเท่านั้น โดยจะจะทำการ mark เครื่องหมายดอกจันไว้หน้าเรคอร์ดที่พอยน์เตอร์ปัจจุบันชี้อยู่

2. ทำการ delete ข้อมูลในฟิลด์ของเรคอร์ดที่กำหนดให้

หลักการในการทำฟังก์ชั่น delete ในกรณีนี้ คือ ต้องให้พอยน์เตอร์ชี้ไปยังเรคอร์ดที่ต้องการ delete ก่อน แล้วทำเช่นเดียวกับกรณีแรก

3. ทำการ delete ข้อมูลในฟิลด์ของเรคอร์ดที่ตรงตามเงื่อนไข (condition) ที่ผู้ใช้เป็นผู้กำหนดให้

ในการทำฟังก์ชั่น delete ในกรณีนี้จะเป็นประโยชน์ในกรณีที่ผู้ใช้อาจจำลำดับที่ของเรคอร์ดใน file.dbf ไม่ได้ หรือผู้ใช้ต้องการ delete ข้อมูลมากกว่า 1 เรคอร์ด เพียงแต่ผู้ใช้กำหนด query ในการทำการฟังก์ชั่น delete เท่านั้น แล้วในส่วนของฟังก์ชั่นนี้ จะทำการหาพอยน์เตอร์ของเรคอร์ดที่ตรงตาม query ที่ผู้ใช้กำหนดมาให้ก่อน โดยอาจได้พอยน์เตอร์หนึ่งหรือมากกว่าหนึ่งเรคอร์ด นั่นคือผู้ใช้สามารถทำการ mark เครื่องหมายดอกจันไว้หน้าเรคอร์ดที่ต้องการลบได้มากกว่า 1 เรคอร์ดนั่นเอง หลักการในการทำฟังก์ชันในกรณีนี้ ก็เช่นเดียวกับกรณีที่2 เพียงแต่ในกรณีนี้สามารถการทำการ mark เครื่องหมายดอกจันไว้หน้าเรคอร์ดได้มากกว่า 1 เรคอร์ด

หลักการในการทำฟังก์ชั่นนี้ ถ้ามีการจัดการกับข้อมูลใน file.dbf แบบindex แล้ว file.bt และ file.ndx จะยังคงไม่มีการ update ข้อมูล เนื่องจากยังไม่มีการลบเรคอร์ดออกเลยเพียงแต่ทำการ mark เครื่องหมายดอกจันไว้หน้าเรคอร์ดใน file.dbf เท่านั้น

-recall เป็นฟังก์ชั่นที่ทำการลบเครื่องหมายดอกจันที่อยู่หน้าเรคอร์ดที่ผ่านการทำฟังก์ชั่น delete แล้ว โดยในส่วนของฟังก์ชั่นนี้สามารถแบ่งได้เป็น 3 กรณี และหลักการในการทำฟังก์ชั่น recall จะทำเช่นเดียวกับฟังก์ชั่น delete ต่างกันก็ตรงที่ฟังก์ชั่น delete จะทำการ mark เครื่องหมายดอกจันไว้หน้าเรคอร์ดที่ต้องการลบ แต่ฟังก์ชั่น recall จะทำการลบเครื่องหมายดอกจันออก

-pack เป็นฟังก์ชั่นที่ทำการลบเรคอร์ดใน file.dbf ที่มีเครื่องหมายดอกจันอยู่ข้างหน้า โดยในส่วนของฟังก์ชั่นนี้สามารถแบ่งได้เป็น 3 กรณีเช่นเดียวกับฟังก์ชั่น delete โดยในการทำฟังก์ชั่นนี้ในแต่ละกรณี ต้องมีการเช็คก่อนว่าเรคอร์ดที่ต้องการลบนั้น มีการ mark เครื่องหมายดอกจันไว้หน้าเรคอร์ดหรือไม่ และหลังจากที่มีการทำฟังก์ชั่น pack แล้ว จะทำการ compress file.dbf ด้วย เพื่อให้มีการ update ขนาดของ file.dbf นอกจากนี้ถ้ามีการจัดการกับข้อมูลใน file.dbf แบบ index แล้ว หลังจากที่มีการ update file.dbf แล้วก็จะทำการ update file.bt และ file.ndx ด้วย

6.4 ฟังก์ชั่นการอ้างอิงตำแหน่งของเรคอร์ดในไฟล์ข้อมูล

-first เป็นฟังก์ชันที่ให้พอยน์เตอร์ชี้ไปยังเรคอร์ดแรกของ file.dbf ที่เปิดอยู่ในปัจจุบัน โดยในส่วนของฟังก์ชั่นนี้จะทำหน้าที่ให้พอยน์เตอร์ seek ไปยังจุดเริ่มต้นข้อมูลในส่วนที่ต่อจากเฮดเดอร์ของ file.dbf โดยเราสามารถทราบความยาวของเฮดเดอร์ได้หลังจากที่มีการโหลดเฮดเดอร์มาไว้ในหน่วยความจำแล้ว(นั่นคือ มีการเรียกฟังก์ชั่น open แล้วนั่นเอง)

-bottom เป็นฟังก์ชั่นที่ให้พอยน์เตอร์ชี้ไปยังเรคอร์ดสุดท้ายของ file.dbf ที่เปิดอยู่ในปัจจุบัน โดยในส่วนของฟังก์ชั่นนี้จะทำหน้าที่ให้พอยน์เตอร์ seek ไปยังจุดสิ้นสุดของ file.dbf ก่อน แล้วหลังจากนั้นก็จะให้พอยน์เตอร์ seek ถอยหลังไปอีกเท่ากับความยาวของเรคอร์ด 1 เรคอร์ด โดยเราสามารถทราบความยาวของเรคอร์ดแต่ละเรคอร์ดได้ หลังจากที่มีการโหลดเฮดเดอร์มาไว้ในหน่วยความจำแล้ว

-goto เป็นฟังก์ชั่นที่ให้พอยน์เตอร์ชี้ไปยังเรคอร์ดที่ต้องการได้เลย โดยบอกลำดับที่ของเรคอร์ดที่ต้องการชี้ไป ในส่วนของฟังก์ชั่นนี้จะทำการคำนวณก่อน โดยใช้ลำดับที่ของเรคอร์ดที่ต้องการชี้ไป คูณกับความยาวของเรคอร์ด หลังจากนั้นก็จะให้พอยน์เตอร์เริ่มต้น seek จากจุดเริ่มต้นข้อมูลไปเท่ากับความยาวที่คำนวณได้

-next เป็นฟังก์ชั่นที่ให้พอยน์เตอร์ชี้ไปยังเรคอร์ดถัดไปจากเรคอร์ดที่พอยน์เตอร์ชี้อยู่ปัจจุบัน ในส่วนของฟังก์ชั่นนี้จะให้พอยน์เตอร์ seek ต่อจากพอยน์เตอร์ปัจจุบันไปเท่ากับความยาวของเรคอร์ด

-previous เป็นฟังก์ชั่นที่ทำให้พอยน์เตอร์ชี้ไปยังเรคอร์ดก่อนเรคอร์ดที่พอยน์เตอร์ปัจจุบันชี้อยู่ ในส่วนของฟังก์ชั่นนี้จะทำให้พอยน์เตอร์ seek ถอยหลังจากพอยน์เตอร์ปัจจุบันไปเท่ากับ

-skip เป็นฟังก์ชั่นที่ทำให้พอยน์เตอร์ชี้ไปยังเรคอร์ดที่ข้ามจากเรคอร์ดที่พอยน์เตอร์ชี้อยู่ปัจจุบันไปเท่ากับจำนวนเรคอร์ดที่ระบุ ในส่วนของฟังก์ชั่นนี้จะทำการคำนวณก่อน โดยใช้จำนวนเรคอร์ดที่ต้องการข้ามไป คูณกับความยาวเรคอร์ด หลังจากนั้นก็ให้พอยน์เตอร์เริ่มต้น seek จากเรคอร์ดที่ชี้อยู่ปัจจุบันไปเท่ากับ ค่าที่คำนวณได้

6.5 ฟังก์ชั่นการดึง (retrieve) ข้อมูล

-list เป็นฟังก์ชั่นที่ทำการแสดงข้อมูลทุก ๆ เรคอร์ดใน file.dbf โดยสามารถทำการเลือกที่จะแสดงฟิลด์ใดก็ได้ในแต่ละเรคอร์ด หลักการในการทำฟังก์ชั่นนี้คือ ให้พอยน์เตอร์ seek ไปยังฟิลด์ที่ต้องการแสดง แล้วทำการดึงข้อมูลในฟิลด์นั้นๆ ทุกๆ เรคอร์ดใน file.dbf ออกมาแสดงทางหน้าจอ

-display เป็นฟังก์ชั่นที่ทำหน้าที่เหมือนกับฟังก์ชั่น list แต่จะแสดงทีละเรคอร์ด

-count เป็นฟังก์ชั่นที่บอกถึงจำนวนเรคอร์ดทั้งหมดที่มีอยู่ใน file.dbf โดยเราสามารถทราบจำนวนเรคอร์ดได้จาก เฮดเดอร์ที่เป็นรายละเอียดเกี่ยวกับไฟล์ใน file.dbf

6.6 ฟังก์ชั่นการทำออร์แกนไนซ์ (organize)

-index เป็นฟังก์ชันในการสร้างไบนารีทรี โดยแต่ละโหนดในไบนารีทรีจะใช้เฉพาะข้อมูลใน field index เท่านั้น และจะทำการเก็บไบนารีทรีนี้ไว้ใน file.bt ส่วน file.ndx จะทำการเก็บเฉพาะพอยน์เตอร์ที่อ้างถึงเรคอร์ดใน file.dbf ที่มีค่าในฟิลด์เท่ากับ field index และข้อมูลใน field index โดย file.bt และfile.ndx นี้จะมีการ update เมื่อมีการจัดการและมีการเปลี่ยนแปลงข้อมูลใน file.dbf แบบ index

-sort เป็นฟังก์ชั่นในการสร้าง file.dbf ขึ้นมาใหม่อีกไฟล์หนึ่ง โดยไฟล์ที่สร้างขึ้นมาใหม่นี้ จะเป็นไฟล์ข้อมูลที่มีเรคอร์ดเรียงกันตาม field index จากน้อยไปหามาก แต่ในกรณีการทำฟังก์ชั่นนี้ จะอนุญาตให้มี field index ที่ซ้ำกันได้ในแต่ละเรคอร์ด ในส่วนของฟังก์ชั่นนี้จะทำการรับ field index มาทำการสร้างไบนารีทรีก่อน จากนั้นจะทำการ sort แบบอินออร์เดอร์ แล้วเอาพอยน์เตอร์ที่ได้จากการ sort ข้อมูลของ field index นั้นชี้ไปยังตำแหน่งของเรคอร์ดใน file.dbf ที่มีพอยน์เตอร์ตรงกับพอยน์เตอร์ในโหนด ต่อจากนั้นก็จะทำการคัดลอกข้อมูลทั้งเรคอร์ดใน file.dbf นั้นลงใน file.dbf ไฟล์ใหม่ ทำเช่นนี้ไปเรื่อย ๆ จนจบ file.dbf

-copy structure เป็นฟังก์ชั่นในการสร้าง file.dbf ขึ้นมาใหม่อีกไฟล์หนึ่ง ซึ่งเฮดเดอร์ของ file.dbf ไฟล์ใหม่นี้จะเหมือนกับ เฮดเดอร์ของ file.dbf ที่เปิดอยู่ปัจจุบัน และ file.dbf ไฟล์ใหม่นี้จะมีเฉพาะข้อมูลส่วนที่เป็นเฮดเดอร์เท่านั้น ในส่วนของฟังก์ชั่นนี้จะทำการคัดลอกข้อมูลของเฮดเดอร์ทั้งหมดที่อยู่ในหน่วยความจำลงในส่วนเฮดเดอร์ของ file.dbf ไฟล์ใหม่

6.7 ฟังก์ชั่นการค้นหาข้อมูล (query)

-query เป็นฟังก์ชั่นในการค้นหาข้อมูลใน file.dbf ที่เปิดอยู่ในปัจจุบันเท่านั้น โดยสามารถระบุฟิลด์ที่ต้องการให้แสดงออกทางหน้าจอได้ ในส่วนของฟังก์ชั่นนี้จะทำการหาพอยน์เตอร์ของเรคอร์ดใน file.dbf ที่มีข้อมูลในฟิลด์ตรงกับเงื่อนไข (condition) ใน query แล้วเอาพอยน์เตอร์ที่หาได้ ชี้ไปยัง file.dbf จากนั้นทำการดึงข้อมูลในเรคอร์ดนั้นทั้งเรคอร์ดหรือดึงเฉพาะบางฟิลด์ที่ผู้ใช้ต้องการดูก็ได้ และในกรณีที่ใน query ประกอบด้วย subquery มากกว่าหนึ่งแล้ว และความสัมพันธ์ระหว่าง subquery อาจจะเป็นการ and หรือ or กัน ในส่วนของฟังก์ชั่นนี้ก็จะทำการหาพอยน์เตอร์ของเรคอร์ดใน file.dbf ที่มีข้อมูลในฟิลด์ตรงกับเงื่อนไขในsubquery โดยจะทำการหาพอยน์เตอร์ของเรคอร์ดใน file.dbf ที่มีข้อมูลในฟิลด์ตรงตามเงื่อนไข ทุก ๆ subqueries จากนั้นก็จะนำเอาพอยน์เตอร์ของแต่ละ subquery ที่หาได้มาทำการ and หรือ or กัน ถ้าเป็นการ and กันก็จะเอาพอยน์เตอร์ของแต่ละ subquery มาทำ intersection กัน แต่ถ้าเป็นการ or กัน ก็จะเอาพอยน์เตอร์ของแต่ละ subquery มาทำการ union กัน ผลลัพธ์ที่ได้จากการ and หรือ or กันนี้จะเป็นพอยน์เตอร์ที่ชี้ไปยังเรคอร์ดใน file.dbf เพื่อทำการดึงข้อมูลในส่วนที่ผู้ใช้ต้องการออกมาแสดงทางหน้าจอ

6.8 ฟังก์ชั่นที่เกี่ยวกับดอส (tool)

-set drive เป็นฟังก์ชั่นที่ทำการ set current drive

-copy file เป็นฟังก์ชั่นที่ทำการคัดลอกไฟล์จาก source file ไปยัง target file ที่ระบุ

-directory เป็นฟังก์ชั่นที่ทำการแสดงรายชื่อไฟล์ เช่นเดียวกับการเรียก dir ใน dos environment

-rename เป็นฟังก์ชั่นที่ทำการเปลี่ยนชื่อไฟล์จากชื่อหนึ่งไปเป็นอีกชื่อหนึ่ง

-erase เป็นฟังก์ชั่นที่ทำการลบไฟล์ที่ต้องการ

-list structure เป็นฟังก์ชั่นที่ทำการแสดงรายละเอียดเกี่ยวฟิลด์ของเรคอร์ดใน file.dbf(structure ของ file.dbf)

บทที่ 7

หน่วยความจำกับ WinBase

เมื่อเปิด file.dbf ผ่าน ฟังก์ชั่น open โปรแกรมจะทำการโหลดเฮดเดอร์ของ file.dbf เก็บไว้ในหน่วยความจำ ซึ่งจะเป็นรายละเอียดของ file.dbf และถ้ามีการจัดการกับข้อมูลใน file.dbf แบบ index ก็จะมีการเปิด file.ndx ดังนั้นในหน่วยความจำก็จะมีรายละเอียดของฟิลด์ที่เป็นคีย์ ซึ่งได้แก่ ชื่อ,ชนิด และความกว้างของ field index นอกจากนี้จะทำการโหลดไบนารีทรีจาก file.bt มาไว้ในหน่วยความจำด้วย

ค่าต่างๆ ในหน่วยความจำจะมีการเปลี่ยนแปลงเพื่อให้ update ตาม file.dbf ที่เปิดอยู่ และอาจจะถูกนำมาใช้เมื่อมีการเรียกใช้ฟังก์ชั่นใดฟังก์ชั่นหนึ่ง เช่น ฟังก์ชั่น count ก็จะนำเอาจำนวนเรคอร์ดทั้งหมด ซึ่งเป็นส่วนหนึ่งในรายละเอียดของ file.dbf มาแสดงให้ผู้ใช้เห็นทางหน้าจอ เป็นต้น ค่าในหน่วยความจำนี้จะอยู่ในหน่วยความจำจนกระทั่งปิด file.dbf นั้น ค่าทุกค่าจะถูก clear ทั้งหมด รวมทั้งหน่วยความจำที่ทำการ allocate มาใช้ก็จะคืนให้กับเครื่องทั้งหมด

ไบนารีทรี ในหน่วยความจำจะถูกนำมาใช้ เมื่อมีการติดต่อกับ field index เช่น การค้นหาเรคอร์ดที่มี field index เท่ากับค่าคงที่ค่าใดค่าหนึ่ง , การ replace ข้อมูลของ field index จากค่าหนึ่งเป็นอีกค่าหนึ่ง เป็นต้น

ในส่วนของการ allocate หน่วยความจำ เพื่อมาใช้เป็นบัฟเฟอร์ จะต้องมีการ free ทุกครั้ง เมื่อไม่มีการใช้บัฟเฟอร์นั้นอีกแล้ว

บทที่ 8

การประเมินประสิทธิภาพ

8.1 ข้อจำกัดของ WinBase

1. สามารถกำหนดจำนวนฟิลด์ในแต่ละเรคอร์ดได้มากที่สุด 10 ฟิลด์
2. ความกว้างของฟิลด์แต่ละฟิลด์มีความกว้างได้มากที่สุด 99 ตัวอักษร
3. สามารถเก็บเรคอร์ดได้มากที่สุด 1 ล้านเรคอร์ด แต่อย่างไรก็ตามก็ขึ้นอยู่กับความสามารถของเครื่องคอมพิวเตอร์ด้วย
4. การทำ index file (file.ndx) สามารถกำหนด key field ได้ 1 key เท่านั้น
5. การเปิด database file (file.dbf) สามารถเปิดได้เพียงครั้งละ 1 ไฟล์
6. การทำฟังก์ชัน query สามารถทำได้บนไฟล์ ไฟล์เดียวเท่านั้น

8.2 แนวทางในการพัฒนาต่อ

1. พัฒนาโปรแกรมให้สามารถเปิด file.dbf ได้มากกว่า 1 ไฟล์ โดยไม่จำเป็นต้องปิดไฟล์เก่าก่อน และสามารถเลือกที่จะติดต่อกับไฟล์ใดไฟล์หนึ่งที่เปิดอยู่ในขณะนั้นได้
2. ไม่จำกัดจำนวนฟิลด์ในแต่ละเรคอร์ด
3. ไม่จำกัดจำนวนเรคอร์ดใน file.dbf
4. การทำ index file (file.ndx) สามารถกำหนด key field ได้มากกว่า 1 คีย์
5. การทำฟังก์ชัน query ควรทำได้มากกว่า 1 ไฟล์
6. เพิ่มฟังก์ชันการทำงานของ WinBase ให้มากขึ้น อาทิเช่น
   - ฟังก์ชันที่สามารถทำการเปลี่ยนแปลงแก้ไข structure ของ database file ได้
   - ฟังก์ชันในการคำนวณ เช่น sum,average เป็นต้น
   - ฟังก์ชันในการสร้างแบบฟอร์มการกรอกรายการ
   - ฯลฯ

8.3 สรุป

โครงงานนี้เป็นการสร้าง WinBase ซึ่งเป็นส่วนหนึ่งของ DBMS (Database-Management System) ที่มีฟังก์ชั่นการทำงานคล้าย dbase III plus โดยผู้ใช้สามารถติดต่อกับฐานข้อมูลผ่านทางเมนู WinBase มีฟังก์ชั่นการทำงานและขอบเขตการทำงานดังได้กล่าวมาแล้ว

กิตติกรรมประกาศ

ในการทำปริญญานิพนธ์ฉบับนี้ ได้รับความช่วยเหลือจากผู้มีพระคุณหลายท่าน ขอกราบขอบพระคุณ อาจารย์ กฤตวัน เครือตราชู อาจารย์ที่ปรึกษา อาจารย์ ดร. บุญธีร์ เครือตราชู ผู้ซึ่งให้การอนุเคราะห์ทั้งด้านความรู้ แนวทางคำแนะนำ อุปกรณ์ในการทำงาน เสียสละทั้งแรงกาย แรงใจและเป็นผู้ให้กำลังใจในการทำงานขอกราบขอบพระคุณอาจารย์ควบคุมการสอบปริญญานิพนธ์ทุกท่านที่สละเวลาที่มีค่ามาควบคุมการสอบ นอกจากนี้ยังขอขอบคุณเพื่อน ๆ ทุกคนที่ช่วยเหลือและให้กำลังใจมาโดยตลอด

หนังสืออ้างอิง

1. Microsoft Coperation,"Microsoft Windows Software Devlopment Kit Reference version 3.0",1990.
2. Microsoft Coperation,"Microsoft Windows Software Devlopment Kit Installation and Update Guide & Guied to Programming version 3.0",1990.
3. Microsoft Coperation,"Microsoft Windows Software Devlopment Kit Tools & SAA CUA Advanced Interface Design Guied version 3.0",1990.
4. Borland International,"Borland C++ version 2.0 Programmer´s Guied",1990.
5. Borland International,"Borland C++ version 2.0 Library Reference",1990.
6. Borland International,"Borland C++ version 2.0 Getting Starte User´s Guied & Resource toolkit",1990.
7. ซีเอ็ด,"ไมโครคอมพิวเตอร์",ฉบับที่ 64,หน้า 333,2533.
8. Peter Norton´s,"Windows 3.0 Power Programming Techniques", 1990.
9. ประพัฒน์ อุทโยภาส,"เรียนรู้ dbase III plus ด้วยตนเอง",ซีเอ็ด,2530.

ภาคผนวก ก

SOURCE CODE

LISTING INDEX.II

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "defs.h"

FILE *fx,*bt;
int  hfx,hbt;
int  datainv1,datainv2,countqry=0,countdel=0,old_ptr=0;
int  showfield,selectfield[10];
char *disp;
long rec,order,old_maxlevel,replaceptr;
long Tlevel=0L,Torder=1L,MAX_LEVEL;
long len_and=0L,len_or=0L,pr=0L,count1=0L,count2=0L,count_d=0L;
long *Result_qry,*ptree,*ptree1,*ptree2,*Del_c;
struct tnode *temp1,*temp2,*root=NULL,*Ftemp,*Nextroot=NULL;
struct tnode *root_sort=NULL;
struct tnode *lftemp,*rgtemp,**prev,*rtree,*temprt,*temp;
struct tnode *newnode,*Difparleft,*Difparright;
struct tnode *Tempnode,*LTempnode,*RTempnode;
struct rep rp;
struct query q[10];
int  tree_leaf(struct tnode *);
int  NOT();
int  del_cond(int );
int  recall_cond();
int  pack_cond();
int  sort_file(struct tnode *);
int sort_function( const void *, const void *);
char *disp_qry();
```

```
long *Operate_AND(long *,long *);
long *Operate_OR(long *,long *);
long *Equal();
long *Not_Eq();
long *Greater();
long *Less();
long *Greater_Eq();
long *Less_Eq();
long *find_ptr();
long *check_ndx(long ,long *);
struct tnode *get_tnode(struct tnode *);
struct tnode *add_tree_inte(struct tnode *,struct tnode *);
struct tnode *add_tree_char(struct tnode *,struct tnode *);
struct tnode *sort_tree_inte(struct tnode *,struct tnode *);
struct tnode *sort_tree_char(struct tnode *,struct tnode *);
struct tnode *create_tree();
struct tnode *sort_tree();
struct tnode *l_rotate(struct tnode *);
struct tnode *r_rotate(struct tnode *);
struct tnode *rotate_tleft(struct tnode *);
struct tnode *rotate_tright(struct tnode *);
struct tnode *del_tnode_inte(char *,struct tnode *);
struct tnode *del_tnode_char(char *,struct tnode *);
struct tnode *find_tnode(struct tnode *,struct tnode **,char *);
struct tnode *Loadbt();
void Traveltree(long,struct tnode *,struct tnode *,struct tnode *);
void show_tree(struct tnode *);
void Wrbtree(struct tnode *);
void Savebt(struct tnode *);
void del_ndxfile(struct tnode *,long );
void freetree(struct tnode *);
void free_tree(struct tnode *);
```

LISTING UINDEX.H

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "udefs.h"

extern FILE *fx,*bt;
extern int  hfx,hbt;
extern int  datainv1,datainv2,countqry,countdel,old_ptr;
extern int  showfield,selectfield[10];
extern char *disp;
extern long rec,order,old_maxlevel,replaceptr;
extern long Tlevel,Torder,MAX_LEVEL;
extern long len_and,len_or,pr,count1,count2,count_d;
extern long *ptree,*Result_qry,*Del_c,*ptree1,*ptree2;
extern struct tnode *temp1,*temp2,*root,*Ftemp,*Nextroot,*root_sort;
extern struct tnode *lftemp,*rgtemp,**prev,*rtree,*temprt,*temp;
extern struct tnode *newnode,*Difparleft,*Difparright;
extern struct tnode *Tempnode,*LTempnode,*RTempnode;
extern struct rep rp;
extern struct query q[10];
int  NOT();
int  del_cond(int );
int  recall_cond();
int  pack_cond();
int  tree_leaf(struct tnode *);
int sort_file(struct tnode *);
int sort_function( const void *, const void *);
char *disp_qry();
```

```
long *check_ndx(long ,long *);
long *Operate_AND(long *,long *);
long *Operate_OR(long *,long *);
long *Equal();
long *Not_Eq();
long *Greater();
long *Less();
long *Greater_Eq();
long *Less_Eq();
long *find_ptr();
struct tnode *get_tnode(struct tnode *);
struct tnode *add_tree_inte(struct tnode *,struct tnode *);
struct tnode *add_tree_char(struct tnode *,struct tnode *);
struct tnode *sort_tree_inte(struct tnode *,struct tnode *);
struct tnode *sort_tree_char(struct tnode *,struct tnode *);
struct tnode *create_tree();
struct tnode *sort_tree();
struct tnode *l_rotate(struct tnode *);
struct tnode *r_rotate(struct tnode *);
struct tnode *rotate_tleft(struct tnode *);
struct tnode *rotate_tright(struct tnode *);
struct tnode *del_tnode_inte(char *,struct tnode *);
struct tnode *del_tnode_char(char *,struct tnode *);
struct tnode *find_tnode(struct tnode *,struct tnode **,char *);
struct tnode *Loadbt();
void Traveltree(long,struct tnode *,struct tnode *,struct tnode *);
void show_tree(struct tnode *);
void Wrbtree(struct tnode *);
void Savebt(struct tnode *);
void del_ndxfile(struct tnode *,long );
void freetree(struct tnode *);
void free_tree(struct tnode *);
```

LISTING DEFS.H

```
/* FOR DEFINE DATA */

#define MAX_STR     100
#define BLANK_CHAR  ' '          /* Define for blank      */
#define NULL_CHAR   '/0'         /* Define for null value */
#define TRUE        1
#define FALSE       0

char *null_str = "";
char *blank    =" ";

/* FOR BINARY TREE */

struct tnode
{
  struct tnode *left;          /* Left pointer of node      */
  long ptr;                    /* Pointer of data           */
  long level;                  /* Level of node             */
  long order;                  /* Position of node in btree */
  char data[MAX_STR];          /* Field index data          */
  struct tnode *right;         /* Right pointer of node     */
} ;
```

```
struct query
{
  int  cond1;                  /* Condition NOT(1) or blank(0) */
  int  FN;                     /* Fieldname of condition       */
  int  operate;                /* Operation of condition
                                          =       (1)
                                          !=      (2)
                                          >       (3)
                                          <       (4)
                                          >=      (5)
                                          <+      (6)          */
  char FV[30] ;                /* Fieldvalue of condition      */
  int  cond2;                  /* Condition AND(2) or OR(3)    */
} ;

struct rep
{
    char *f_repname ;
    char *oldval;
    char *newval;
};
```

LISTING UDEFS.H

```
/* FOR DEFINE DATA */

#define MAX_STR    100
#define TRUE       1
#define FALSE      0

extern BLANK_CHAR;            /* Define for blank      */
extern NULL_CHAR;             /* Define for null value */
extern char *null_str;
extern char *blank;

/* FOR BINARY TREE */

struct tnode
{
  struct tnode *left;     /* Left pointer of node      */
  long ptr;               /* Pointer of data           */
  long level;             /* Level of node             */
  long order;             /* Position of node in btree */
  char data[MAX_STR];     /* Field index data          */
  struct tnode *right;    /* Rigth pointer of node     */
} ;
```

```
struct query
{
  int cond1;              /* Condition NOT(1) or blank(0) */
  int FN;                 /* Fieldname of condition       */
  int operate;            /* Operation of condition
                                    =       (1)
                                    !=      (2)
                                    >       (3)
                                    <       (4)
                                    >=      (5)
                                    <=      (6)          */
  char FV[30];            /* Fieldvalue of condition      */
  int cond2;              /* Condition AND(2) or OR(3)    */
} ;

struct rep
{
   char *f_repname;
   char *oldval;
   char *newval;
};
```

LISTING HEAD.H

```
#include <stdio.h>
#include <dos.h>

#define MAXFIELD        10          /* Number of maximum field        */
#define MAXFIELDNAME    10          /* Number of maximum fieldname    */
#define FIELDTYPE       1           /* Width of field type            */
#define FIELDWIDE       2           /* Width of field wide            */
#define DECLEN          2           /* Width of dec                   */
#define fieldcount      11          /* Position of fieldcount         */
#define fieldlen        15          /* Length of field detial         */
#define beginmark       10
#define headfile        32          /* Length of header file          */

FILE *fd,*ft,*fm;
int  hfd,hft,hfm;
int  PosFieldNdx,WideFieldNdx,widemem,same=0,q_flag=0;
int  PFieldNdx,PosFieldSort,WFieldNdx,WideFieldSort;
int  i,result,rtrim,length;
int  file_data=0,file_index=0,file_mem=0;
int  length,headlen,total,fieldamt,del=0;
long fieldname,memdel;
long j,reccount,memnum,recnum;
char Code[] = "SU1";                /* Code for database file         */
char *fieldindex;
char filedb[12],filem[12],filendx[12],filebt[12];
char TypeFieldNdx[FIELDTYPE+1];
char TFieldNdx[FIELDTYPE+1],TypeFieldSort[FIELDTYPE+1];
char field[MAXFIELD][MAXFIELDNAME+1]; /* Array of field name          */
```

```
char fieldtype[MAXFIELD][FIELDTYPE+1]; /* Array of field type   */
char fieldwidth[MAXFIELD][FIELDWIDE+1];/* Array of field width  */
char fielddec[MAXFIELD][DECLEN+1];     /* Array of field decimal*/
struct date d;

/* FUNCTION FOR CALL */

int createdb(char []);
int read_header(char []);
int index(char []);
int pack_ndx(long );
int check_pointer(long );
int copy_struct(char []);
int sort(char [],char []);
int memo_record(char []);
int list_all(long );
int list_sum(int );
int file_close();
int add_record(char []);
int point_record(long );
int next_record();
int first_record();
int last_record();
int skip_record(long );
int pre_record();
long *Qry(int );
char *mem_display();
char *display();
char *disp_qry();
```

LISTING UHEAD.H

```
#include <stdio.h>
#include <dos.h>

#define MAXFIELD      10        /* Number of maximum field      */
#define MAXFIELDNAME  10        /* Number of maximum fieldname  */
#define FIELDTYPE     1         /* Width of field type          */
#define FIELDWIDE     2         /* Width of field wide          */
#define DECLEN        2         /* Width of dec                 */
#define fieldcount    11        /* Position of fieldcount       */
#define fieldlen      15        /* Length of field detail       */
#define beginmark     10
#define headfile      32        /* Length of header file        */

extern FILE *fd,*ft,*fm;
extern int hfd,hft,hfm;
extern int  PosFieldNdx,WideFieldNdx,widemem,same,q_flag;
extern int  PFieldNdx,PosFieldSort,WFieldNdx,WideFieldSort;
extern int  i,result,rtrim,length;
extern int  file_data,file_index,file_mem,del;
extern int  length,headlen,total,fieldamt;
extern long fieldname,memdel;
extern long j,reccount,memnum,recnum;
extern char Code[];             /* Code for database file       */
extern char *fieldindex;
extern char filedb[12],filem[12],filendx[12],filebt[12];
extern char TypeFieldNdx[FIELDTYPE+1];
extern char TFieldNdx[FIELDTYPE+1],TypeFieldSort[FIELDTYPE+1];
```

```
extern char field[MAXFIELD][MAXFIELDNAME+1];  /* Array of field name */
extern char fieldtype[MAXFIELD][FIELDTYPE+1]; /* Array of field type */
extern char fieldwidth[MAXFIELD][FIELDWIDE+1];/* Array of field width */
extern char fielddec[MAXFIELD][DECLEN+1];     /* Array of field decimal *
extern struct date d;

/* FUNCTION FOR CALL */

int createdb(char []);
int read_header(char []);
int index(char []);
int pack_ndx(long );
int check_pointer(long );
int copy_struct(char []);
int sort(char [],char []);
int memo_record(char []);
int list_all(long );
int list_sum(int );
int file_close();
int add_record(char []);
int point_record(long );
int next_record();
int first_record();
int last_record();
int skip_record(long );
int pre_record();
long *Qry(int );
char *mem_display();
char *display();
char *disp_qry();
```

```
                        ************************
                        *     PROGRAM ADD.C    *
                        ************************
#include <Windows.h>
#include <io.h>
#include <string.h>
#include "uhead.h"
#include "uindex.h"

extern HWND hwnd;

int read_header(char filename[])
{
  struct ftime f1,f2;
  char buf2[3],buf3[4],buf4[5],buf6[7];
  OFSTRUCT ofstruct;

  /* Open database file */

  strcpy(filedb,filename);
  strcpy(filem,filename);
  strcat(filedb,".dbf");
  strcat(filem,".mem");

  if ((hfd = OpenFile(filedb,&ofstruct,OF_EXIST)) == -1)
     MessageBox(hwnd,
                (LPSTR) "File not exist.",
                (LPSTR) "File",
                MB_OK);
```

```
if((hfd=OpenFile(filedb,&ofstruct,OF_READWRITE)) == -1)
{
   MessageBox(hwnd,
              (LPSTR) "Can't open database file.",
              (LPSTR) "File",
              MB_OK);
   return 1;
}

/* ------------------ Read detail of file ------------------ */

lseek(hfd,0L,SEEK_SET);
read(hfd,buf3,strlen(Code));
buf3[strlen(Code)] = '\0';

if (strcmp(buf3,Code) != 0)
{
   MessageBox(hwnd,
              (LPSTR) "This file is not database file.",
              (LPSTR) "File",
              MB_OK);
   close(hfd);
   return 1;
}
lseek(hfd,fieldcount,SEEK_SET);
read(hfd,buf2,2);
buf2[2] = '\0';
fieldamt = atoi(buf2);       /* Read number of record */
read(hfd,buf6,6);
buf6[6] = '\0';
reccount = atol(buf6);       /* Read reccount */
```

```
read(hfd,buf3,3);
buf3[3] = '\0';
headlen = atoi(buf3);        /* Read width of header file */

read(hfd,buf4,4);
buf4[4] = '\0';
total = atoi(buf4);          /* Read width of record */

read(hfd,buf6,6);
buf6[6] = '\0';
memnum = atol(buf6);         /* Read number of memo record */
rtrim = fieldamt*fieldlen;
fieldname = 32L;

/* ------------------ Read detial of field ----------------- */

for (i=0;i<fieldamt;i++)
{
    lseek(hfd,fieldname,SEEK_SET);
    read(hfd,field[i],MAXFIELDNAME);      /* Read field name */
    field[i][MAXFIELDNAME] = '\0';
    MessageBox(hwnd,
               (LPSTR) field[i],
               (LPSTR) "Field",
               MB_OK);

    read(hfd,fieldtype[i],FIELDTYPE);     /* Read field type */
    fieldtype[i][FIELDTYPE] = '\0';
```

```
        MessageBox(hwnd,
                   (LPSTR) fieldtype[i],
                   (LPSTR) "Field",
                   MB_OK);

        read(hfd,fieldwidth[i],FIELDWIDE);      /* Read field width */
        fieldwidth[i][FIELDWIDE] = '\0';
        MessageBox(hwnd,
                   (LPSTR) fieldwidth[i],
                   (LPSTR) "Field",
                   MB_OK);

        if (stromp(fieldtype[i],"M") == 0)
          widemem = atoi(fieldwidth[i]);
        read(hfd,fielddec[i],DECLEN);      /* Read field decimal */
        fielddec[i][DECLEN] = '\0';
        MessageBox(hwnd,
                   (LPSTR) fielddec[i],
                   (LPSTR) "Field",
                   MB_OK);
        fieldname = fieldname+15L;
    }
    fd = fdopen(hfd,"r+w");
    if (check_mark() == 1)
    {
       file_data= 0;
       close(hfd);
       return 1;
    }
    file_data = 1;
```

```
    if (memnum > 0L && file_mem == 0 && widemem != 0)
    {
       if ((hfm = OpenFile(filem,&ofstruct,OF_READWRITE)) == -1)
       {
          MessageBox(hwnd,
                     (LPSTR) "Can't open memo file.",
                     (LPSTR) "File",
                     MB_OK);
          file_mem = 0;
          return 1;
       }
       getftime(hfd,&f1);
       getftime(hfm,&f2);
       if ((f1.ft_tsec != f2.ft_tsec) || (f1.ft_min != f2.ft_min)
         || (f1.ft_hour != f2.ft_hour) || (f1.ft_day != f2.ft_day)
         || (f1.ft_month != f2.ft_month) || (f1.ft_year != f2.ft_year))
       {
          MessageBox(hwnd,
                     (LPSTR) "Memo file mismatch.",
                     (LPSTR) "File",
                     MB_OK);


                     close(hfm);
                     return 1;
       }
       file_mem = 1;
    }
    return 0;
}
```

```
/*---------------------- Check beginmark ----------------------*/

check_mark()
{
  int beg;

  lseek(hfd,headlen,SEEK_SET);
  beg = getw(fd);
  if (beg != beginmark)
  {
     MessageBox(hwnd,
                (LPSTR) "Beginmark mismatch.",
                (LPSTR) "File",
                MB_OK);
     return 1;
  }
  return 0;
}
```

```
/* ------------------ ADD RECORD DATA ---------------- */

add(char *Record)
{
  char temp[100];
  char b[] = "        ";
  int wide = 0;

  if (file_index == 1)
  {
     i = 0;
     while (i != PosFieldNdx && i<fieldamt)
     {
          wide = wide+atoi(fieldwidth[i]);
          i++;
     }
     /* -------- get fieldindex data to node -------- */
     memcpy(temp,Record+wide,WideFieldNdx);
     temp[WideFieldNdx] = '\0';
     if (insert(temp) == 1)
     {
        same = 0;
        return 1;
     }
  }
  reccount++;        /* INCREMENT RECCOUNT */
  write_header_file(hfd,total,fieldamt);
  lseek(hfd,0L,SEEK_END);
  write(hfd,b,strlen(b));
  write(hfd,Record,strlen(Record));      /* WRITE RECORD TO FILE */
  return 0;
}
```

```
/* ------------ DISPLAY RECORD ------------- */

char *display(long d)
{
  int fieldd;
  char *dis,*bufd;
  char blak[] = "    ";

  if (file_mem == 1)
     fieldd = fieldamt-1;
  else
     fieldd = fieldamt;
  if (q_flag == 0)
     d = check_pointer(d);
  lseek(hfd,headlen+9+((total+7)*(d-1)),SEEK_SET);
  if ((dis = malloc(total+1+(3*(fieldd-1)))) != NULL)
  {
     for (i=0;i<fieldd;i++)
     {
        if (i == 0)
        {
           if ((bufd = malloc(atoi(fieldwidth[i])+1)) != NULL)
           {
              read(hfd,bufd,atoi(fieldwidth[i])+1);
              bufd[atoi(fieldwidth[i])+1] = '\0';
              strcpy(dis,bufd);
           }
        }
```

```
            else
            {
               if ((bufd = malloc(atoi(fieldwidth[i]))) != NULL)
               {
                  read(hfd,bufd,atoi(fieldwidth[i]));
                  bufd[atoi(fieldwidth[i])] = '\0';
                  strcat(dis,bufd);
               }
            }
         free(bufd);
         blak[2] = '\0';
         strcat(dis,blak);
         }
      }
      dis[total+1+(3*(fieldd-1))] = '\0';
      free(dis);
      return (dis);
}

/* ---------------- LIST RECORD ---------------- */

list_all(long l)
{
   long ptr;

   ptr = check_pointer(l);
   lseek(hfd,headlen+9+((total+7)*(ptr-1)),SEEK_SET);
   return 0;
}
```

```
/* ------------- List record for sum -------------- */

list_sum(int countq)
{
  long *ptr,*sort_ptr,*q_ptr;
  long d;
  char *disl;

  if ((ptr = Qry(countq)) == NULL)
  {
     MessageBox(hwnd,
                (LPSTR) "Record not found.",
                (LPSTR) "List sum",
                MB_OK);
     return 1;
  }
  if ((q_ptr = malloc((count1*4)+1)) == NULL)
  {
     MessageBox(hwnd,
                (LPSTR) "Can't display result.",
                (LPSTR) "List sum",
                MB_OK);
     return 1;
  }
  for (j=0L;j<count1;j++)
      *(q_ptr+j) = *(ptr+j);
```

```
  if ((sort_ptr = check_ndx(count1,q_ptr)) == NULL)
  {
     MessageBox(hwnd,
                (LPSTR) "Can't display result.",
                (LPSTR) "List sum",
                MB_OK);
     return 1;
  }
  for (j=0L;j<count1;j++)
      *(q_ptr+j) = *(sort_ptr+j);
  for (j=0L;j<count1;j++)
  {
      d = *(q_ptr+j)+1L;
      disl = display(d);
      MessageBox(hwnd,
                 (LPSTR) disl,
                 (LPSTR) "List sum",
                 MB_OK);
  }
  free(q_ptr);
  return 0;
}
```

```
                        ************************
                        *     PROGRAM BT_CH.C    *
                        ************************

#include <Windows.h>
#include <uhead.h>
#include <uindex.h>
#include <math.h>

/* ------------------------------------------------------------------ */
/*    Routine load b-tree in memory  for managing index data          */
/* ------------------------------------------------------------------ */

struct tnode *Loadbt()
{
 long count,nodeorder,father,l_rtree;
 long diforder,diffather;
 char buf1[2],buf2[3],buf6[7];
 long loop = 0L;

 newnode = get_tnode(newnode);
 lseek(hbt,0L,SEEK_SET);
 read(hbt,buf6,6);              /* READ MAX LEVEL OF BTREE */
 buf6[6] = '\0';
 old_maxlevel=atol(buf6);
 order=(pow(2,old_maxlevel))-1;  /* CALCULATE COUNTER OF TNODE */
 lseek(hbt,6L,SEEK_SET);
 read(hbt,buf6,6);              /* READ POINTER ON DISK */
 buf6[6] = '\0';
 newnode->ptr=atol(buf6);
 lseek(hbt,12L,SEEK_SET);
```

```
read(hbt,buf6,6);              /* READ LEVEL */
buf6[6] = '\0';
newnode->level = atol(buf6);
lseek(hbt,18L,SEEK_SET);
read(hbt,buf6,6);              /* READ ORDER */
buf6[6] = '\0';
newnode->order = atol(buf6);
lseek(hbt,24L,SEEK_SET);
read(hbt,newnode->data,WideFieldNdx);
newnode->data[WideFieldNdx] = '\0';
root=newnode;

for (count=2L;count<=order;count++)
{
 if (loop < reccount-1)
 {
  if (Difparright!=NULL)
  {
     loop++;
     RTempnode=Difparright;
     LTempnode=NULL;
     Difparright=NULL ;
     father=RTempnode->order/2;
     Traveltree(father,LTempnode,RTempnode,root);
     if (loop == (reccount-1))
        return(root);
  }
  if (Difparleft!=NULL)
  {
     loop++;
     LTempnode=Difparleft;
```

```
    l_rtree=0L;
    Difparleft=NULL;
}
else
{
    loop++;
    read(hbt,buf6,6);
    buf6[6]='\0';
    Tempnode = get_tnode(Tempnode);
    Tempnode->ptr = atol(buf6);   /* READ POINTER ON DISK */
    read(hbt,buf6,6);
    buf6[6]='\0';
    Tempnode->level = atol(buf6); /* READ LEVEL */
    read(hbt,buf6,6);
    buf6[6]='\0';
    Tempnode->order = atol(buf6); /* READ ORDER */
    read(hbt,Tempnode->data,WideFieldNdx);
    Tempnode->data[WideFieldNdx] = '\0';
    l_rtree = Tempnode->order % 2;

    /* CHECK TEMPNODE IS LEFT OR RIGHT NODE */
    if (l_rtree == 0L)
       LTempnode=Tempnode;
}

if (l_rtree == 0L)
{
    father = LTempnode->order /2; /* FIND FATHER 'S LTEMPNODE */
    if (feof(bt))
    count=order+1;
```

```
else
{
   if (count+1 <= order )
   {
      count++;
      loop++;
      Tempnode=get_tnode(Tempnode);
      read(hbt,buf6,6);
      buf6[6]='\0';
      Tempnode->ptr=atol(buf6);
      read(hbt,buf6,6);
      buf6[6]='\0';
      Tempnode->level=atol(buf6);
      read(hbt,buf6,6);
      buf6[6]='\0';
      Tempnode->order=atol(buf6);
      read(hbt,Tempnode->data,WideFieldNdx);
      Tempnode->data[WideFieldNdx] = '\0';
      diforder=Tempnode->order % 2;
      if (diforder == 0)
      {
         RTempnode=NULL;
         Difparleft = Tempnode;
         loop=loop-1;
         Traveltree(father,LTempnode,RTempnode,root);
      }
      else
      {
         diffather=Tempnode->order /2;
```

```
                    /* CHECK TEMPNODE IS SAME FATHER */
                    if (father == diffather)
                    {
                       RTempnode=Tempnode;
                       Traveltree(father,LTempnode,RTempnode,root);
                    }
                    else
                    {
                       Difparright=Tempnode;
                       RTempnode=NULL;
                       loop=loop-1;
                       Traveltree(father,LTempnode,RTempnode,root);
                    }
                 }
              }
              else
              {
                 RTempnode=NULL;
                 Traveltree(father,LTempnode,RTempnode,root);
              }
           }
        }
        else
        {
           RTempnode=Tempnode;
           father=RTempnode->order / 2;
           LTempnode=NULL;
           Traveltree(father,LTempnode,RTempnode,root);
        }
     }
```

```
  else
      count = order+1;
 } /* end of 'for' loop */

 return(root);
}

/* ------------------------------------------------------------ */
/*     Routine travel on tree for find parent 's children       */
/* ------------------------------------------------------------ */

void Traveltree(long par,struct tnode *Ltemp,struct tnode *Rtemp,
 struct tnode *tree)
{
  if (tree != NULL)
  {
    if (tree->order == par)
    {
       tree->left =Ltemp;
       tree->right=Rtemp;
    }
    if (tree->left != NULL)
       Traveltree(par,Ltemp,Rtemp,tree->left);
    if (tree->right != NULL)
       Traveltree(par,Ltemp,Rtemp,tree->right);
  }
}
```

```
/* ------------------------------------------------------------ */
/*        Free memory of b-tree after close all files           */
/* ------------------------------------------------------------ */

void free_tree(struct tnode *tree)
{

  if (tree != NULL)
  {
  /* show left subtree first */
  if (tree->left != NULL)
     free_tree(tree->left);

/* -------- Free tree in memory --------*/
  free(tree);

/* ------------------------------------ */
/* show right subtree */
  if (tree->right != NULL)
  free_tree(tree->right);
  }
}

/* ------------------------------------------------------------ */
```

```
                    ************************
                    *     PROGRAM CLOSE.C    *
                    ************************

#include <Windows.h>
#include <io.h>
#include "uhead.h"
#include "uindex.h"

extern HWND hwnd;

/* CLOSE ALL FILE */

file_close()
{
  struct ftime f;
  OFSTRUCT ofstruct;

  if (file_data == 1)
  {
     if (file_mem == 1 || file_index == 1)
     {
        close(hfd);
        if ((hfd = OpenFile(filedb,&ofstruct,OF_READ)) == -1)
        {
           MessageBox(hwnd,
                      (LPSTR) "Can't open database to settime.",
                      (LPSTR) "Close file",
                      MB_OK);
```

```
        MessageBox(hwnd,
                   (LPSTR) "Can't save index file for database.",
                   (LPSTR) "Close file",
        MB_OK);
        return 1;
    }
    getftime(hfd,&f);
    setftime(hfd,&f);
}
if (file_mem == 1)
{
    close(hfm);
    if ((hfm = OpenFile(filem,&ofstruct,OF_READ)) == -1)
    {
        MessageBox(hwnd,
                   (LPSTR) "Can't open memo file to set time.",
                   (LPSTR) "Close file",
                   MB_OK);
        MessageBox(hwnd,
                   (LPSTR) "Can't update memo file.",
                   (LPSTR) "Close file",
                   MB_OK);
      return 1;
    }
    setftime(hfm,&f);
    close(hfm);
    file_mem = 0;
}
```

```
if (file_index == 1)
{
   close(hbt);
   close(hfx);
   if ((hbt = OpenFile(filebt,&ofstruct,OF_READ)) == -1)
   {
      MessageBox(hwnd,
                 (LPSTR) "Can't open btree to set time.",
                 (LPSTR) "Close file",
                 MB_OK);
      return 1;
   }
   if ((hfx = OpenFile(filendx,&ofstruct,OF_READ)) == -1)
   {
      MessageBox(hwnd,
                 (LPSTR) "Can't open index to set time.",
                 (LPSTR) "Close file",
                 MB_OK);
      return 1;
   }
   setftime(hbt,&f);
   setftime(hfx,&f);
   close(hbt);
   close(hfx);
   file_index = 0;
   root = NULL;
   Difparright = NULL;
   Difparleft = NULL;
   fieldindex = " ";
   Nextroot = NULL;
   Tlevel = 0;
```

```
            Torder = 1;
            MAX_LEVEL = 0;
            old_maxlevel = 0;
            Tempnode = NULL;
            RTempnode = NULL;
            count1 = 0L;
            count2 = 0L;
        }
        close(hfd);
        file_data = 0;
    }
    else
    {
        MessageBox(hwnd,
                   (LPSTR) "Have not file open.",
                   (LPSTR) "Close file",
                   MB_OK);
        return 1;
    }
    return 0;
}
```

```
                    ****************************
                    *     PROGRAM COMPRESS.C     *
                    ****************************

#include <Windows.h>
#include "nhead.h"
#include "uindex.h"
#include <sys\stat.h>

extern char filename[];
extern HWND hwnd;

/* ------------------ Sort pointer ------------------*/

int sort_function( const void *a, const void *b)
{
   return(*(int*)a-*(int*)b);
}

/* ---------- Pack database file by determining condition ------- */
pack_cond(countdel)
{
    long n,z,k,ndxptr,*Del_qry;
    char buf6[7],buf[2];
    void *a,*b;

    Del_qry=Qry(countdel);/* SOLVE QRIES TO FIND POINTER TO RECORD */
```

```
if (Del_qry==NULL)/* NO RECORD MATCH QRIES */
{
   MessageBox(hwnd,
              (LPSTR) "Record not found.",
              (LPSTR) "Pack by condition",
              MB_OK);
   return 1;
}
count_d=count1;/* NUMBER OF RECORD THAT MATCH QRIES */
if ((Del_c=malloc((count_d*4)+1))==NULL)
{
   MessageBox(hwnd,
              (LPSTR) "Can't alloc memory.",
              (LPSTR) "Pack cond",
              MB_OK);
   return 1;
}
for (j=0L;j<count_d;j++)
   *(Del_c+j)=*(Del_qry+j);
for (j=0L;j<count_d;j++)
{
    lseek(hfd,headlen+9+(*(Del_c+j)*(total+7)),SEEK_SET);
    read(hfd,buf,1);
    buf[1] = '\0';
    /* Check mark delete record */
    if (strcmp(buf,"*") != 0)
    {
       MessageBox(hwnd,
                  (LPSTR) "Record should been delete before.",
                  (LPSTR) "Pack cond",
                  MB_OK);
```

```
            return 1;
        }
    }
    qsort((void *)Del_c, count_d, sizeof(Del_c[0]), sort_function);

    /* Pack record that match qries */
    for (z=0L;z<count_d;z++)
    {
        k = 0L;
        lseek(hfx,4L,SEEK_SET);
        read(hfx,buf6,6);
        buf6[6] = '\0';
        ndxptr = atol(buf6);
        *(Del_c+z) = *(Del_c+z)-z;
        while (k<reccount && ndxptr != *(Del_c+z))
        {
            k++;
            lseek(hfx,4L+(k*(WideFieldNdx+6)),SEEK_SET);
            read(hfx,buf6,6);
            buf6[6] = '\0';
            ndxptr = atol(buf6);
        }
        pack(k+1);
    }
    return 0;
}
```

```
/* ---------------------- Pack database file ---------------------- */

pack(long recno)
{
  struct stat statbuf;
  long ptr,loop,memrec,rec;
  char *source,buf[2],buf6[6];
  char blanks[] = "       ";
  void *malloc(size_t);

  if (recno > reccount || recno < 1)/* INPUT ERROR */
  {
     MessageBox(hwnd,
                (LPSTR) "Have not delete records.",
                (LPSTR) "Pack",
                MB_OK);
     return 1;
  }
  /* Check pointer to mark file.dbf */
  ptr = check_pointer(recno);
  rec = ptr;
  lseek(hfd,(headlen+9+(rec-1)*(total+7)),SEEK_SET);
  read(hfd,buf,1);
  buf[1] = '\0';
  /* Check mark delete record */
  if (stromp(buf,"*") != 0)
  {
     MessageBox(hwnd,
                (LPSTR) "Record has not ever been delete.",
                (LPSTR) "Pack",
                MB_OK);
```

```
    return 1;
}
reccount--;/* DECREMENT RECCOUNT */
if (reccount > 0)
{
    loop = reccount-ptr+1L;
    length = total+7;
    if (file_mem == 1)/* DATABASE FILE HAS MEMO FILE */
    {
        seek(hfd,(headlen+3+(ptr-1)*(total+7)),SEEK_SET);
        read(hfd,buf6,6);
        buf6[6] = '\0';
        if (strcmp(buf6,blanks) != 0)/* RECORD HAS MEMO RECORD */
        {
            memrec = atol(buf6);
            memdel = memrec;/* MARK MEMO RECORD THAT WILL DELETE */
            mem_pack(memrec);     /* DELETE MEMO RECORD */
            del = 1;/* MARK FOR UPDATE POINTER OF MEMO RECORD */
        }
    }
    /* Delete record in database file */
    for (j=0L;j<loop;j++)
    {
        lseek(hfd,(headlen+3+(ptr+j)*(total+7)),SEEK_SET);
        if ((source = malloc(length)) == NULL)/* ALLOCATE MEMORY FOR DAT
        {
            MessageBox(hwnd,
                       (LPSTR) "Not enough memory.",
                       (LPSTR) "Pack",
                       MB_OK);
            return 1;
        }
```

```
            read(hfd,source,(length));/* READ DATA TO MEMORY */
            source[length] = '\0';
            lseek(hfd,(headlen+3+(ptr+j-1)*(total+7)),SEEK_SET);
            write(hfd,source,length);/* WRITE DATA TO TEMP FILE */
            free(source);
        }
    }
    write_header_file(hfd,total,fieldamt);/* UPDATE HEADER FILE */
    fstat(hfd,&statbuf);
    chsize(hfd,statbuf.st_size-(total+7L));/* UPDATE DATABASE FILE SIZE */

    /* Check index file open? */
    if (file_index == 1)/* TYPE OPEN DATABASE FILE IS INDEX FILE */
    {
        rec = recno;
        pack_ndx(rec-1); /* UPDATE INDEX FILE & BTREE */
    }
    del = 0;
    return 0;
}
```

```
/* ---------- Mark delete record by determining condition ------- */
del_cond(countdel)
{
    long *Del_qry;

    Del_qry=Qry(countdel);/* FIND RECORD THAT MATCH QRIES */
    if (Del_qry==NULL)/* NO RECORD THAT MATCH QRIES */
    {
       MessageBox(hwnd,
                  (LPSTR) "Record not found.",
                  (LPSTR) "Delete by condition",
                  MB_OK);
      return 1;
    }
    count_d=count1;/* NUMBER OF RECORD THAT MATCH QRIES */
    if ((Del_c=malloc((count_d*4)+1))==NULL)
    {
       MessageBox(hwnd,
                  (LPSTR) "Can't alloc memory",
                  (LPSTR) "Delete by condition",
                  MB_OK);
      return 1;
    }
    for (j=0L;j<count_d;j++)
        *(Del_c+j)=*(Del_qry+j);
    del=1;
    for (j=0L;j<count_d;j++)
        delete(*(Del_c+j)+1);
    del=0;
    return 0;
}
```

```
/* ---------------------- Mark delete record ---------------------- */

delete(long recno)
{
  long ptr,rec;
  char buf[2];

  /* Check reccount && recnum */
  if (recno > reccount || recno < 1)
  {
     MessageBox(hwnd,
                (LPSTR) "Have not delete record.",
                (LPSTR) "Delete",
                MB_OK);
     return 1;
  }

  /* Check mark delete record */
  if (del==0)
     ptr = check_pointer(recno);/* FIND REAL POINTER */
  else
     ptr=recno;

  /* Write mark delete record */
  lseek(hfd,(headlen+9+((ptr-1)*(total+7))),SEEK_SET);
  write(hfd,"*",1);
  return 0;
}
```

```
/* ------ Delete mark delete record by determining condition----- */

recall_cond(countdel)
{
   long *Del_qry;
   char buf[2];

   Del_qry=Qry(countdel);/* FIND RECORD THAT MATCH QRIES */
   if (Del_qry==NULL)/* NO RECORD THAT MATCH QRIES */
   {
      MessageBox(hwnd,
                 (LPSTR) "Record not found.",
                 (LPSTR) "Recall by condition",
                 MB_OK);
     return 1;
   }
   count_d=count1;/* NUMBER OF RECORD THAT MATCH QRIES */
   if ((Del_c=malloc((count_d*4)+1))==NULL)
   {
      MessageBox(hwnd,
                 (LPSTR) "Can't alloc memory",
                 (LPSTR) "Recall by condition",
                 MB_OK);
                 return 1;
   }
   for (j=0L;j<count_d;j++)
       *(Del_c+j)=*(Del_qry+j);
   del=1;
```

```
    for (j=0L;j<count_d;j++)/* CHECK RECORD THAT NOT DELETE BEFORE RECALL
    {
        lseek(hfd,(headlen+9+(*(Del_c+j)*(total+7))),SEEK_SET);
        read(hfd,buf,1);
        buf[1] = '\0';

        /* check mark delete record */
        if (strcmp(buf,"*") != 0)
        {
           MessageBox(hwnd,
                      (LPSTR) "Record should been delete before",
                      (LPSTR) "Recall by condition",
                      MB_OK);
           return 1;
        }
    }
    for (j=0L;j<count_d;j++)
       recall(*(Del_c+j)+1);
    del=0;
    return 0;
}
```

```
/* ------------------ Delete mark delete record ----------------- */

recall(long recno)
{
  char buf[2];
  long ptr,rec;

  if (recno > reccount)/* INPUT ERROR */
  {
     MessageBox(hwnd,
                (LPSTR) "No have recall record.",
                (LPSTR) "Recall",
                MB_OK);
     return 1;
  }

  /* Check mark delete record */
  if (del == 0)
     ptr = check_pointer(recno);/* FIND REAL POINTER */
  else
     ptr = recno;
  lseek(hfd,(headlen+9+((rec-1)*(total+7))),SEEK_SET);
  read(hfd,buf,1);
  buf[1] = '\0';
```

```
   /* check mark delete record */
   if (strcmp(buf,"*") != 0)
   {
      MessageBox(hwnd,
                 (LPSTR) "Record has not ever been delete.",
                 (LPSTR) "Recall",
                 MB_OK);
      return 1;
   }
   lseek(hfd,(headlen+9+((ptr-1)*(total+7))),SEEK_SET);
   write(hfd," ",1);/* Delete mark record */
   return 0;
}

/* ----------------- Find real pointer ----------------- */

check_pointer(long recno)
{
   long ptr;
   char buf6[7];

   /* Index file open? */
   if (file_index == 1)
   {
      /* Get pointer for point database file */
      lseek(hfx,4L+((WideFieldNdx+6)*(recno-1)),SEEK_SET);
      read(buf6,6,1,fx);
      buf6[6] = '\0';
      ptr = atol(buf6);
      ptr++;
      recno = ptr;
   }
```

```
    else
       ptr = recno;
    return (ptr);
}

/* --------- Find pointer of index record && sort pointer -------- */

long *check_ndx(long num,long *qrys)
{
    long k,z,ndxptr;
    long *seq_ptr;
    char buf6[7];

    if ((seq_ptr = malloc((num*4)+1)) == NULL)
    {
       MessageBox(hwnd,
                  (LPSTR) "Can't alloc for sort pointer",
                  (LPSTR) "Check index",
                  MB_OK);
       return 1;
    }

    /* Index file open */
    if (file_index == 1)
    {
       for (z=0L;z<num;z++)
       {
           k = 0L;
           lseek(hfx,4L,SEEK_SET);
           read(hfx,buf6,6);
           buf6[6] = '\0';
           ndxptr = atol(buf6);
```

```
  else
     ptr = recno;
  return (ptr);
}

/* --------- Find pointer of index record && sort pointer -------- */

long *check_ndx(long num,long *qrys)
{
    long k,z,ndxptr,
    long *seq_ptr;
    char buf6[7];

    if ((seq_ptr = malloc((num*4)+1)) == NULL)
    {
       MessageBox(hwnd,
                  (LPSTR) "Can't alloc for sort pointer",
                  (LPSTR) "Check index",
                  MB_OK);
       return 1;
    }

    /* Index file open */
    if (file_index == 1)
    {
       for (z=0L;z<num;z++)
       {
           k = 0L;
           lseek(hfx,4L,SEEK_SET);
           read(hfx,buf6,6);
           buf6[6] = '\0';
           ndxptr = atol(buf6);
```

```
            while (k<reccount && ndxptr != *(qrys+z))
            {
                  k++;
                  lseek(hfx,4L+(k*(WideFieldNdx+6)),SEEK_SET);
                  read(hfx,buf6,6);
                  buf6[6] = '\0';
                  ndxptr = atol(buf6);
            }
            *(seq_ptr+z) = k;
      }
   }
   else
   {
      for (j=0L;j<num;j++)
         *(seq_ptr+j) = *(qrys+j);
   }

   /* Sort index rerord pointer */
   qsort((void *)seq_ptr, num, sizeof(seq_ptr[0]), sort_function);
   free(seq_ptr);
   return (seq_ptr);
}
```

```
                    ****************************
                    *      PROGRAM COPY_STR.C    *
                    ****************************

#include <Windows.h>
#include <alloc.h>
#include "uhead.h"
#include "uindex.h"

char *source;
extern HWND hwnd;

/* COPY STRUCTER */

copy_struct(char file_copy[])
{
  int wAction;
  OFSTRUCT ofStruct;

  /* ------------------ Check copy file exist ---------------- */

  strcat(file_copy, ".dbf");
  hft = OpenFile(file_copy,&ofStruct,OF_EXIST);
  if (hft >= 0)
  {
     wAction = MessageBox(hwnd,
                          (LPSTR) "File exist. Overwrite?",
                          (LPSTR) "File",
                          MB_OKCANCEL);
```

```
        if (wAction == 2)
            return 0;
    }
    if ((hft = OpenFile(file_copy,&ofStruct,OF_CREATE)) == -1)
    {
        MessageBox(hwnd,
                   (LPSTR) "Can't create file copy.",
                   (LPSTR) "File",
                   MB_OK);
        return 1;
    }
    ft = fdopen(hft,"w");

    /* WRITE HEADER FILE TO TEMP FILE */
    write_header_file(ft,total,fieldamt);
    list_struc();
    lseek(hft,headfile,SEEK_SET);
    write(hft,source,length);/* WRITE HEADER FIELD TO COPY FILE */
    putw(beginmark,ft);/* WRITE BEGINMARK TO TEMP FILE */
    free(source);
    return 0;
}
```

```
/* ---------------- LIST STRUCTER ---------------- */

list_struc()
{
  length = rtrim;

  /* write header field & data record */
  if ((source = malloc(length)) == NULL)/* ALLOCATE MEMORY FOR DATA */
  {
     MessageBox(hwnd,
                (LPSTR) "Not enough memory",
                (LPSTR) "List Structure",
                MB_OK);
     close(hft);
     return 1;
  }
  lseek(hfd,headfile,SEEK_SET);
  read(hfd,source,length);/* READ HEADER FIELD TO MEMORY */
  source[length] = '\0';
  return 0;
}
```

```
/* ------------------------------------------------------------------ */
/*  Routine display query that user select field name follow          */
/*  queries (max query allow 10 queries )                             */
/* ------------------------------------------------------------------ */

char *disp_qry()
{
   int wide_q=0,fqry_wide,old_wide,k,y;
   char *disp_q,*fieldbuf;
   char space[]="     ";  /* blank 5 */
   long point_qry;

   for (k=0;k<showfield;k++)
   {
     i=selectfield[k];
     wide_q=wide_q+(atoi(fieldwidth[i]))+5;
   }
   if ((disp_q=malloc(wide_q+1))==NULL)
   {
      MessageBox(hwnd,
                 (LPSTR) "Not enough memory",
                 (LPSTR) "Disp_qry",
                 MB_OK);
      return NULL;
   }
   for (k=0;k<showfield;k++)
   {
       j=0;
       wide_q=0;
```

```
        while ((i!=selectfield[k]) && (i< fieldamt))
        {
              wide_q = wide_q + (atoi(fieldwidth[i]));
              i++;
        }
        point_qry = check_pointer(*(Result_qry+j)+1);
        lseek(hfd,((headlen+10)+((*(Result_qry+j))*(total+7))+wide_q),
                    SEEK_SET);
        fqry_wide=atoi(fieldwidth[i]);
        if ((fieldbuf=malloc(fqry_wide+1)) == NULL)
        {
           MessageBox(hwnd,
                      (LPSTR) "Not enough memory",
                      (LPSTR) "Disp_qry",
                      MB_OK);
           return NULL;
        }
        read(hfd,fieldbuf,fqry_wide);
        fieldbuf[fqry_wide]='\0';
        if (k==0)
           strcpy(disp_q,fieldbuf);
        else
           strcat(disp_q,fieldbuf);
        space[5]='\0';
        strcat(disp_q,space);
        free(fieldbuf);
    } /* end of for loop */
    return(disp_q);
}
```

```
                        ***************************
                        *     PROGRAM CREA_DA.C     *
                        ***************************

#define blak  "       "

#include <Windows.h>
#include "uhead.h"

extern char filename[];
extern HWND hwnd;

int createdb(char files[])
{
  int wAction;
  OFSTRUCT ofstruct;

  strcpy(filedb,files);
  strcat(filedb, ".dbf");
  reccount = 0L;            /* INITIAL RECCOUNT */
  memnum = 0L;              /* INITIAL NUMBER OF MEMO RECORD */

  if ((hfd = OpenFile(filedb,&ofstruct,OF_EXIST)) != -1)
  {
     wAction = MessageBox(hwnd,
                          (LPSTR) "File exist. Overwrite Y/N?",
                          (LPSTR) "File",
                          MB_OKCANCEL);
```

```
    if (wAction == 2)
       return 1;
  }
  if ((hfd = OpenFile(filedb,&ofstruct,OF_CREATE)) == -1)
  {
     MessageBox(hwnd,
                (LPSTR) "Can't create datafile.",
                (LPSTR) "File",
                MB_OK);
     return 1;
  }
  return 0;
}

/* --------- SEND STRUCTURE OF DATABASE FILE --------- */

int send_struct(char fields[],int total,int fieldamt)
{
  int wAction;
  char mem_buf[] = "M";

  if (fieldamt == 0)
  {
     wAction = MessageBox(hwnd,
                          (LPSTR) "Empty field name.Try again?",
                          (LPSTR) "File",
                          MB_OKCANCEL);
```

```
        if (wAction == 2)
        {
            close(hfd);
            remove(filedb);
            return 0;
        }
        else
            return 1;
    }
    write_header_file(hfd,total,fieldamt);
    write_header_field(hfd,fields);
    close(hfd);
    read_header(filename);
    for(i=0;i<fieldamt;i++)
        if ((strcmp(fieldtype[i],mem_buf)) == 0)
        {
            widemem = atoi(fieldwidth[i]);
            total = total-widemem;
            write_header_file(hfd,total,fieldamt);
        }
    file_data = 1;
    return 0;
}
```

```
/* ------------ CREATE DATA BASE ------------- */

write_header_file(int h,int total,int fieldamt)
{
  char buf2[3],buf3[4],buf4[5],buf6[7];

 /* --------------- DETAIL ABOUT HEADER FILE -------------- */

  rtrim = fieldamt*15; /* WIDTH FOR FIELD DETAIL */
  getdate(&d);        /* CREATE DATE */
  lseek(h,0L,SEEK_SET);
  write(h,Code,strlen(Code));   /* WRITE CODE TO DATABASE FILE */
  itoa(d.da_day,buf2,10);
  strcat(buf2,blak);
  buf2[2] = '\0';
  write(h,buf2,2);
  itoa(d.da_mon,buf2,10);
  strcat(buf2,blak);
  buf2[2] = '\0';
  write(h,buf2,2);
  itoa(d.da_year,buf4,10);
  strcat(buf4,blak);
  buf4[4] = '\0';
  write(h,buf4,4);
  itoa(fieldamt,buf2,10);
  strcat(buf2,blak);
  buf2[2] = '\0';
  write(h,buf2,2);
  ltoa(reccount,buf6,10);
  strcat(buf6,blak);
  buf6[6] = '\0';
```

```
  write(h,buf6,6);
  headlen = headfile+rtrim;
  itoa(headlen,buf3,10);
  strcat(buf3,blak);
  buf3[3] = '\0';
  write(h,buf3,3);
  itoa(total,buf4,10);
  strcat(buf4,blak);
  buf4[4] = '\0';
  write(h,buf4,4);
  itoa(memnum,buf6,10);
  strcat(buf6,blak);
  buf6[6] = '\0';
  write(h,buf6,6);
  return 0;
}

write_header_field(int h,char *fields)
{
  FILE *f;
  char headfield[150];

 /* ---------------- DETAIL ABOUT HEADER FIELD ---------------- */

  strncpy(headfield,fields,rtrim);
  headfield[rtrim]='\0';
  lseek(h,headfile,SEEK_SET);
  write(h,headfield,strlen(headfield));/* STORE DETAIL OF FIELD */
  f = fdopen(h,"w");
  putw(beginmark,f);
  return 0;
}
```

```
                    ****************************
                    *    PROGRAM INDEX_CH.C    *
                    ****************************

/* ------------------------------------------------------------ */
/* Program for make index file , sort file , replace field,   */
/* create b-tree and save b-tree in disk       */
/* ------------------------------------------------------------ */

#include <Windows.h>
#include "uhead.h"
#include "uindex.h"
#include <io.h>

char   *getdat();
extern char *fieldindex;
extern HWND hwnd;

/* ------------------------------------------------------------ */
/*              Routine for create index file      */
/* ------------------------------------------------------------ */

index(char fileindex[])
{
  struct ftime f;
  char buf1[2],buf2[3];
  char space[]="      ";
  int wAction;
  OFSTRUCT ofStruct;
```

```
strcpy(filendx,fileindex);
strcpy(filebt,fileindex);
strcat(filendx,".ndx");
strcat(filebt,".bt");

/* ----------------- Check file index exist ---------------- */

hfx = OpenFile(filendx,&ofStruct,OF_EXIST);
if (hfx >= 0)
{
   wAction = MessageBox(hwnd,
                        (LPSTR) "File index exist. Overwrite?",
                        (LPSTR) "File Index",
                        MB_OKCANCEL);

   if (wAction == 2)
      return 0;
}

if (checkfieldindex(fieldindex) == 1)/* Check field index */
   return 1;
WideFieldNdx=WFieldNdx;
PosFieldNdx=PFieldNdx;
strcpy(TypeFieldNdx,TFieldNdx);
if ((root = create_tree()) == NULL)/* Index field not unique */
{
   if (same == 1)
      same = 0;
   return 1;
}
```

```
/*------------------- Open file index & file btree---------------- */

if ((hfx = OpenFile(filendx,&ofStruct,OF_CREATE | OF_WRITE)) == -1)
{
   MessageBox(hwnd,
              (LPSTR) "Can't create file index",
              (LPSTR) "File Index",
              MB_OK);
   return 1;
}
lseek(hfx,0L,SEEK_SET);
itoa(PosFieldNdx,buf1,10);
strcat(buf1,space);
buf1[1]='\0';
write(hfx,buf1,1);
itoa(WideFieldNdx,buf2,10);
strcat(buf2,space);
buf2[2]='\0';
write(hfx,buf2,2);
write(hfx,TypeFieldNdx,1);
show_tree(root);
if ((hbt = OpenFile(filebt,&ofStruct,OF_CREATE | OF_WRITE)) == -1)
{
   MessageBox(hwnd,
              (LPSTR) "Can't create file btree",
              (LPSTR) "File Btree",
              MB_OK);
   return 1;
}
Wrbtree(root);
close(hbt);
```

```
close(hfx);
close(hfd);
if ((hfd = OpenFile(filedb,&ofStruct,OF_READWRITE)) == -1)
{
   MessageBox(hwnd,
              (LPSTR) "Can't open database file to set time",
              (LPSTR) "File Database",
              MB_OK);
   return 1;
}
getftime(hfd,&f);
setftime(hfd,&f);
if ((hbt = OpenFile(filebt,&ofStruct,OF_READ)) == -1)
{
   MessageBox(hwnd,
              (LPSTR) "Can't open btree to set time",
              (LPSTR) "File Btree",
              MB_OK);
   return 1;
}
if ((hfx = OpenFile(filendx,&ofStruct,OF_READ)) == -1)
{
   MessageBox(hwnd,
              (LPSTR) "Can't open btree to set time",
              (LPSTR) "File Index",
              MB_OK);
   return 1;
}
setftime(hbt,&f);
setftime(hfx,&f);
close(hbt);
```

```
  close(hfx);
  return 0;
}

/*-----------------------------------------------------------------*/
/* Routine for check field index , when user send fieldname or   */
/* fieldnumber whenever user want create index file or sort file */
/*-----------------------------------------------------------------*/

checkfieldindex(char *fieldnum)
{
  /* SEND INDEX FIELD NAME */
  i = 0;
  result = strcmp(fieldnum,field[i]);
  while (result != 0 && i<fieldamt)
  {
        i++;
        result = strcmp(fieldnum,field[i]);
  }
  if (result == 0)
  {
     strcpy(TFieldNdx,fieldtype[i]);
     PFieldNdx = i;
     WFieldNdx = atoi(fieldwidth[i]);
     MessageBox(hwnd,
                (LPSTR) PFieldNdx,
                (LPSTR) "Position field index",
                MB_OK);
     MessageBox(hwnd,
                (LPSTR) WFieldNdx,
                (LPSTR) "Wide field index",
                MB_OK);
```

```
   MessageBox(hwnd,
              (LPSTR) TFieldNdx,
              (LPSTR) "Type field index",
              MB_OK);
   return 0;
}

/* SEND INDEX FIELD NUMBER */
i = atoi(fieldnum);
if (i >= 0 && i <= fieldamt)
{
   PFieldNdx = i-1;
   WFieldNdx = atoi(fieldwidth[i-1]);
   strcpy(TFieldNdx,fieldtype[i-1]);
   MessageBox(hwnd,
              (LPSTR) PFieldNdx,
              (LPSTR) "Position field index",
              MB_OK);
   MessageBox(hwnd,
              (LPSTR) WFieldNdx,
              (LPSTR) "Wide field index",
              MB_OK);
   MessageBox(hwnd,
              (LPSTR) TFieldNdx,
              (LPSTR) "Type field index",
              MB_OK);
   return 0;
}
```

```
  /* No have field name in file database */
  MessageBox(hwnd,
             (LPSTR) "Fieldname not correct",
             (LPSTR) "Error",
             MB_OK);
  return 1;
}

/*-------------------------------------------------------------------*/
struct tnode *create_tree()
{
  /* --------- seek to fieldindex ---------- */

  i = 0;
  lseek(hfd,(headlen+10),SEEK_SET);
  while (i != PosFieldNdx && i<fieldamt)
  {
        lseek(hfd,atoi(fieldwidth[i]),SEEK_CUR);
        i++;
  }
  for (j=0L;j<reccount;j++)
  {

  /* --------- check type index field ---------- */

      if (TypeFieldNdx == "N")/* type field index is numeric */
      {
         temp1 = get_tnode(temp1);
         temp1->ptr = j;
         strcpy(temp1->data,getdat());
         root = add_tree_inte(temp1,root);
```

```
char *getdat()
{
  char temp[100];

  /* -------- get fieldindex data to node -------- */
  read(hfd,temp,(WFieldNdx));
  temp[WFieldNdx] = '\0';
  MessageBox(hwnd,
             (LPSTR) temp,
             (LPSTR) "Getdat",
             MB_OK);

  lseek(hfd,(total+7-WFieldNdx),SEEK_CUR);
  return (temp);
}

/*-------------------------------------------------------------------*/

struct tnode *get_tnode(struct tnode *item)
{
  void *malloc(size_t);
  item=(struct tnode *) malloc(sizeof(struct tnode));
  if (item != NULL)
  {
     item->left=NULL;
     item->right=NULL;
     strcpy(item->data,null_str);
  }
```

```
  else
     MessageBox(hwnd,
                (LPSTR) "Nothing allocated",
                (LPSTR) "Get_tnode",
                MB_OK);
     return(item);
}

/* ------------------------------------------------------------------ */

struct tnode *add_tree_inte(struct tnode *new,struct tnode *root)
{
  if (root==Nextroot)
  {
     Tlevel = Tlevel +1;
     new->level=Tlevel;
     new->order=Torder;
     root = new;
     MAX_LEVEL=Tlevel;
     if (old_maxlevel>MAX_LEVEL)
        MAX_LEVEL=old_maxlevel;
     return(root);
  }
  else
  {
     Tlevel=root->level;
     datainv1 = atoi(new->data);
     datainv2 = atoi(root->data);
     result = datainv1 - datainv2;
  }
```

```
if (result != 0)
{
   if (result < 0)
   {
      Torder = root->order*2;
      if (root->left==NULL)
      {
         Nextroot->left=NULL;
         Nextroot->right=NULL;
         root->left=add_tree_inte(new,Nextroot);
      }
      else
         root->left = add_tree_inte(new,root->left);
   }
   else
   {
      Torder = (root->order*2) +1;
      if (root->right==NULL)
      {
         Nextroot->left=NULL;
         Nextroot->right=NULL;
         root->right=add_tree_inte(new,Nextroot);
      }
      else
         root->right=add_tree_inte(new,root->right);
   }
   return(root);
}
```

```
  else /* found field index in the same */
  {
     MessageBox(hwnd,
                (LPSTR) "Field index not unique.",
                (LPSTR) "Add_tree_inte",
                MB_OK);
     same=1;
     return(root);
  }
}
/* ------------------------------------------------------------ */

struct tnode *add_tree_char(struct tnode *new,struct tnode *root)
{
  if (root==Nextroot)
  {
     Tlevel = Tlevel +1;
     new->level=Tlevel;
     new->order=Torder;
     root = new;
     MAX_LEVEL=Tlevel;
     if (old_maxlevel>MAX_LEVEL)
        MAX_LEVEL=old_maxlevel;
     return(root);
  }
  else
  {
     Tlevel=root->level;
     result=strcmp(new->data,root->data);
  }
```

```
if (result != 0)
{
   if (result < 0)
   {
      Torder = root->order*2;
      if (root->left==NULL)
      {
         Nextroot->left=NULL;
         Nextroot->right=NULL;
         root->left=add_tree_char(new,Nextroot);
      }
      else
         root->left = add_tree_char(new,root->left);
   }
   else
   {
      Torder = (root->order*2) +1;
      if (root->right==NULL)
      {
         Nextroot->left=NULL;
         Nextroot->right=NULL;
         root->right=add_tree_char(new,Nextroot);
      }
      else
         root->right=add_tree_char(new,root->right);
   }
}
```

```
   else  /* Found field index in the same */
   {
      MessageBox(hwnd,
                 (LPSTR) "Field index not unique.",
                 (LPSTR) "Add_tree_inte",
                 MB_OK);
      same = 1;
      return(root);
   }
   return(root);
}

/* ------------------------------------------------------------------ */

void show_tree(struct tnode *tree)
{
   char buf6[7];
   char space[]= "      ";

   if (tree != NULL)
   {
      /* show left subtree first */
      if (tree->left != NULL)
         show_tree(tree->left);

      /* ----------- Write to disk ----------- */
      ltoa(tree->ptr,buf6,10);
      strcat(buf6,space);
      buf6[6]='\0';
      write(hfx,buf6,6);
      write(hfx,tree->data,WideFieldNdx);
```

```
      if (MAX_LEVEL<tree->level)
         MAX_LEVEL=tree->level;
      /* ------------------------------------ */

      /* show right subtree */
      if (tree->right != NULL)
         show_tree(tree->right);
   }
}

/* ------------------------------------------------------------------ */


void del_ndxfile(struct tnode *tree,long ptrdel)
{
   char buf6[7];
   char blanks[] = "      ";
   long memrec;

   if (tree != NULL)
   {
      /* show left subtree first */
      if (tree->left != NULL)
         del_ndxfile(tree->left,ptrdel);

      /* ----------- Modify index to disk ----------- */
      if (tree->ptr > ptrdel)
         tree->ptr = tree->ptr-1;
```

```
if (file_mem == 1 && del == 1)
{
   memrec = tree->ptr;
   lseek(hfd,headlen+3+(memrec*(total+7)),SEEK_SET);
   read(hfd,buf6,6);
   buf6[6] ='\0';
   if (strcmp(buf6,blanks) != 0)
   {
      memrec = atol(buf6);
      if (memrec > memdel)
      {
         memrec--;
         ltoa(memrec,buf6,10);
         strcat(buf6,blanks);
         buf6[6] = '\0';
         lseek(hfd,-6,SEEK_CUR);
         write(hfd,buf6,6);
      }
   }
}
ltoa(tree->ptr,buf6,10);
strcat(buf6,blanks);
buf6[6]='\0';
write(hfx,buf6,6);
write(hfx,tree->data,WideFieldNdx);
if (MAX_LEVEL<tree->level)
   MAX_LEVEL=tree->level;
/* ---------------------------------- */
```

```
      /* show right subtree */
      if (tree->right != NULL)
         del_ndxfile(tree->right,ptrdel);
   }
}

/* ------------------------------------------------------------------ */

void Wrbtree(struct tnode *tree)
{
   char buf6[7];
   char space[]="      ";

   lseek(hbt,0L,SEEK_SET);
   ltoa(MAX_LEVEL,buf6,10);
   strcat(buf6,space);
   buf6[6]='\0';
   write(hbt,buf6,6);
   order=pow(2,MAX_LEVEL)-1;
   for (rec=0L;rec<=order;rec++)
   Savebt(tree);
}

/* ------------------------------------------------------------------ */
```

```
void Savebt(struct tnode *btree)
{
   char buf6[7];
   char space[]="      ";
   if (btree != NULL)
   {
      if (btree->order == rec)
      {
         ltoa(btree->ptr,buf6,10);
         strcat(buf6,space);
         buf6[6]='\0';
         write(hbt,buf6,6);

         ltoa(btree->level,buf6,10);
         strcat(buf6,space);
         buf6[6]='\0';
         write(hbt,buf6,6);

         ltoa(btree->order,buf6,10);
         strcat(buf6,space);
         buf6[6]='\0';
         write(hbt,buf6,6);
         write(hbt,btree->data,WideFieldNdx);
      }
      /* search left subtree first */
      if (btree->left != NULL)
         Savebt(btree->left);
      /* search right subtree */
      if (btree->right != NULL)
         Savebt(btree->right);
   }
}
```

```
WideFieldSort=WFieldNdx;
PosFieldSort=PFieldNdx;
strcpy(TypeFieldSort,TFieldNdx);
if ((root_sort = sort_tree()) == NULL)/* Index field not unique */
   return 1;

/* ----------- open file index ----------- */
copy_struct(filesort);
sort_file(root_sort);
close(hft);
if (file_mem == 1)
{
   if ((hmemsort = OpenFile(file,&ofStruct,OF_WRITE)) == -1)
   {
      MessageBox(hwnd,
                 (LPSTR) "Can't open memsort file",
                 (LPSTR) "File Memo",
                 MB_OK);
      return 1;
   }
   lseek(hfm,0L,SEEK_SET);
   for (j=0L;j<memnum;j++)
   {
       if ((memr = malloc(widemem)) == NULL)
       {
          MessageBox(hwnd,
                     (LPSTR) "Not enough memory to memo sort",
                     (LPSTR) "Error",
                     MB_OK);
          close(hmemsort);
          return 1;
       }
```

```
            read(hfm,memr,widemem);
            memr[widemem] = '\0';
            write(hmemsort,memr,widemem);
            free(memr);
        }
        close(hmemsort);
    }
    if ((hft = OpenFile(filesort,&ofStruct,READ)) == -1)
    {
        MessageBox(hwnd,
                   (LPSTR) "Can't open sort file to set time",
                   (LPSTR) "File Sort",
                   MB_OK);
        return 1;
    }
    if ((hmemsort = OpenFile(file,&ofStruct,READ)) == -1)
    {
        MessageBox(hwnd,
                   (LPSTR) "Can't open memsort file to set time",
                   (LPSTR) "File Memo",
                   MB_OK);
        close(hft);
        return 1;
    }
    getftime(hft,&f);
    setftime(hft,&f);
    setftime(hmemsort,&f);
    close(hft);
    close(hmemsort);
    return 0;
}
/*-------------------------------------------------------------------------*/
```

```
struct tnode *sort_tree()
{
  /* --------- seek to fieldindex ----------- */
  i = 0;
  lseek(hfd,(headlen+10),SEEK_SET);
  while (i != PosFieldSort && i<fieldamt)
  {
        lseek(hfd,atoi(fieldwidth[i]),SEEK_CUR);
        i++;
  }
  for (j=0L;j<reccount;j++)
  {
     /* --------- check type index field ---------- */
     if (TypeFieldSort == "N")/* type field index is numeric */
     {
        temp1 = get_tnode(temp1);
        strcpy(temp1->data,getdat());
        temp1->ptr = j;
        root_sort = sort_tree_inte(temp1,root_sort);
     }
     else      /* type field index is character or date */
     {
        temp1 = get_tnode(temp1);
        strcpy(temp1->data,getdat());
        temp1->ptr = j;
        root_sort = sort_tree_char(temp1,root_sort);
     }
  }
  return (root_sort);
}
/*-------------------------------------------------------------------*/
```

```
struct tnode *sort_tree_inte(struct tnode *new,struct tnode *root_sort)
{
  if (root_sort==Nextroot)
  {
     Tlevel = Tlevel +1;
     new->level=Tlevel;
     new->order=Torder;
     root_sort = new;
     MAX_LEVEL=Tlevel;
     if (old_maxlevel > MAX_LEVEL)
        MAX_LEVEL=old_maxlevel;
     return(root_sort);
  }
  else
  {
     Tlevel=root_sort->level;
     datainv1 = atoi(new->data);
     datainv2 = atoi(root_sort->data);
     result = datainv1 - datainv2;
  }
  if (result <= 0)
  {
     Torder = root_sort->order*2;
     if (root_sort->left==NULL)
     {
        Nextroot->left=NULL;
        Nextroot->right=NULL;
        root_sort->left=sort_tree_inte(new,Nextroot);
     }
```

```
    else
        root_sort->left = sort_tree_inte(new,root_sort->left);
  }
  else
  {
     Torder = (root_sort->order*2) +1;
     if (root_sort->right==NULL)
     {
        Nextroot->left=NULL;
        Nextroot->right=NULL;
        root_sort->right=sort_tree_inte(new,Nextroot);
     }
     else
        root_sort->right=sort_tree_inte(new,root_sort->right);
  }
  return(root_sort);
}
/* ------------------------------------------------------------------ */

struct tnode *sort_tree_char(struct tnode *new,struct tnode *root_sort)
{
  if (root_sort==Nextroot)
  {
     Tlevel = Tlevel +1;
     new->level=Tlevel;
     new->order=Torder;
     root_sort = new;
     MAX_LEVEL=Tlevel;
     if (old_maxlevel>MAX_LEVEL)
        MAX_LEVEL=old_maxlevel;
     return(root_sort);
  }
```

```
    else
    {
       Tlevel=root_sort->level;
       result=strcmp(new->data,root_sort->data);
    }
    if (result <= 0)
    {
       Torder = root_sort->order*2;
       if (root_sort->left==NULL)
       {
          Nextroot->left=NULL;
          Nextroot->right=NULL;
          root_sort->left=sort_tree_char(new,Nextroot);
       }
       else
          root_sort->left = sort_tree_char(new,root_sort->left);
    }
    else
    {
       Torder = (root_sort->order*2) +1;
       if (root_sort->right==NULL)
       {
          Nextroot->left=NULL;
          Nextroot->right=NULL;
          root_sort->right=sort_tree_char(new,Nextroot);
       }
       else
          root_sort->right=sort_tree_char(new,root_sort->right);
    }
    return(root_sort);
}
/* ---------- */
```

```
int sort_file(struct tnode *tree)
{
  char *rectemp;

  if (tree != NULL)
  {
     /* show left subtree first */
     if (tree->left != NULL)
        sort_file(tree->left);
     /* ----------- Write to disk ----------- */
     lseek(hfd,headlen+3+((tree->ptr)*(total+7)),SEEK_SET);
     if ((rectemp = malloc(total+7)) == NULL)
     {
        MessageBox(hwnd,
                   (LPSTR) "Not enough memory",
                   (LPSTR) "Sort file",
                   MB_OK);
        return 1;
     }
     read(hfd,rectemp,total+7);
     rectemp[total+7]='\0';
     lseek(hft,0L,SEEK_END);
     write(hft,rectemp,total+7);
     free(rectemp);
     /* ------------------------------------ */
     /* show right subtree */
     if (tree->right != NULL)
        sort_file(tree->right);
  }
  return 0;
}
/* ------------------------------------------------------------------ */
```

```
Replace(struct rep r)
{
  int length;

  if (checkfieldindex(r.f_repname) == 1)
     return 1;
  WideFieldSort=WFieldNdx;
  PosFieldSort=PFieldNdx;
  strcpy(TypeFieldSort,TFieldNdx);

  if (strcmp(r.oldval,r.newval)==0)
     return 0;
  if (PosFieldSort == PosFieldNdx)
  {
     temp=find_tnode(root,&temp2,r.oldval);
     if (temp != NULL)
     {
        replaceptr=temp->ptr;
        old_ptr=1;
        if (insert(r.newval)==1)
        {
           old_ptr=0;
           return 1;
        }
        old_ptr=0;
        if (strcmp(TypeFieldNdx,"N")==0)
           root=del_tnode_inte(r.oldval,root);
        else
           root=del_tnode_char(r.oldval,root);
```

```
        MAX_LEVEL=0;
        lseek(hfx,4L,SEEK_SET);
        show_tree(root);
        Wrbtree(root);

        length=(headlen+10)+((total+7)*temp->ptr);
        for (i=0;i<PosFieldNdx;i++)
            length=length+atoi(fieldwidth[i]);
        lseek(hfd,length,SEEK_SET);
        write(hfd,r.newval,WideFieldNdx);
    }
    else /* Field not found */
        return 1;
}
else /* field replace isn't fieldindex */
{
  if ((root_sort=sort_tree())==NULL)
     return 1;
  for (j=0L;j<reccount;j++)
  {
        temp=find_tnode(root_sort,&temp2,r.oldval);
        if (temp != NULL)
        {
           length=(headlen+10)+((total+7)*temp->ptr);
           for (i=0;i<PosFieldSort;i++)
               length=length+atoi(fieldwidth[i]);
           lseek(hfd,length,SEEK_SET);
           write(hfd,r.newval,WideFieldSort);
           if (temp->left != NULL)
              root_sort=temp->left;
```

```
            else
               j=reccount;
         }
      }  /* end of for loop */
   }
   if (temp==NULL)
   {
      MessageBox(hwnd,
                 (LPSTR) "Record not found",
                 (LPSTR) "Replace",
                 MB_OK);
      return 1;
   }
   return 0;
}

/* ---------------------- find node ------------------- */

struct tnode *find_tnode(struct tnode *tree,struct tnode **prev,char *val)
{
   if (tree != NULL)
   {
      /* Until found val or end of tree */
      while ((tree != NULL) && (strcmp(tree->data,val)!=0))
      {
            *prev=tree;
            /* left or right */
            if (strcmp(tree->data,val)<0)
               tree=tree->right;
```

```
            else
                tree=tree->left;
        }
    }
    return(tree);
}

/* ------------------------------------------------------------------- */
```

```
                    ************************
                    *     PROGRAM MEMO.C     *
                    ************************

#define blanks  "       "

#include <Windows.h>
#include <sys\stat.h>
#include <errno.h>
#include <io.h>
#include <alloc.h>
#include "uhead.h"

extern HWND hwnd;

/* -------------- MEMO FILE ---------------- */

memo_record(char mrecord[])
{
  long ptr,recno;
  char *buf,buf6[7];
  OFSTRUCT ofstruct;

/* Check file memo exist if not exist,create it,if not append to it */

  if (memicmp(mrecord," ",1) == 0)/* Have not add memo record */
     return 0;
```

```
if (file_mem != 1 && memnum == 0L)/* Memo file not exist */
{
   /* Create memo file */
   if ((hfm = OpenFile(filem,&ofstruct,OF_CREATE)) == -1)
   {
      MessageBox(hwnd,
                 (LPSTR) "Can't create memo file.",
                 (LPSTR) "File",
                 MB_OK);
      return 1;
   }
   close(hfm);
   if ((hfm = OpenFile(filem,&ofstruct,OF_READWRITE)) == -1)
   {
      MessageBox(hwnd,
                 (LPSTR) "Can't open memo file.",
                 (LPSTR) "File",
                 MB_OK);
      return 1;
   }
   file_mem = 1;/* Open memo file mark */
}
if (file_mem != 1)
{
   MessageBox(hwnd,
              (LPSTR) "Memo file not open.",
              (LPSTR) "File",
              MB_OK);
   return 1;
}
```

```
    ptr = check_pointer(recnum);/* Find real pointer */
    recno = ptr;
    lseek(hfd,headlen+3+((recno-1)*(total+7)),SEEK_SET);
    read(hfd,buf6,6);
    buf6[6] ='\0';
    if (strcmp(buf6,blanks) == 0)/* Record has not memo record before */
    {
       ltoa(memnum,buf6,10);
       strcat(buf6,blanks);
       buf[6] = '\0';
       recno = ptr;
       lseek(hfd,headlen+3+((total+7)*(recno-1)),SEEK_SET);
       write(hfd,buf6,6);
       memnum++;
       lseek(hfm,0L,SEEK_END);
    }
    else/* Has memo record before add memo record */
    {
       recno = atol(buf6);
       lseek(hfm,(recno*widemem),SEEK_SET);
    }
    write(hfm,mrecord,widemem);/* Write memo record to memo file */
    write_header_file(hfd,total,fieldamt);/* Update header database file */
    return 0;
}
```

```
/* ---------------- MEMO DISPLAY ----------------- */

char *mem_display()
{
  long ptr,memrec;
  char *mem_record;
  char buf6[7];

  if (memnum == 0L)/* Memo file not exist */
     return (NULL);
  if (file_mem == 0)/* Memo file not open */
  {
     MessageBox(hwnd,
              (LPSTR) "Memo file not open.",
              (LPSTR) "File",
              MB_OK);
     return (NULL);
  }
  ptr = check_pointer(recnum);/* Find real pointer */
  lseek(hfd,headlen+3+((ptr-1)*(total+7)),SEEK_SET);
  read(hfd,buf6,6);
  buf6[6] ='\0';
  if (strcmp(buf6,blanks) == 0)/* Has not memo record */
     return (NULL);
  memrec = atol(buf6);
  lseek(hfm,(memrec*widemem),SEEK_SET);
```

```
    if ((mem_record= malloc(widemem)) == NULL)
    {
       MessageBox(hwnd,
                  (LPSTR) "Not enough memory.",
                  (LPSTR) "File",
                  MB_OK);
       return NULL;
    }
    read(hfm,mem_record,widemem);/* Read memo record */
    mem_record[widemem] = '\0';
    free(mem_record);
    return (mem_record);
}

/* ---------------- PACK MEMO RECORD ---------------- */

mem_pack(long memrec)
{
    long loop;
    char *source;
    struct stat statbuf;

    loop = memnum-memrec-1L;

    /* write data record */
    for (j=0L;j<loop;j++)
    {
        lseek(hfm,(memrec+j+1)*(widemem),SEEK_SET);
```

```
        if ((source = malloc(widemem)) == NULL)/* ALLOCATE MEMORY FOR DATA */
        {
           MessageBox(hwnd,
                      (LPSTR) "Not enough memory.",
                      (LPSTR) "File",
                      MB_OK);
           return 1;
        }
        read(hfm,source,(widemem));/* READ DATA TO MEMORY */
        source[widemem] = '\0';
        lseek(hfm,(memrec+j)*(widemem),SEEK_SET);
        write(hfm,source,widemem);/* WRITE DATA TO TEMP FILE */
        free(source);
     }
     fstat(hfm,&statbuf);
     chsize(hfm,statbuf.st_size-widemem);
     memnum--;/* DECREMENT NUMBER OF MEMO RECORD */
     return 0;
}
```

```
                    ****************************
                    *    PROGRAM MODI_CH.C     *
                    ****************************

#include <Windows.h>
#include <uhead.h>
#include <uindex.h>
#include <sys\stat.h>

extern HWND hwnd;

insert(char *insnode)
{
  char buf1[2],buf2[3];

  if (root == NULL)
  {
     MessageBox(hwnd,
              (LPSTR) "Must open index file before",
              (LPSTR) "Insert",
              MB_OK);
     return 1;
  }
  temp = get_tnode(temp);
  if (temp != NULL)
  {
     if (old_ptr == 1)
        temp->ptr = replaceptr;
     else
        temp->ptr=reccount;
```

```
        strcpy(temp->data,insnode);
        temp->data[WideFieldNdx] = '\0';
    }
    else
    {
        MessageBox(hwnd,
                   (LPSTR) "Not enough memory.",
                   (LPSTR) "Alloc",
                   MB_OK);
        return 1;
    }
    if (strcmp(TypeFieldNdx,"N") == 0)
        root = add_tree_inte(temp,root);
    else
        root = add_tree_char(temp,root);
    if (same == 1)
    {
        same = 0;
        return 1;
    }
    if (old_ptr==0)
    {
        lseek(hfx,4L,SEEK_SET);
        show_tree(root);
        Wrbtree(root);
    }
    return 0;
}
```

```
/* --------------operation delete record for indexed file ---------- */

pack_ndx(long ptrnode)
{
  char *delvalue;
  struct stat statbuf;
  char buf1[2],buf2[3],buf6[7];

  if (root == NULL)
  {
     MessageBox(hwnd,
              (LPSTR) "Must open index file before",
              (LPSTR) "Pack_ndx",
              MB_OK);
     return 1;
  }
  lseek(hfx,(((WideFieldNdx+6)*ptrnode)+10),SEEK_SET);
  if ((delvalue = malloc(WideFieldNdx)) == NULL)
  {
     printf("Can't alloc delvalue\n");
     return 1;
  }
  read(hfx,delvalue,WideFieldNdx);
  delvalue[WideFieldNdx] = '\0';
  if (strcmp(TypeFieldNdx,"N")==0)
  {
     root = del_tnode_inte(delvalue,root);
     free(delvalue);
  }
```

```
  else
  {
     root = del_tnode_char(delvalue,root);
     free(delvalue);
  }

  MAX_LEVEL=0L;
  lseek(hfx,(WideFieldNdx*ptrnode)+4L,SEEK_SET);
  read(hfx,buf6,6);
  buf6[6]='\0';
  ptrnode=atol(buf6);
  lseek(hfx,4L,SEEK_SET);
  del_ndxfile(root,ptrnode);
  fstat(hfx,&statbuf);
  chsize(hfx,statbuf.st_size-(6+WideFieldNdx));
  Wrbtree(root);
  fstat(hbt,&statbuf);
  chsize(hbt,statbuf.st_size-(18+WideFieldNdx));
  return 0;
}

/* ------------------------------------------------------ */

struct tnode *l_rotate(struct tnode *temprt)
{
  struct tnode *temp =temprt;

  if (temprt != NULL)
  {
     temprt = temprt->right;
     temprt->level=temp->level;
```

```
        temprt->order=temp->order;
        *prev=temprt;

        while (temprt->right != NULL)
              temprt=rotate_tright(temprt);
        order=temprt->order;
        temprt=*prev;
        temp->right=temprt->left;
        temp->level=temp->level+1;
        temp->order=(temp->order*2);
        if (temp->left != NULL)
        {
            lftemp=temp->left;
            lftemp->level=lftemp->level+1;
            lftemp->order=(lftemp->order*2);
        }
        temprt->left  = temp;
    }
    return (temprt)
}

/* ---------------------------------------------- */

struct tnode *rotate_tright(struct tnode *temprt)
{
    rgtemp=temprt->right;
    rgtemp->level=temprt->level+1;
    rgtemp->order=(temprt->order*2)+1;
```

```
  if (temprt->left != NULL)
  {
     lftemp=temprt->left;
     lftemp->level=rgtemp->level;
     lftemp->order=rgtemp->order-1;
  }
  temprt=temprt->right;
  return(temprt);
}

/* ------------------------------------------------ */

struct tnode *r_rotate(struct tnode *rtree)
{

 struct tnode *temp = rtree;
 if (rtree != NULL)
 {
    rtree=rtree->left;
    rtree->level=temp->level;
    rtree->order=temp->order;
    *prev=rtree;
    while (rtree->left != NULL)
          rtree=rotate_tleft(rtree);
    rtree=*prev;
    temp->left=rtree->right;
    temp->level=temp->level+1;
    temp->order=(temp->order*2)+1;
```

```
int tree_leaf(struct tnode *item)
{
 if ((item->left == NULL) && (item->right == NULL))
    return(TRUE);
 else
    return(FALSE);

}

/* ------------------------------------------------------ */

struct tnode *del_tnode_inte(char *str,struct tnode *tree)
{
  result=atoi(str)-atoi(tree->data);

  if (tree== NULL)
  {
     MessageBox(hwnd,
                (LPSTR) "Value not found,return NULL",
                (LPSTR) "del_tnode_inte",
                MB_OK);
     return(NULL);
  }
  if (result == 0)
  {
     if (tree_leaf(tree))
     {
        free(tree);
        return(NULL);
     }
```

```
        else
        {
            if (tree->left !=NULL)
            {
                tree=r_rotate(tree);
                tree=del_tnode_inte(str,tree);
            }
            else
            {
                tree = l_rotate(tree);
                tree=del_tnode_inte(str,tree);
            }
        }
    }
    else
    {
        if (result < 0)
            tree->left=del_tnode_inte(str,tree->left);
        else
            tree->right=del_tnode_inte(str,tree->right);
    }
    return(tree);
}

/* ------------------------------------------------------------ */
```

```
struct tnode *del_tnode_char(char *str,struct tnode *tree)
{
  if (tree== NULL)
  {
     MessageBox(hwnd,
                (LPSTR) "Value not found,return NULL",
                (LPSTR) "del_tnode_char",
                MB_OK);
     return(NULL);
  }
  if (strcmp(str,tree->data) == 0)
  {
     if (tree_leaf(tree))
     {
        free(tree);
        return(NULL);
     }
     else
     {
        if (tree->left !=NULL)
        {
           tree=r_rotate(tree);
           tree=del_tnode_char(str,tree);
        }
        else
        {
           tree = l_rotate(tree);
           tree=del_tnode_char(str,tree);
        }
     }
  }
```

```
    else
    {
        if (strcmp(str,tree->data) < 0)
            tree->left=del_tnode_char(str,tree->left);
        else
            tree->right=del_tnode_char(str,tree->right);
    }
    return(tree);
}
```

```
                    ****************************
                    *     PROGRAM POINT_RE.C     *
                    ****************************

/* ---------------- PROGRAM FOR MENU POSITION ---------------- */

#include <Windows.h>
#include "uhead.h"

/* ------------- point to record number -------------- */

point_record(long num)
{
      recnum = num;
      return 0;
}
/* ------------- point to next record -------------- */

next_record()
{
      recnum++;
      return 0;
}
/* --------------- point to previous record ---------------- */

pre_record()
{
      recnum--;
      return 0;
}
```

```
/* ---------------- skip record ------------------ */

skip_record(long skip)
{
      recnum = recnum+skip;
      return 0;
}

/* ---------------- point to first record ---------------- */

first_record()
{
      recnum = 1L;
      return 0;
}

/* ---------------- point to last record ---------------- */

last_record()
{
      recnum = reccount;
      return 0;
}
```

```
                    **********************
                    *     PROGRAM QRY.C    *
                    **********************

#include<Windows.h>
#include "uhead.h"
#include "uindex.h"

int  opt,cond_pt1,cond_pt2,k=0;
long count=0L,slen;
long *Fptr_qry,*Sptr_qry,*tempqry;
char No[2];
char qfield[30];

extern HWND hwnd;

long *Qry(countqry)
{
   if (countqry == 0)
   {
      MessageBox(hwnd,
                 (LPSTR) "No have query for search",
                 (LPSTR) "Query",
                 MB_OK);
      return NULL;
   }
```

```
/* ---------------- Search for one query ----------- */

if (countqry==1)
{
   if ((q[k].cond2==2) || (q[k].cond2==3))
   {
      MessageBox(hwnd,
                 (LPSTR) "Syntax query error",
                 (LPSTR) "Query",
                 MB_OK);
      return NULL;
   }
   if (q[k].cond1==1)
      opt=NOT();          /* find opposite operate */
   else
      opt=q[k].operate;
   itoa(q[k].FN,No,10);
   checkfieldindex(No);
   if ((ptree=malloc((reccount*4)+1))==NULL)
   {
      MessageBox(hwnd,
                 (LPSTR) "Can't alloc memory",
                 (LPSTR) "Query",
                 MB_OK);
      return NULL;
   }
   Fptr_qry=find_ptr();
   if (count==0L)
      return NULL;
   count1=count;
```

```
    if ((ptree1=malloc((reccount*4)+1))==NULL)
    {
       MessageBox(hwnd,
                  (LPSTR) "Can't alloc memory",
                  (LPSTR) "Query",
                  MB_OK);
       return NULL;
    }
    for (j=0L;j<count1;j++)
        *(ptree1+j)=*(Fptr_qry+j);

    free(ptree);
    return(ptree1);
}

/* --------- Search for greater than one query ---------*/

if (q[k].cond2==0)      /* no have condition */
{
   MessageBox(hwnd,
              (LPSTR) "Syntax query error",
              (LPSTR) "Query",
              MB_OK);
   return NULL;
}
if (countqry != 0)
{
   if (q[k].cond1==1)
      opt=NOT();           /*find opposite operate */
   else
      opt=q[k].operate;
```

```
itoa(q[k].FN,No,10);
checkfieldindex(No);
if ((ptree=malloc((reccount*4)+1))==NULL)
{
   MessageBox(hwnd,
              (LPSTR) "Can't alloc memory",
              (LPSTR) "Query",
              MB_OK);
   return NULL;
}
Fptr_qry=find_ptr();        /* ptr of query# 1 */
if (Fptr_qry==NULL)
   *Fptr_qry = -1;
count1=count;
if ((ptree1=malloc((reccount*4)+1))==NULL)
{
   MessageBox(hwnd,
              (LPSTR) "Can't alloc memory",
              (LPSTR) "Query",
              MB_OK);
   return NULL;
}
if (count1 == 0L)
   *ptree1 = -1;
for (j=0L;j<count1;j++)
    *(ptree1+j)=*(Fptr_qry+j);
free(ptree);
cond_pt1=q[k].cond2;    /* consider AND,OR condition of query# 1 */
countqry--;
k++;
```

```
while (countqry != 0)
{
      if (cond_pt1==0 || cond_pt1==1)
      {
         MessageBox(hwnd,
                    (LPSTR) "Syntax query error",
                    (LPSTR) "Query",
                    MB_OK);
         return NULL;
      }
      if (q[k].cond1==1)
         opt=NOT();              /*find opposite operate */
      else
         opt=q[k].operate;
      itoa(q[k].FN,No,10);
      checkfieldindex(No);
      if ((ptree=malloc((reccount*4)+1))==NULL)
      {
         MessageBox(hwnd,
                    (LPSTR) "Can't alloc memory",
                    (LPSTR) "Query",
                    MB_OK);
         return NULL;
      }
      Sptr_qry=find_ptr();
      if (Sptr_qry==NULL)
         *Sptr_qry = -1;
      count2=count;
```

```
if ((ptree2=malloc((reccount*4)+1))==NULL)
{
   MessageBox(hwnd,
              (LPSTR) "Can't alloc memory",
              (LPSTR) "Query",
              MB_OK);
   return NULL;
}
for (j=0L;j<count2;j++)
    *(ptree2+j)= *(Sptr_qry+j);
free(ptree);

/* consider AND,OR condition of query# 2 */
cond_pt2=q[k].cond2;
if (cond_pt1==2)
{
   ptree1=Operate_AND(ptree1,ptree2);
   free(tempqry);
   count1=len_and;
   if (ptree1==NULL)
      *ptree1 = -1;
   if ((countqry-1)==0)
   {
      if (cond_pt2 != 0)
      {
         MessageBox(hwnd,
                    (LPSTR) "Syntax query error",
                    (LPSTR) "Query",
                    MB_OK);
         return NULL;
      }
```

```
        }
        cond_pt1=cond_pt2;
        countqry--;
        k++;
        if (countqry==0)
        {
           if (cond_pt1 != 0)
           {
              MessageBox(hwnd,
                         (LPSTR) "Syntax query error",
                         (LPSTR) "Query",
                         MB_OK);
              return NULL;
           }
        }
     }
  else
  {
     ptree1=Operate_OR(ptree1,ptree2);
     free(tempqry);
     count1=len_or;
     if (ptree1==NULL)
        *ptree1 = -1;
     if ((countqry-1)==0)
     {
        if (cond_pt2 != 0)
        {
           MessageBox(hwnd,
                      (LPSTR) "Syntax query error",
                      (LPSTR) "Query",
                      MB_OK);
```

```
                    return NULL;
                }
            }
            cond_pt1=cond_pt2;
            countqry--;
            k++;
            if (countqry==0)
            {
                if (cond_pt1 != 0 )
                {
                    MessageBox(hwnd,
                               (LPSTR) "Syntax query error",
                               (LPSTR) "Query",
                               MB_OK);
                    return NULL;
                }
            }
        }
    } /* end of while loop */
}
if (count1==0L)
    return NULL;

return(ptree1);
}
/* ------------------------------------------------------ */
```

```
long *find_ptr()
{
   switch(opt)
   {
         case 1: /* Equal */
                 pr=0L;
                 count=0L;
                 ptree=Equal();
                 break;

        case 2: /* Not Equal */
                 pr=0L;
                 count=0L;
                 ptree=Not_Eq();
                 break;

       case 3: /* Greater */
                 pr=0L;
                 count=0L;
                 ptree=Greater();
                 break;

        case 4: /* Less */
                 pr=0L;
                 count=0L;
                 ptree=Less();
                 break;
```

```
            case 5: /* Greater or Equal */
                    pr=0L;
                    count=0L;
                    ptree=Greater_Eq();
                    break;

            case 6: /* Less or Equal */
                    pr=0L;
                    count=0L;
                    ptree=Less_Eq();
                    break;
    } /* end of switch case */

    ptree[count]='\0';
    return(ptree);
}
/* ------------------------------------------------------------ */

NOT()
{
  if (q[k].operate==1)  /* not = is != */
  {
     q[k].operate=2;
     return(q[k].operate);
  }
  if (q[k].operate==2)  /* not != is = */
  {
     q[k].operate=1;
     return(q[k].operate);
  }
```

```
  if (q[k].operate==3)  /* not > is <= */
  {
     q[k].operate=6;
     return(q[k].operate);
  }
  if (q[k].operate==4)  /* not < is >= */
  {
     q[k].operate=5;
     return(q[k].operate);
  }
  if (q[k].operate==5)  /* not >= is < */
  {
     q[k].operate=4;
     return(q[k].operate);
  }
  if (q[k].operate==6)  /* not <= is > */
  {
     q[k].operate=3;
     return(q[k].operate);
  }
  return 0;
}
/* ------------------------------------------------------ */
```

```
long *Operate_AND(long *setqry1,long *setqry2)
{
 long q1,q2,x=0L,lqry1,lqry2;
 long *And_qry;

 len_and=0L;
 if (count1 != 0L && count2 != 0L)
 {
    if (count1 <= count2)
    {
       lqry1 = count1;
       lqry2 = count2;
    }
    else
    {
       if ((tempqry=malloc((count1*4)+1))==NULL)
       {
          MessageBox(hwnd,
                     (LPSTR) "Can't alloc memory",
                     (LPSTR) "AND",
                     MB_OK);
          return NULL;
       }
       for (j=0L;j<count1;j++)
           *(tempqry+j)=*(setqry1+j);
       tempqry[count1]='\0';
       for (j=0L;j<count2;j++)
           *(setqry1+j)=*(setqry2+j);
       setqry1[count2]='\0';
       for (j=0L;j<count1;j++)
           *(setqry2+j)=*(tempqry+j);
```

```
    setqry2[count1]='\0';
    lqry1 = count2;
    lqry2 = count1;
}

if ((And_qry=malloc((reccount*4)+1))==NULL)
{
    MessageBox(hwnd,
               (LPSTR) "Can't alloc memory",
               (LPSTR) "AND",
               MB_OK);
    return NULL;
}
for (q1=0L;q1<lqry1;q1++)
{
    for (q2=0L;q2<lqry2;q2++)
    {
        if (*(setqry1+q1) == *(setqry2+q2))
        {
            len_and++;
            *(And_qry+x) = *(setqry2+q2);
            for (j=0L;j<(lqry2-(q2+1));j++)
            *(setqry2+(q2+j))= *(setqry2+(q2+j+1));
            x++;
            lqry2--;
            setqry2[lqry2]='\0';
            q2=lqry2;
        }
    }  /* end of for loop 2*/
} /* end of for loop 1*/
}
```

```
 else
     if (countqry==0)
        return NULL;
     else
     {
        *(And_qry)= -1;
        return(And_qry);
     }
     And_qry[len_and]='\0';
     return (And_qry);
}

/* ------------------------------------------------------------ */

long *Operate_OR(long *setqry1,long *setqry2)
{
 long q1,q2;
 long *Or_qry;
 long lqry1,lqry2;

 len_or=0L;
 if (count1 != 0L && count2 != 0L)
 {
    if (count1 <= count2)
    {
       lqry1 = count1;
       lqry2 = count2;
    }
```

```
else
{
   if ((tempqry=malloc((count1*4)+1))==NULL)
   {
      MessageBox(hwnd,
                 (LPSTR) "Can't alloc memory",
                 (LPSTR) "OR",
                 MB_OK);
     return NULL;
   }

   for (j=0L;j<count1;j++)
       *(tempqry+j)=*(setqry1+j);
   tempqry[count1]='\0';
   for (j=0L;j<count2;j++)
       *(setqry1+j)=*(setqry2+j);
   setqry1[count2]='\0';
   for (j=0L;j<count1;j++)
       *(setqry2+j)=*(tempqry+j);
   setqry2[count1]='\0';

   lqry1 = count2;
   lqry2 = count1;
}
```

```
    for (q1=0L;q1<lqry1;q1++)
    {
       for (q2=0L;q2<lqry2;q2++)
       {
           if (*(setqry2+q2) == *(setqry1+q1))
           {
              for (j=0L;j<(lqry2-(q2+1));j++)
                  *(setqry2+(q2+j))= *(setqry2+(q2+j+1));

              lqry2--;
              setqry2[lqry2]='\0';
              q2=lqry2;
           }
       } /* end of for loop 1 */
    } /* end of for loop 2*/
    for (j=0L;j<lqry2;j++)
        *(setqry1+(lqry1+j))= *(setqry2+j);
}
else  /* count1 or count2 = 0 */
{
   if ((count1==0L) && (count2==0L))
   {
      if (countqry==0)
         return NULL;
      else
      {
         *(setqry1)= -1;
         return(setqry1);
      }
   }
```

```
    if (count1 == 0L)
    {
       for (j=0L;j<count2;j++)
           setqry1= setqry2;
       len_or=count1+count2;
       setqry1[len_or]='\0';
       return(setqry1);
    }
    if (count2 == 0L)
    {
       len_or=count1+count2;
       setqry1[len_or]='\0';
       return(setqry1);
    }
 }
 len_or=lqry1+lqry2;
 setqry1[len_or]='\0';
 return(setqry1);
}

/* ------------------------------------------------------------ */

long *Equal()
{
  i=0;
  lseek(hfd,(headlen+10),SEEK_SET);
  while (i != PFieldNdx && i< fieldamt)
  {
     lseek(hfd,atoi(fieldwidth[i]),SEEK_CUR);
     i++;
  }
```

```
    for (j=0L;j<reccount;j++)
    {
        read(hfd,qfield,WFieldNdx);
        qfield[WFieldNdx] = '\0';
        if (strcmp(qfield,q[k].FV)==0)
        {
           count++;
           *(ptree+pr)=j;
           pr++;
        }
        lseek(hfd,(total+7-WFieldNdx),SEEK_CUR);
    }
    if (count==0L)
       return NULL;

    return(ptree);
}

/* ---------------------------------------------------------- */

long *Not_Eq()
{
    i=0;
    lseek(hfd,(headlen+10),SEEK_SET);
    while (i != PFieldNdx && i< fieldamt)
    {
          lseek(hfd,atoi(fieldwidth[i]),SEEK_CUR);
          i++;
    }
```

```
   for (j=0L;j<reccount;j++)
   {
        read(hfd,qfield,WFieldNdx);
        qfield[WFieldNdx] = '\0';
        if (strcmp(qfield,q[k].FV) != 0)
        {
           count++;
           *(ptree+pr)=j;
           pr++;
        }
        lseek(hfd,(total+7-WFieldNdx),SEEK_CUR);
   }
   if (count==0L)
      return NULL;

   return(ptree);
}

/* ------------------------------------------------------------ */

long *Greater()
{
   i=0;
   lseek(hfd,(headlen+10),SEEK_SET);
   while (i != PFieldNdx && i< fieldamt)
   {
         lseek(hfd,atoi(fieldwidth[i]),SEEK_CUR);
         i++;
   }
```

```
    for (j=0L;j<reccount;j++)
    {
        read(hfd,qfield,WFieldNdx);
        qfield[WFieldNdx] = '\0';
        if (strcmp(qfield,q[k].FV) > 0)
        {
           count++;
           *(ptree+pr)=j;
           pr++;
        }
        lseek(hfd,(total+7-WFieldNdx),SEEK_CUR);
    }
    if (count==0L)
       return NULL;

    return(ptree);
}

/* ---------------------------------------------------------- */

long *Less()
{
  i=0;
  lseek(hfd,(headlen+10),SEEK_SET);
  while (i != PFieldNdx && i< fieldamt)
  {
        lseek(hfd,atoi(fieldwidth[i]),SEEK_CUR);
        i++;
  }
```

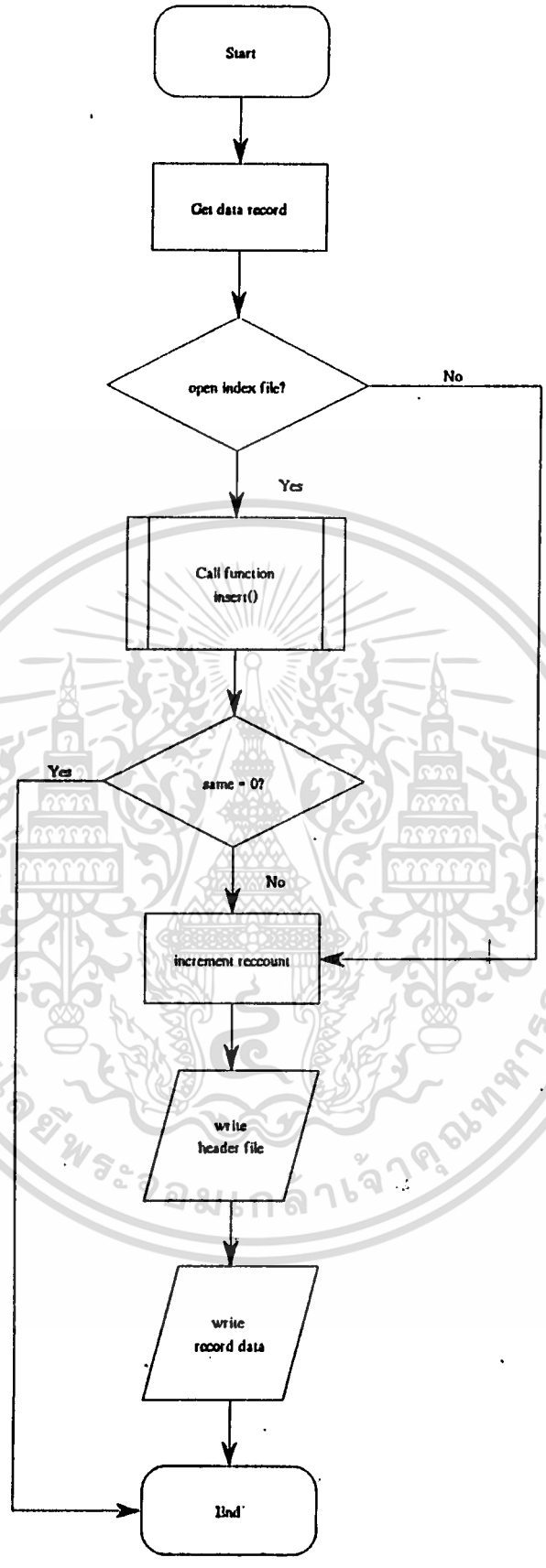
```
    for (j=0L;j<reccount;j++)
    {
        read(hfd,qfield,WFieldNdx);
        qfield[WFieldNdx] = '\0';
        if (strcmp(qfield,q[k].FV) < 0)
        {
           count++;
           *(ptree+pr)=j;
           pr++;
        }
        lseek(hfd,(total+7-WFieldNdx),SEEK_CUR);
    }
    if (count==0L)
       return NULL;

    return(ptree);
}

/* ---------------------------------------------------------- */

long *Greater_Eq()
{
    i=0;
    lseek(hfd,(headlen+10),SEEK_SET);
    while (i != PFieldNdx && i< fieldamt)
    {
          lseek(hfd,atoi(fieldwidth[i]),SEEK_CUR);
          i++;
    }
```

```
  for (j=0L;j<reccount;j++)
  {
      read(hfd,qfield,WFieldNdx);
      qfield[WFieldNdx] = '\0';
      if ((strcmp(qfield,q[k].FV)==0) || (strcmp(qfield,q[k].FV) > 0))
      {
         count++;
         *(ptree+pr)=j;
         pr++;
      }
      lseek(hfd,(total+7-WFieldNdx),SEEK_CUR);
  }
  if (count==0L)
     return NULL;

  return(ptree);
}

/* ------------------------------------------------------------ */

long *Less_Eq()
{
  i=0;
  lseek(hfd,(headlen+10),SEEK_SET);
  while (i != PFieldNdx && i< fieldamt)
  {
        lseek(hfd,atoi(fieldwidth[i]),SEEK_CUR);
        i++;
  }
```

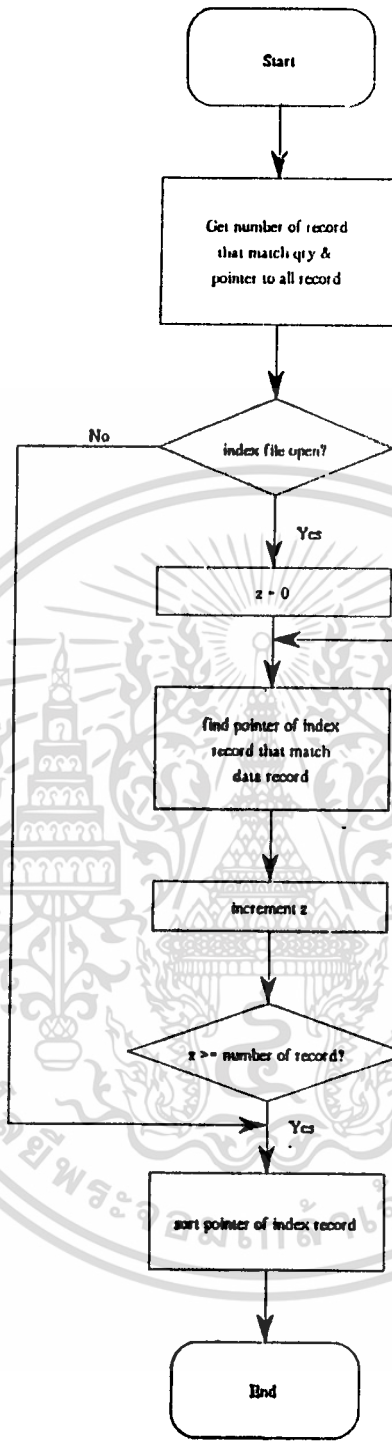
```
  for (j=0L;j<reccount;j++)
  {
      read(hfd,qfield,WFieldNdx);
      qfield[WFieldNdx] = '\0';
      if ((strcmp(qfield,q[k].FV)==0)|| (strcmp(qfield,q[k].FV) < 0))
      {
         count++;
         *(ptree+pr)=j;
         pr++;
      }
      lseek(hfd,(total+7-WFieldNdx),SEEK_CUR);
  }
  if (count==0L)
     return NULL;

  return(ptree);
}

/* ---------------------------------------------------------- */
```

ภาคผนวก ข

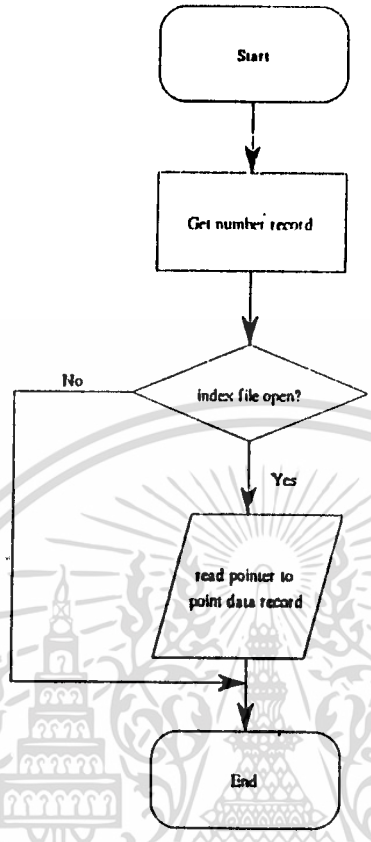
FLOWCHART



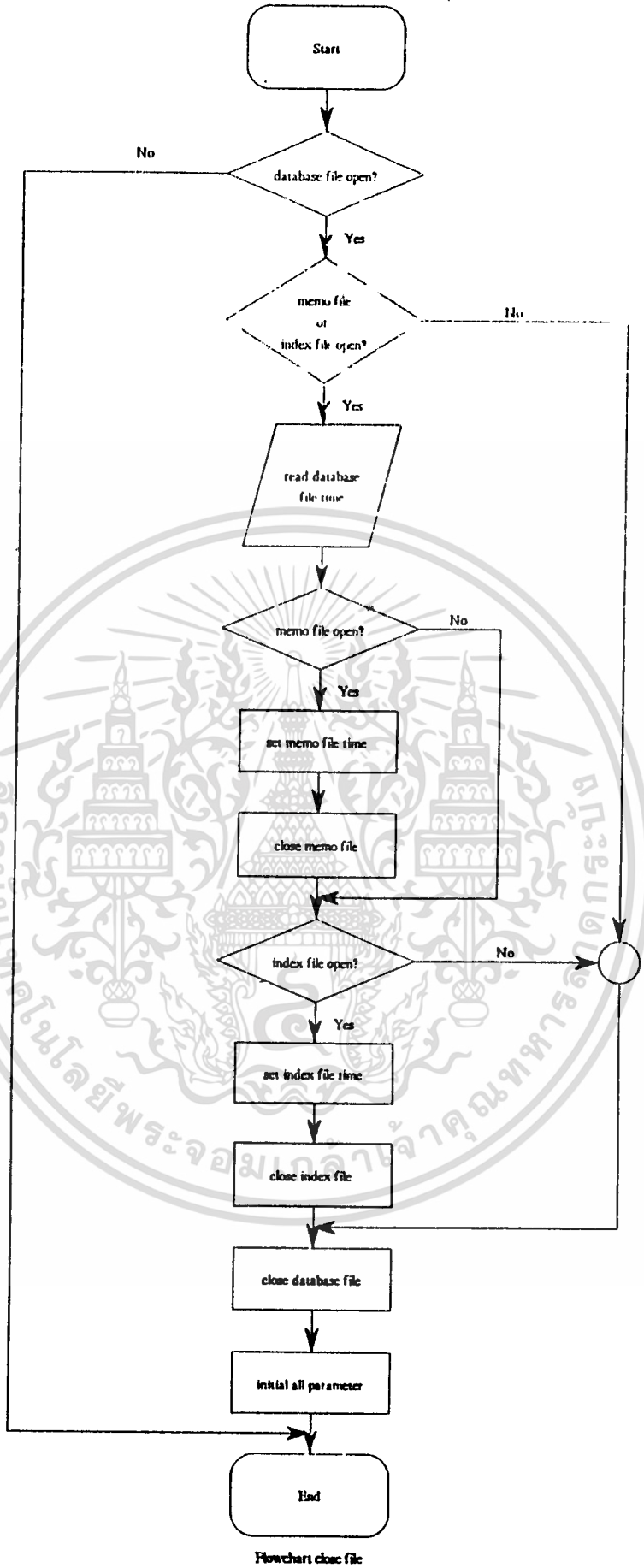
Flowchart appen to database file



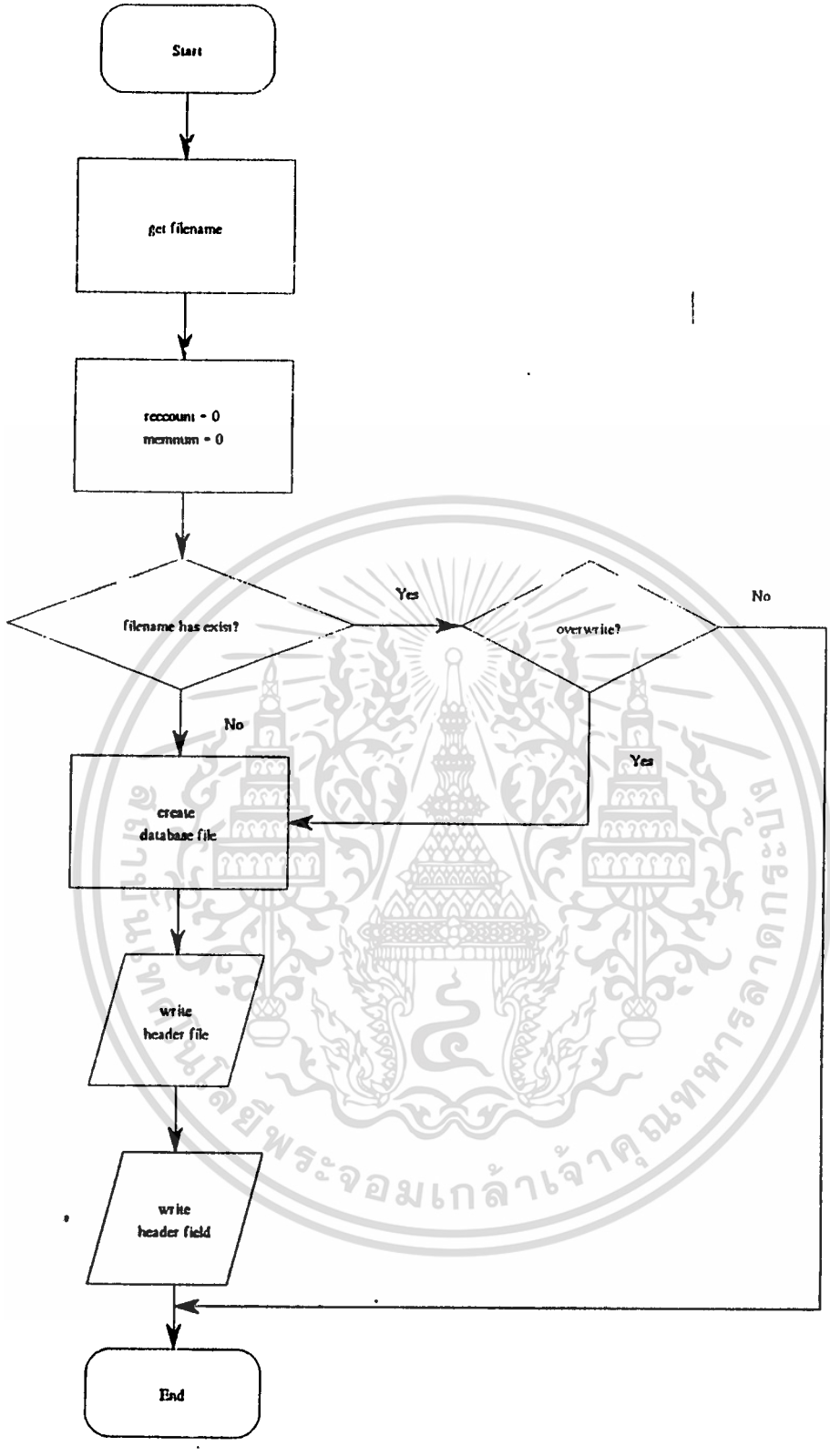
Flowchart function check_ndx



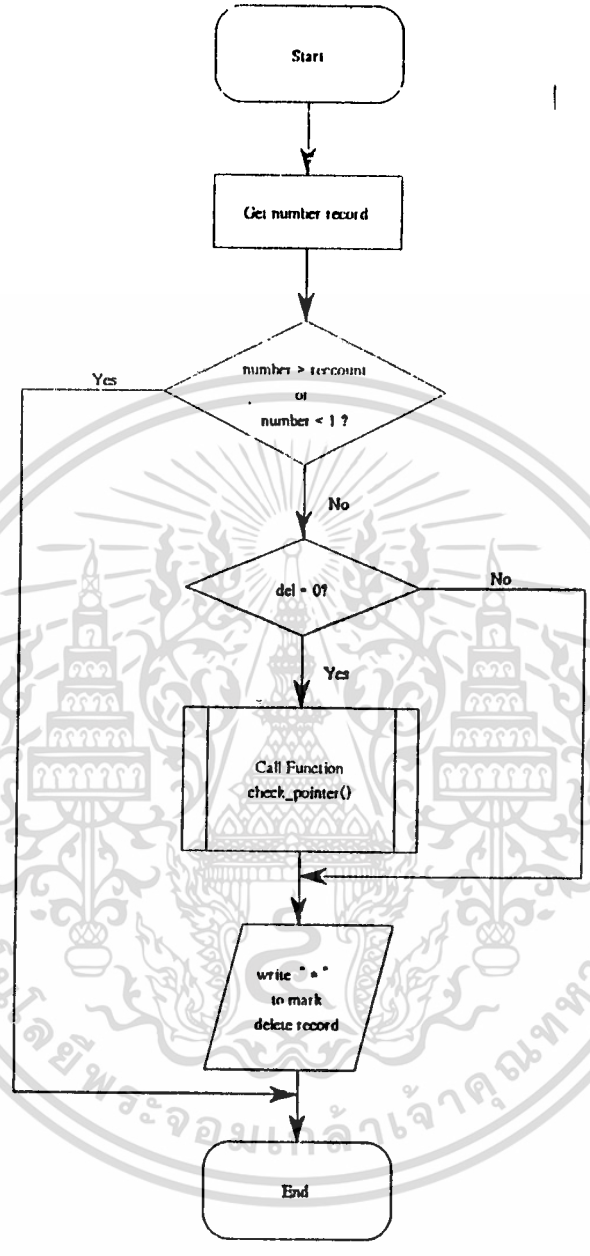
Flowchart function check_pointer



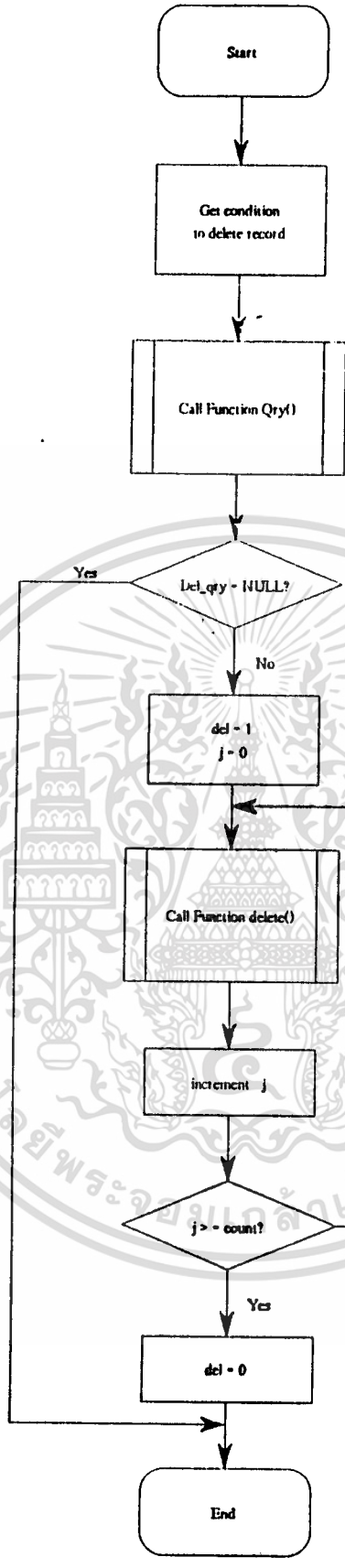
Flowchart close file



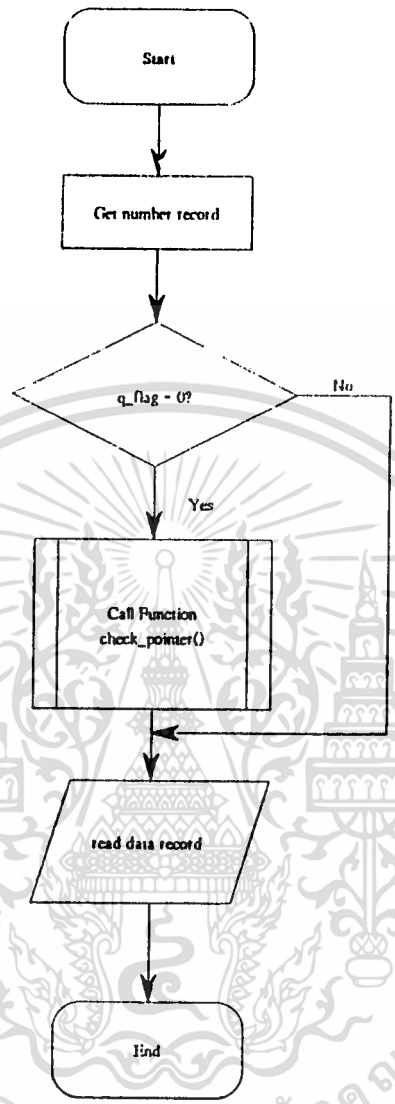
Flowchart create database file



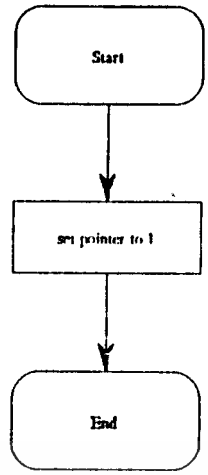
Flowchart delete record



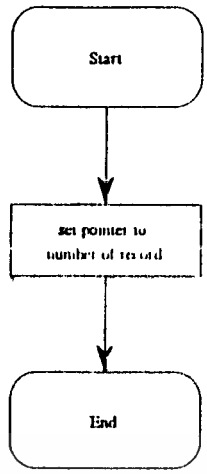
Flowchart delete condition



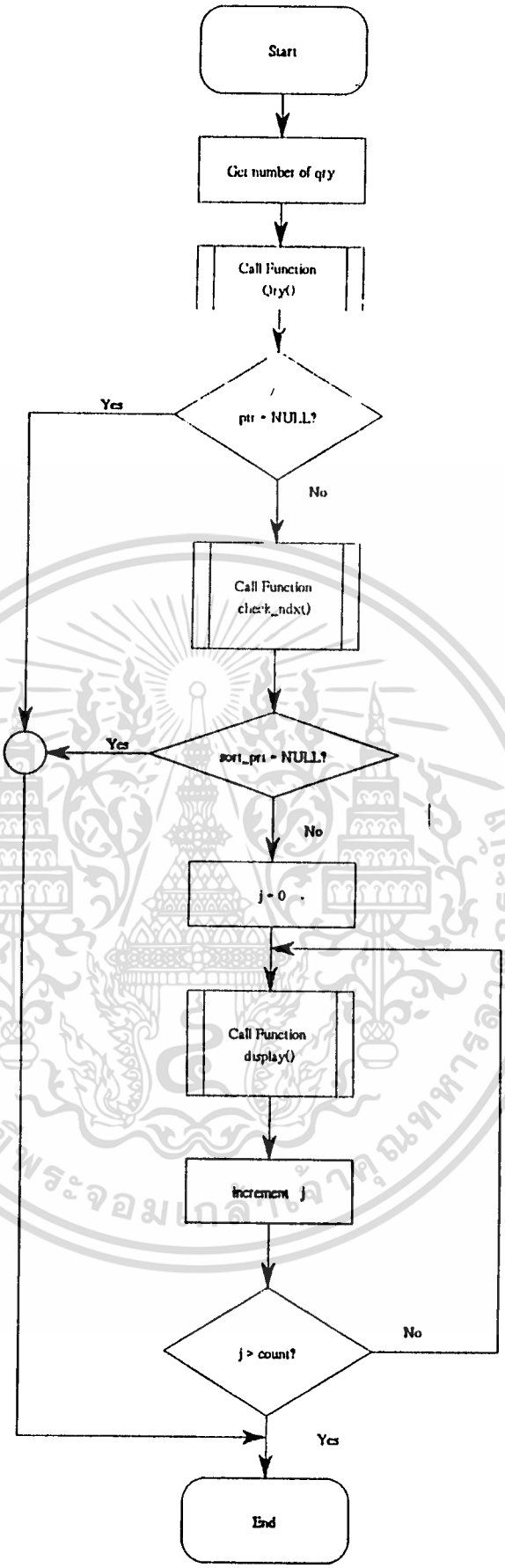
Flowchart display record



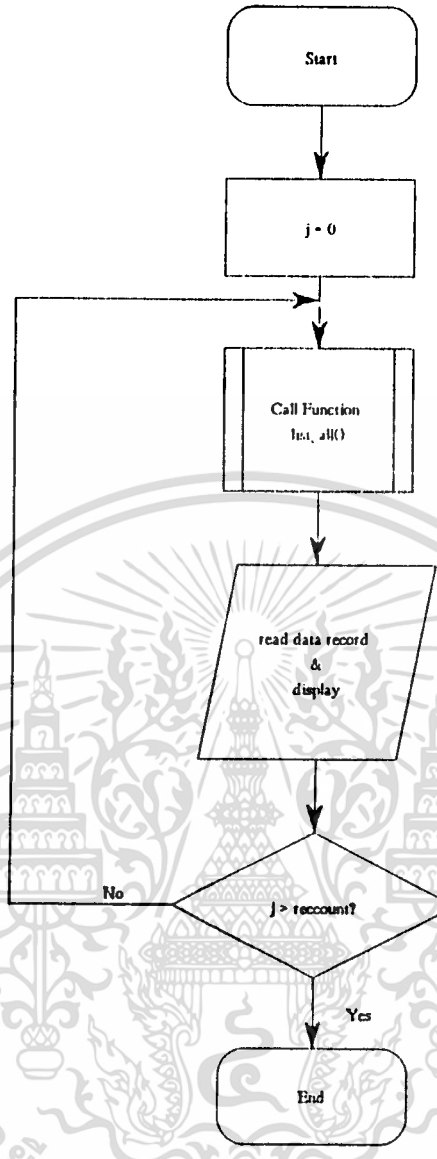
Flowchart point to first record



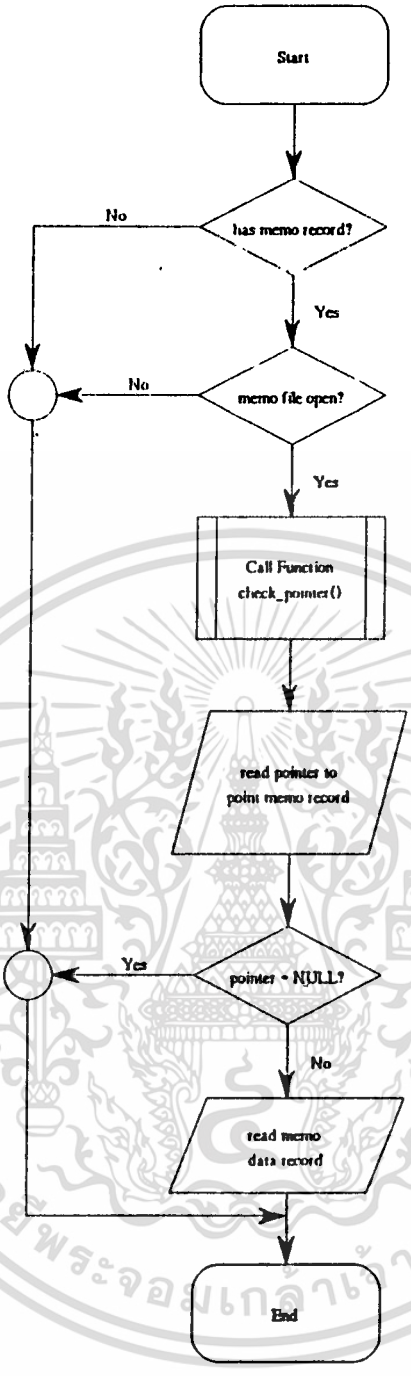
Flowchart point to last record



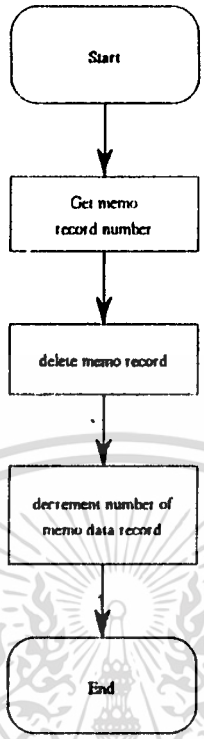
Flowchart list_condition



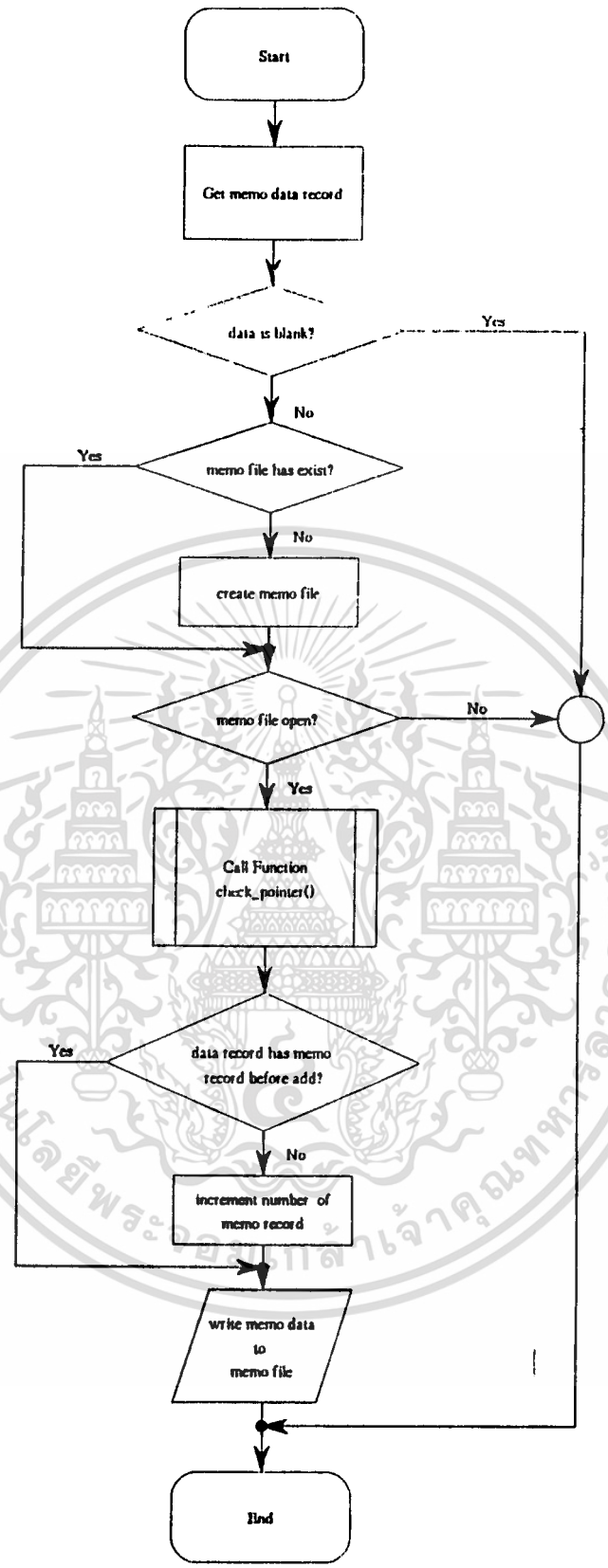
Flowchart list all record



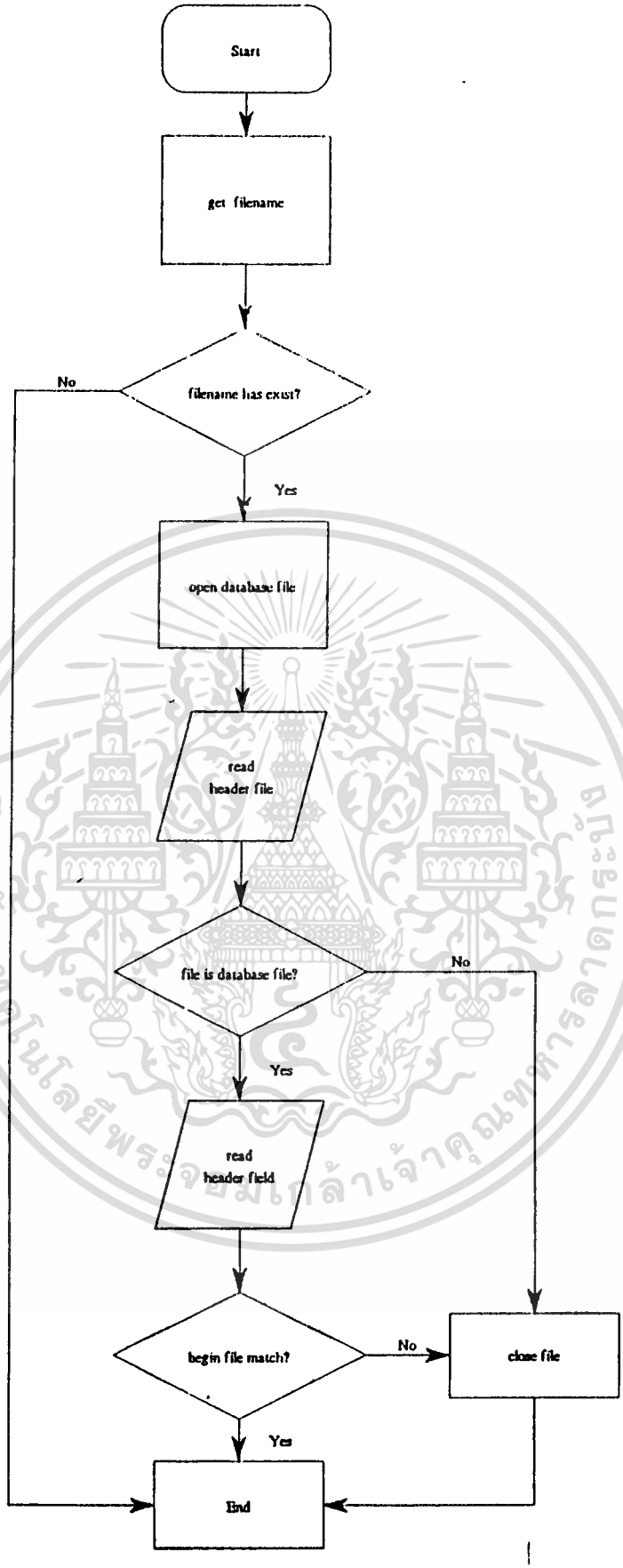
Flowchart memo display



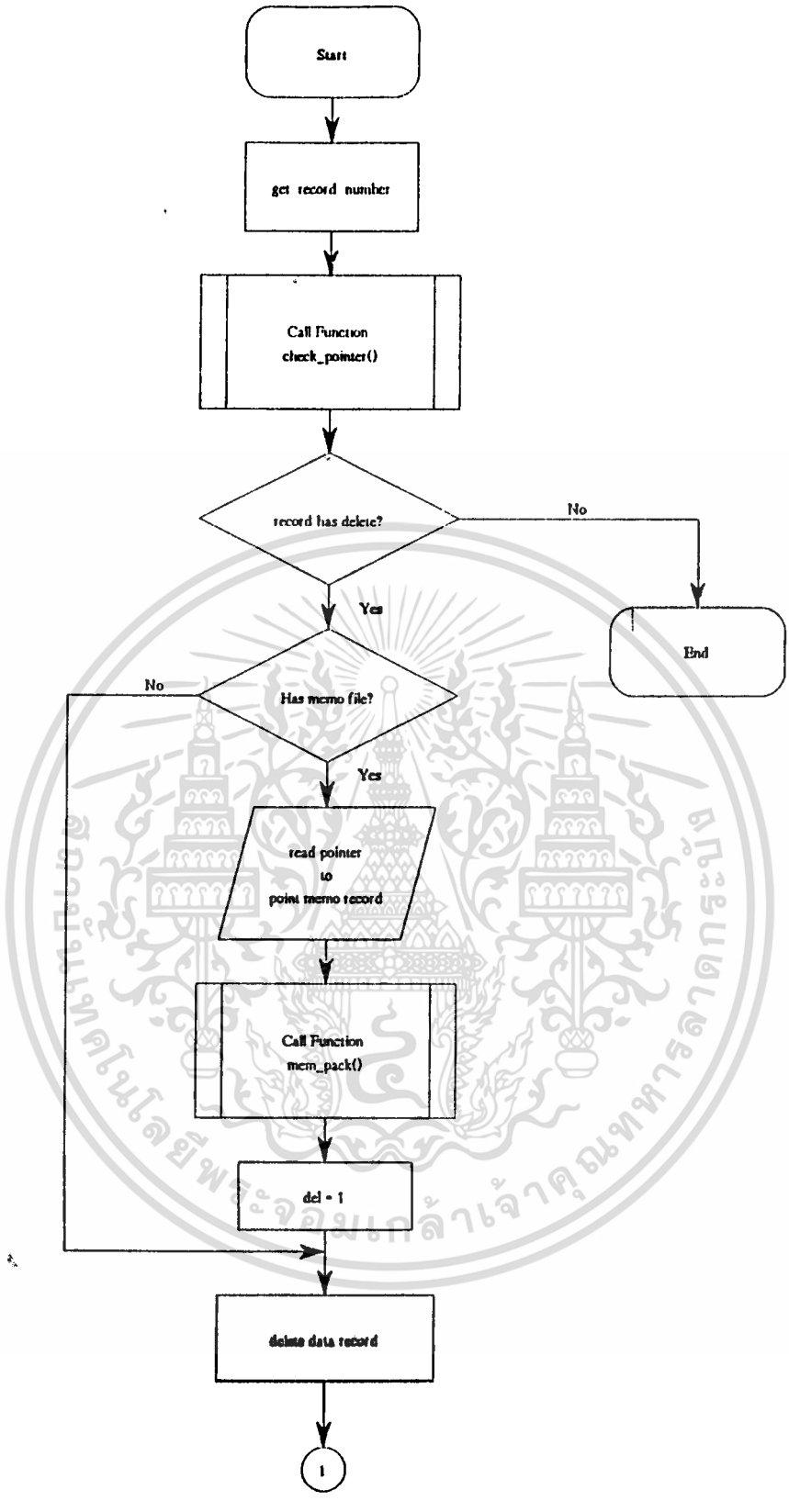
Flowchart delete memo record



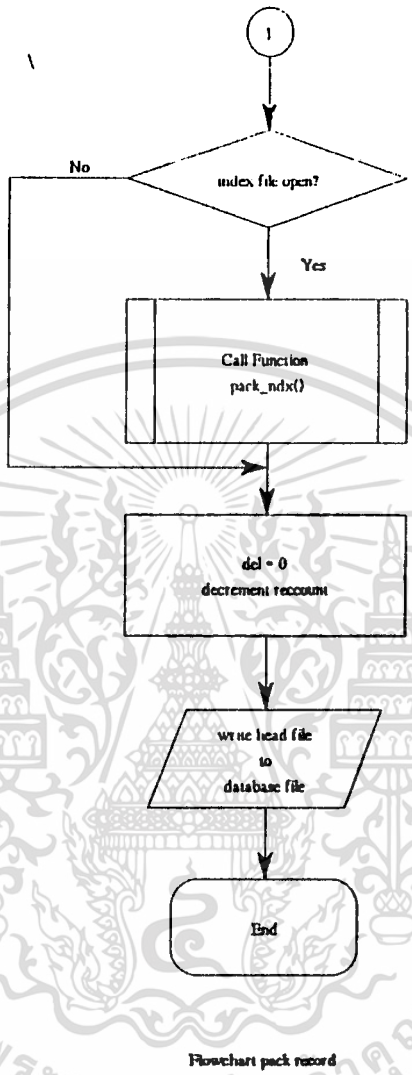
Flowchart add memo data record



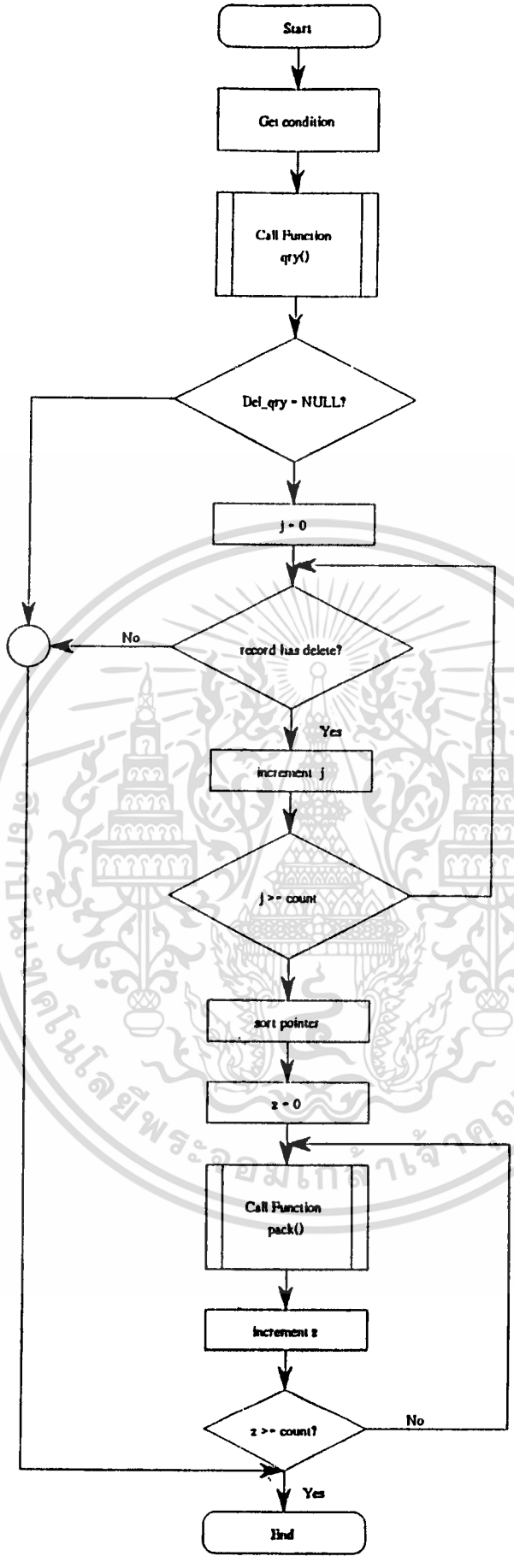
Flowchart open database file



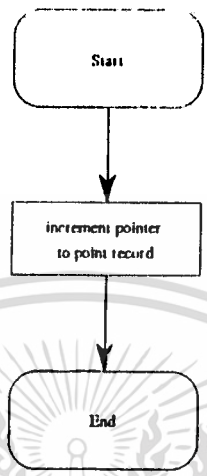
Flowchart pack record



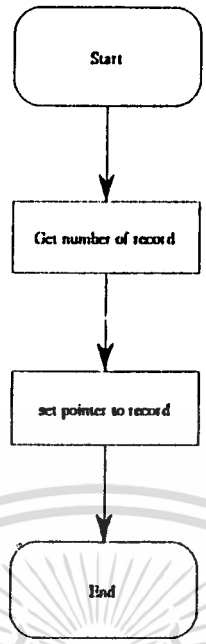
Flowchart pack record



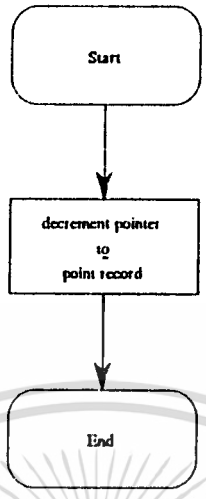
Flowchart pack condition



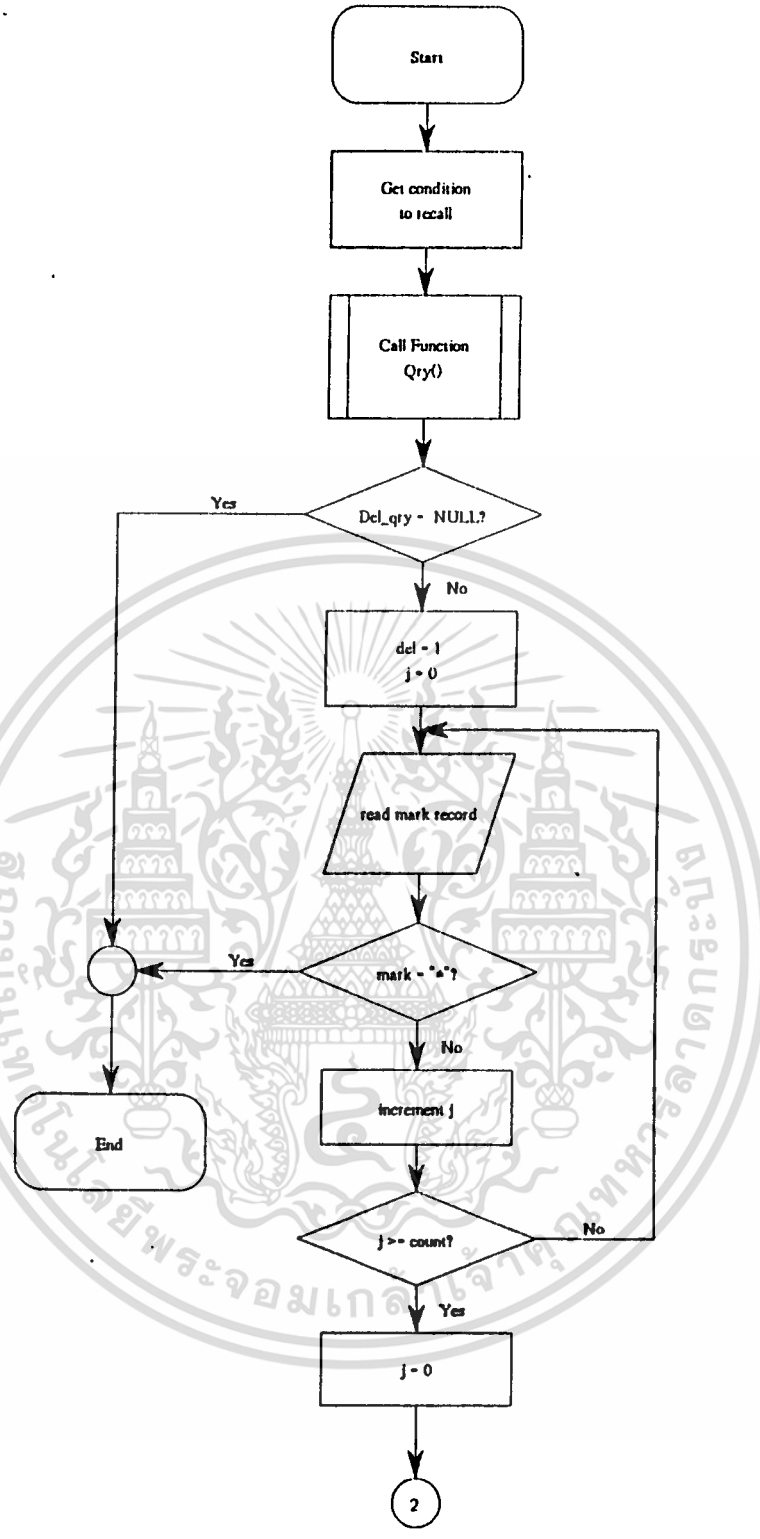
Flowchart point to next record



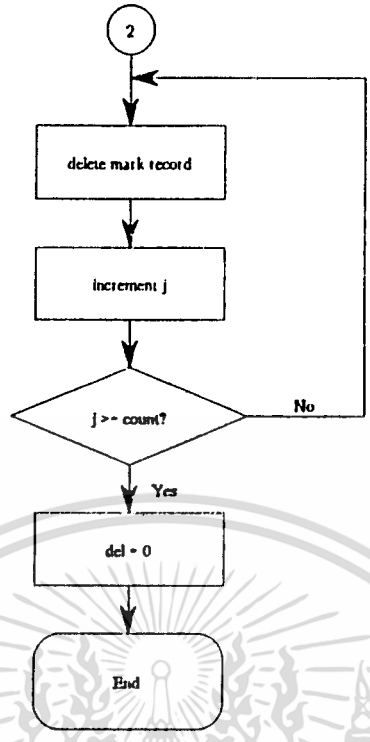
Flowchart point to record



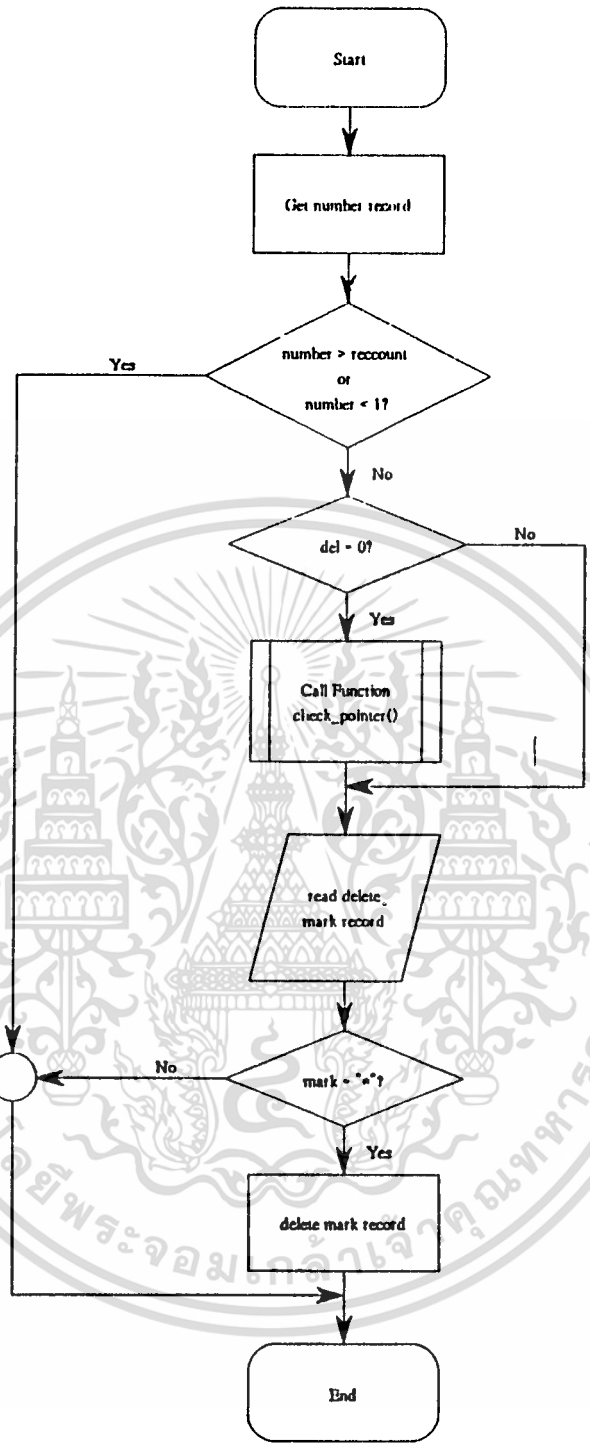
Flowchart point to previous record



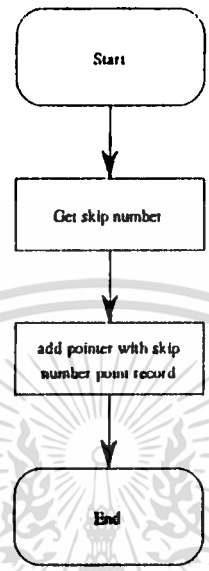
Flowchart recall condition



Flowchart recall condition

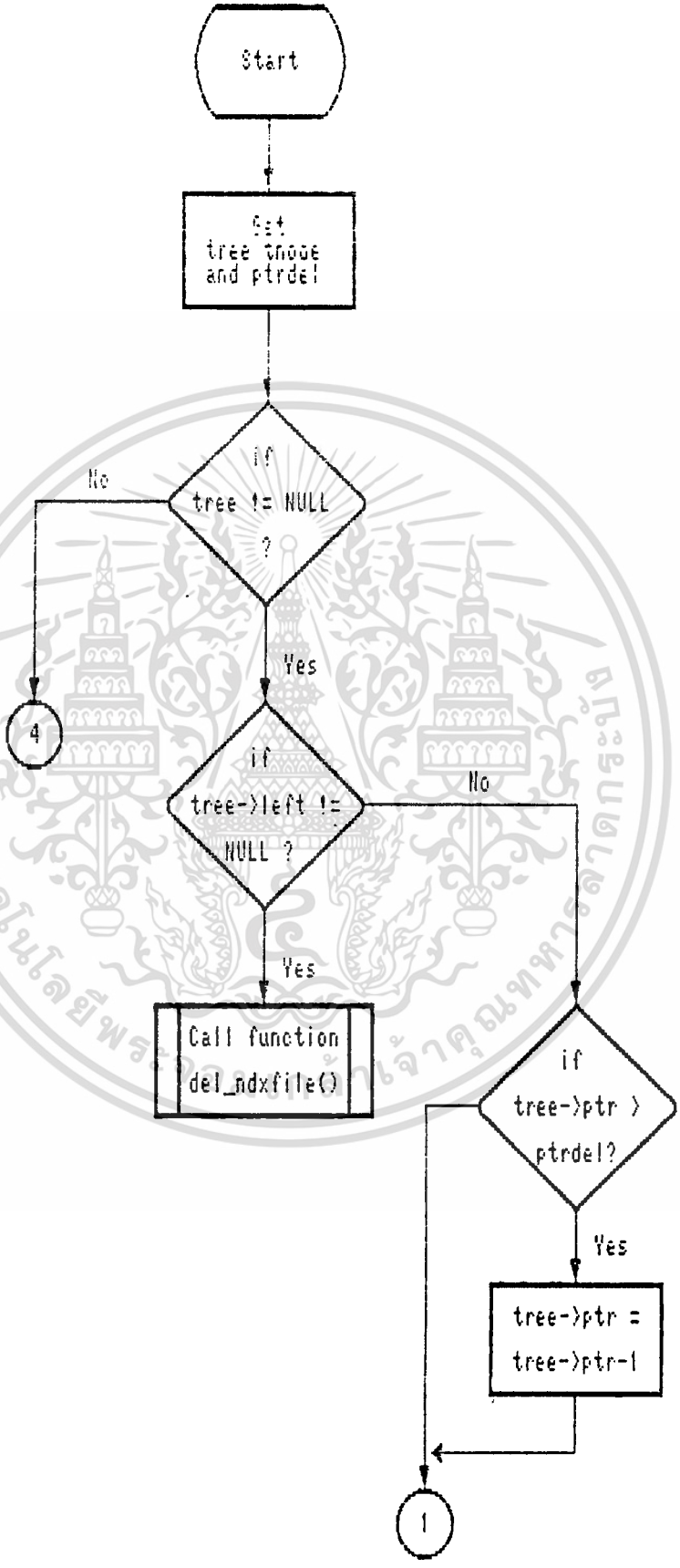


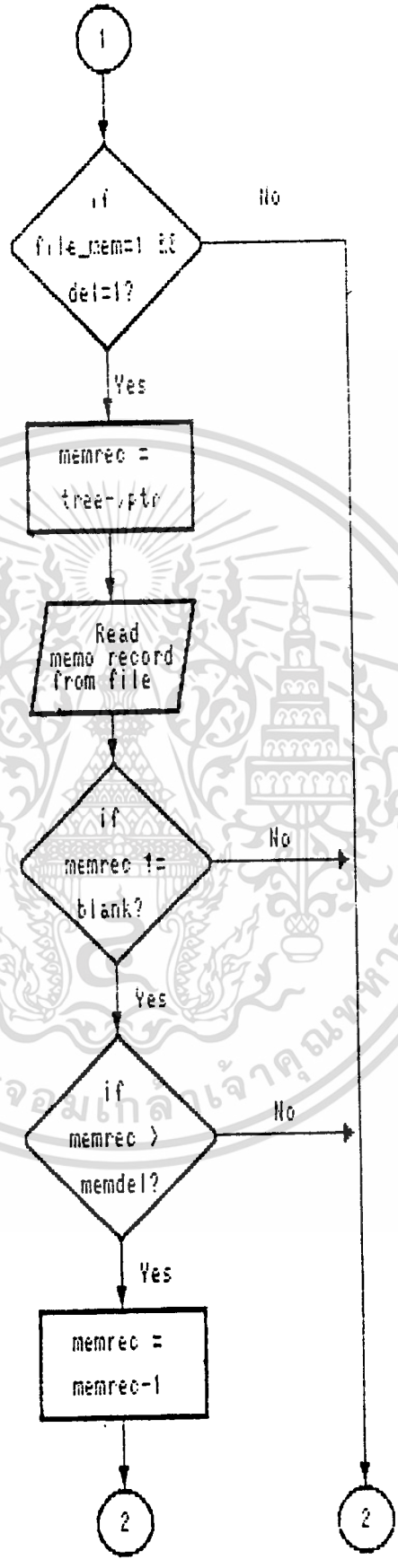
Flowchart recall record

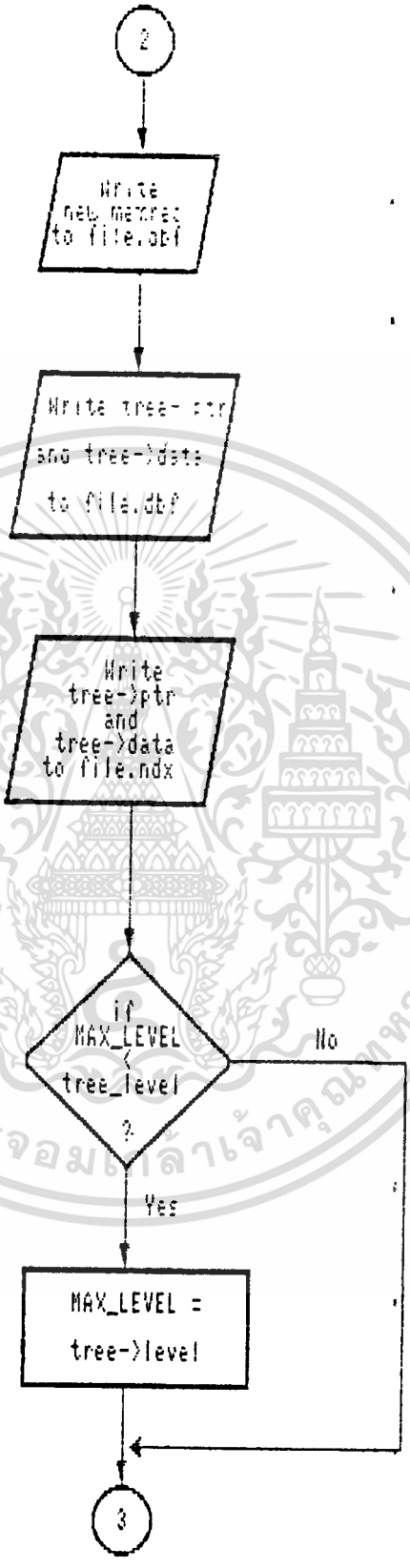


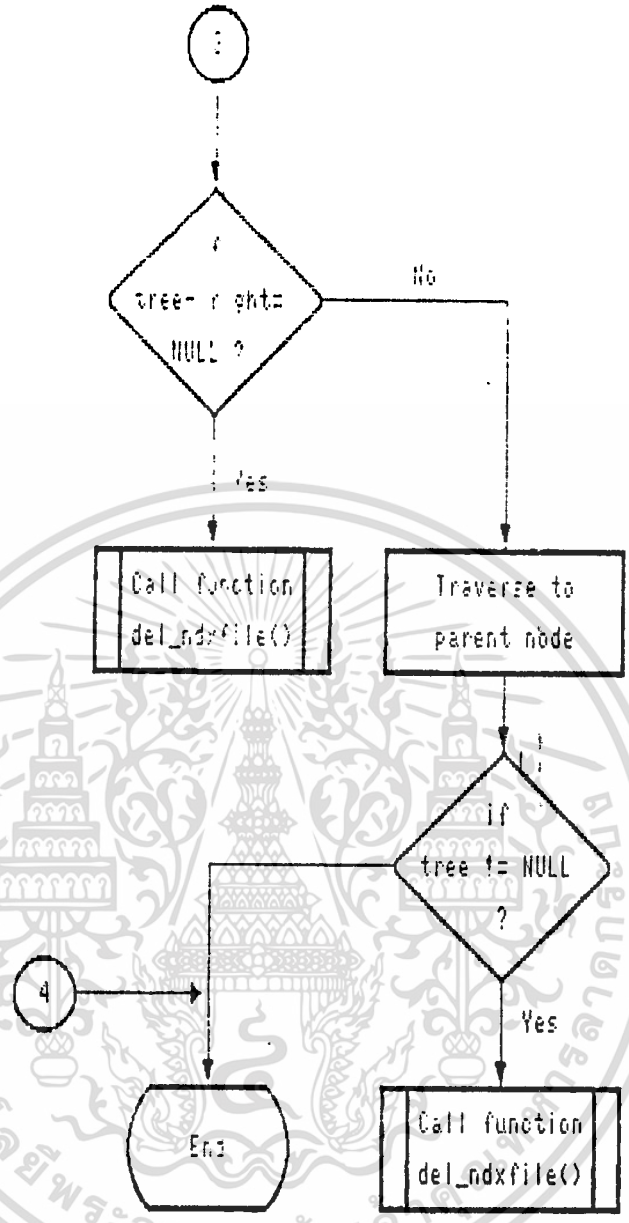
Flowchart point to skip record

Routine del_ndxfile

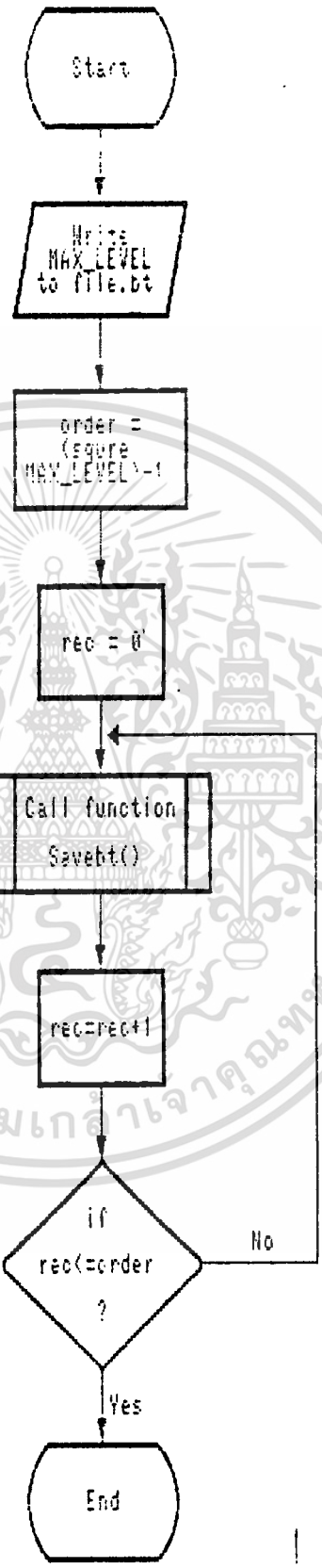


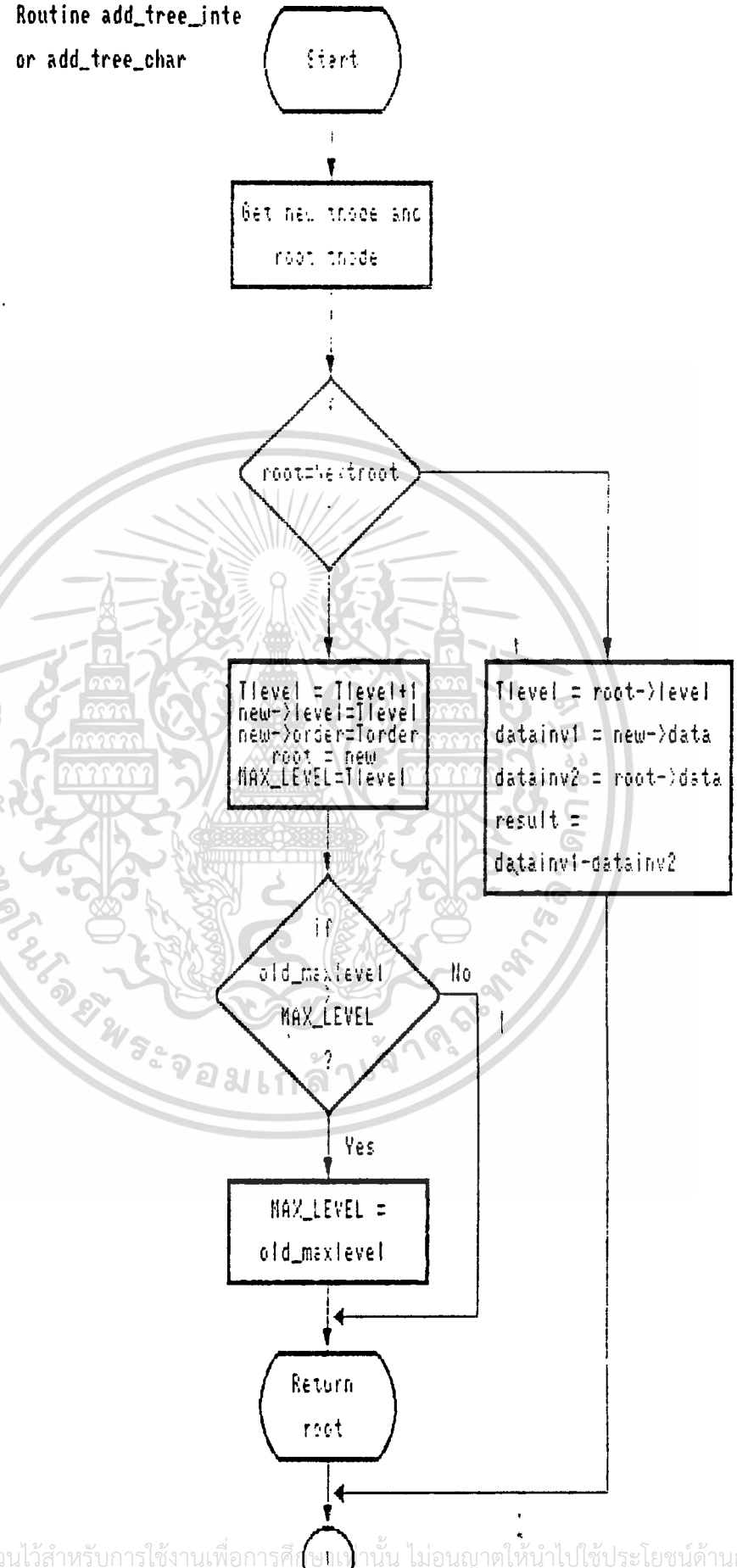


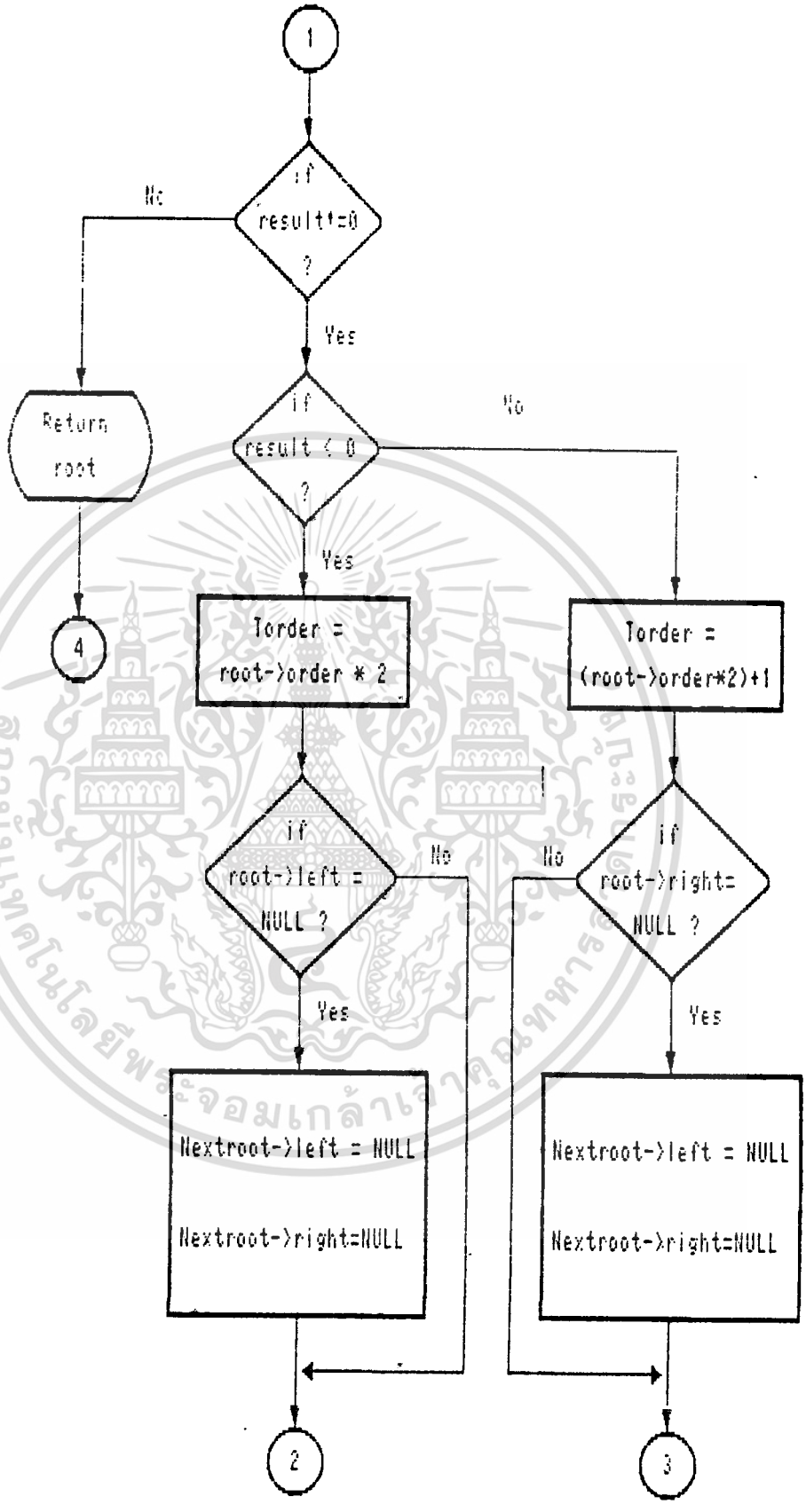


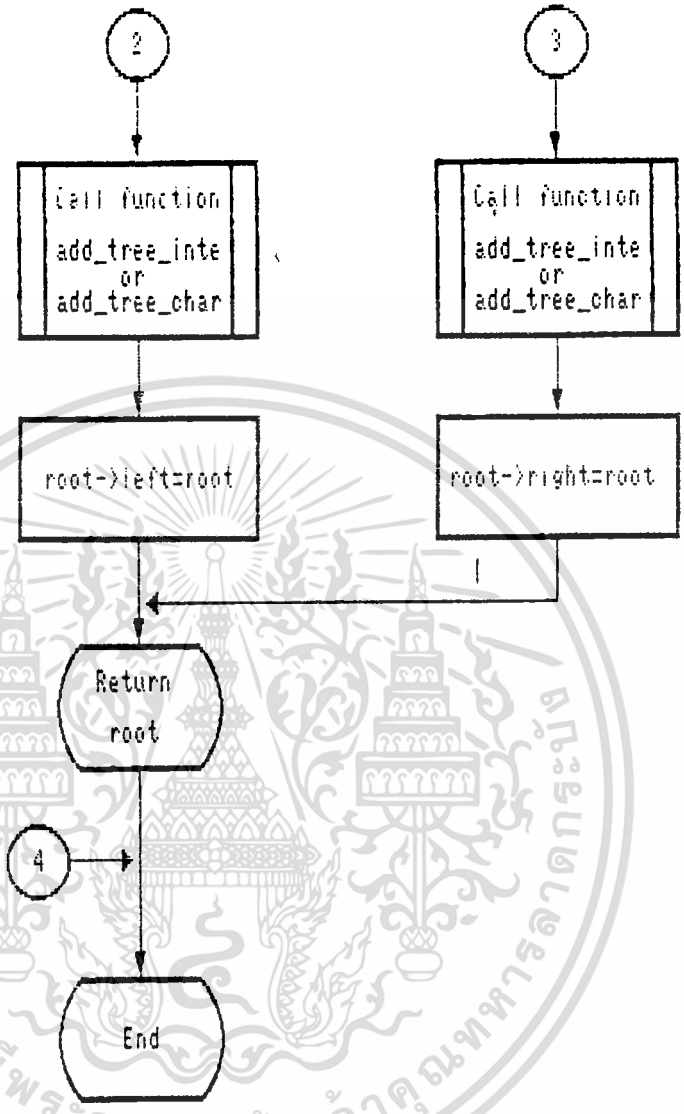


Routine Hrbtree

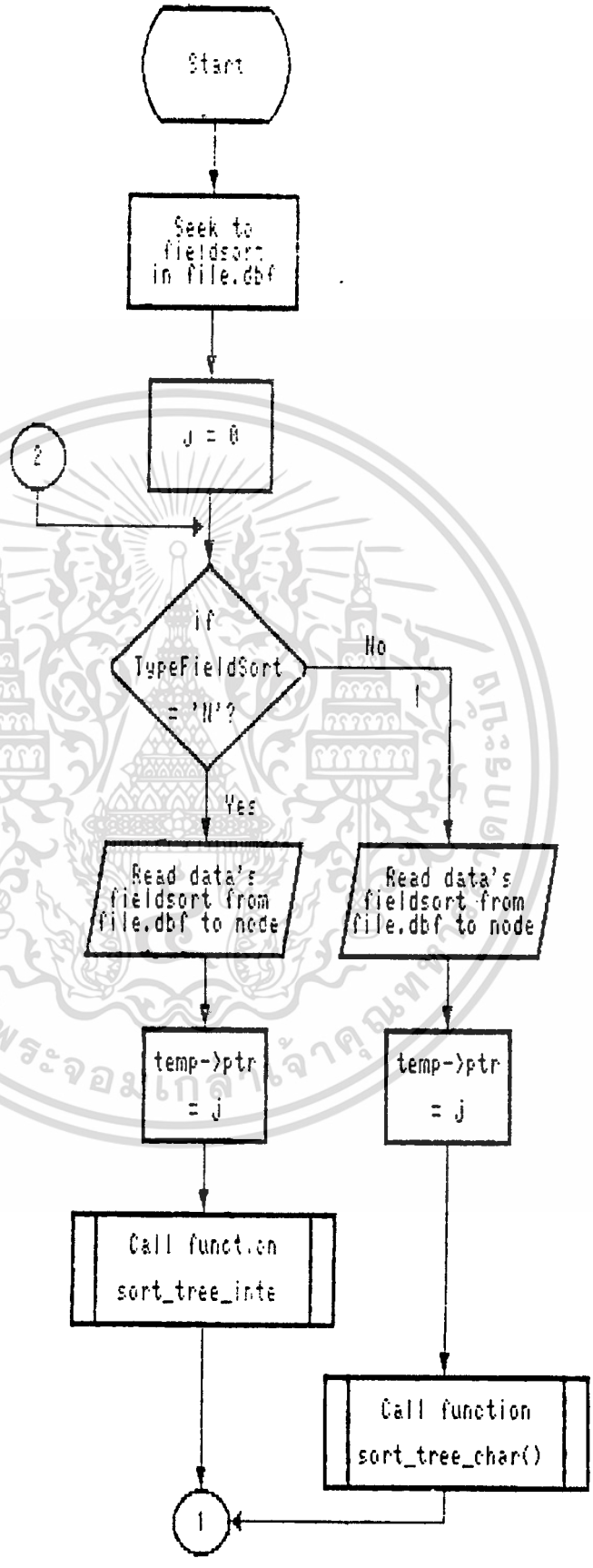


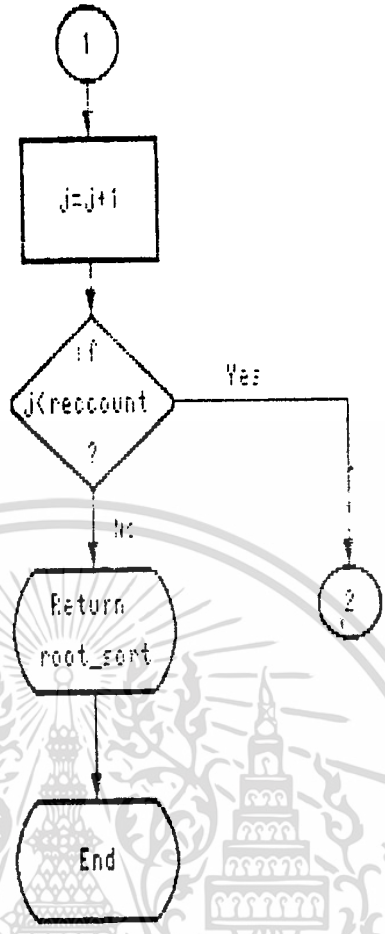




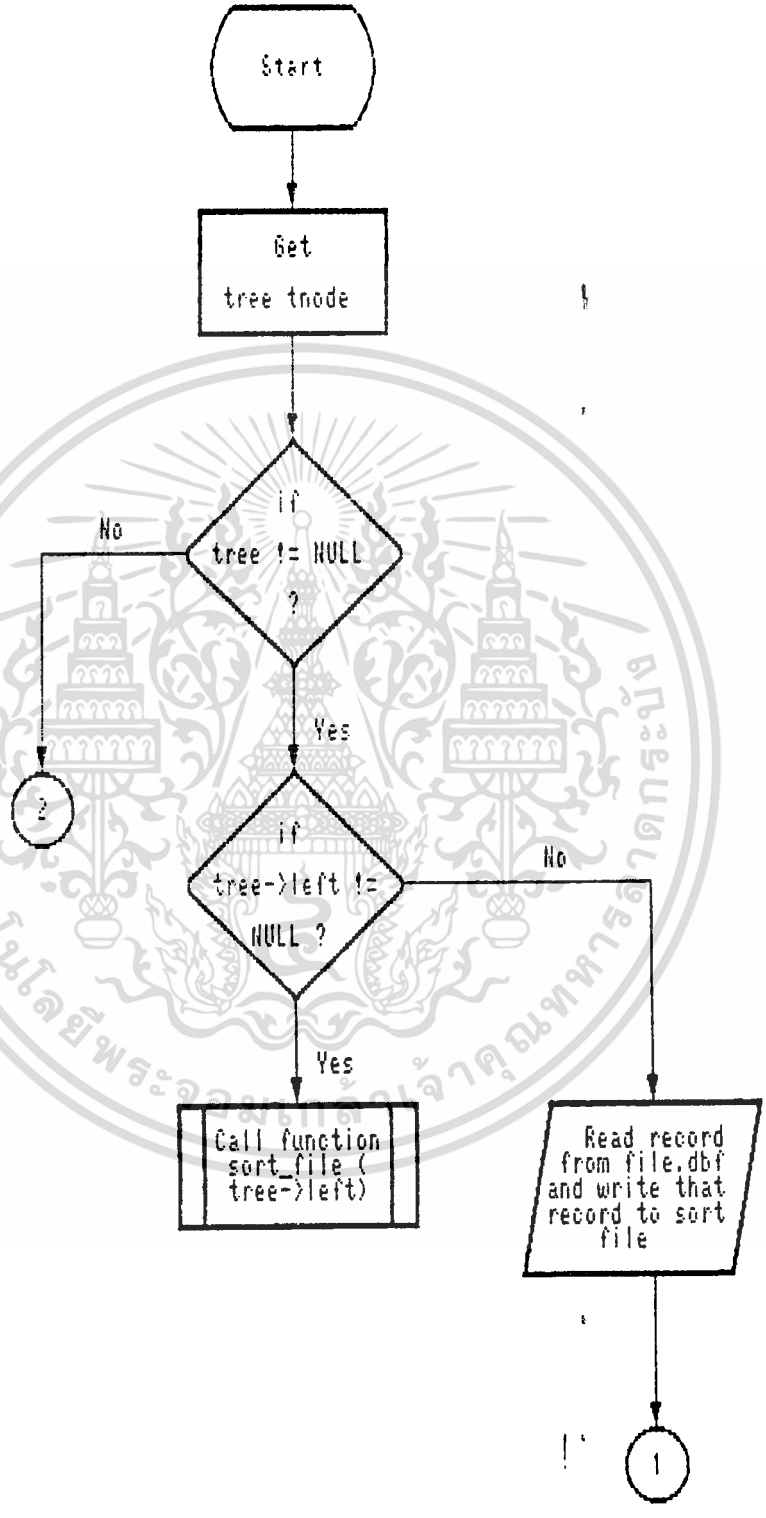


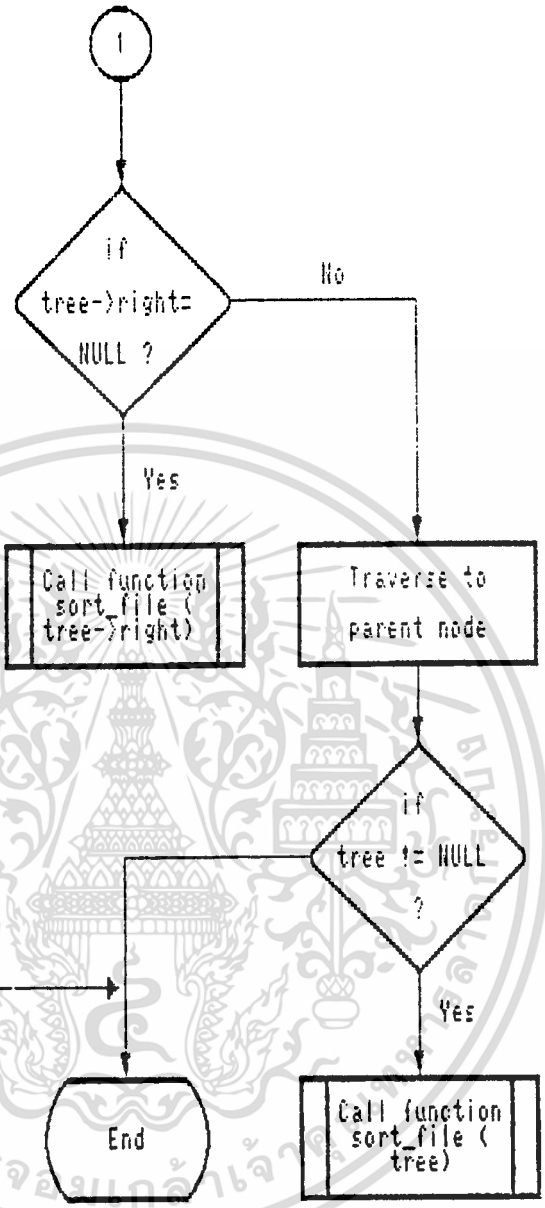
Routine sort_tree

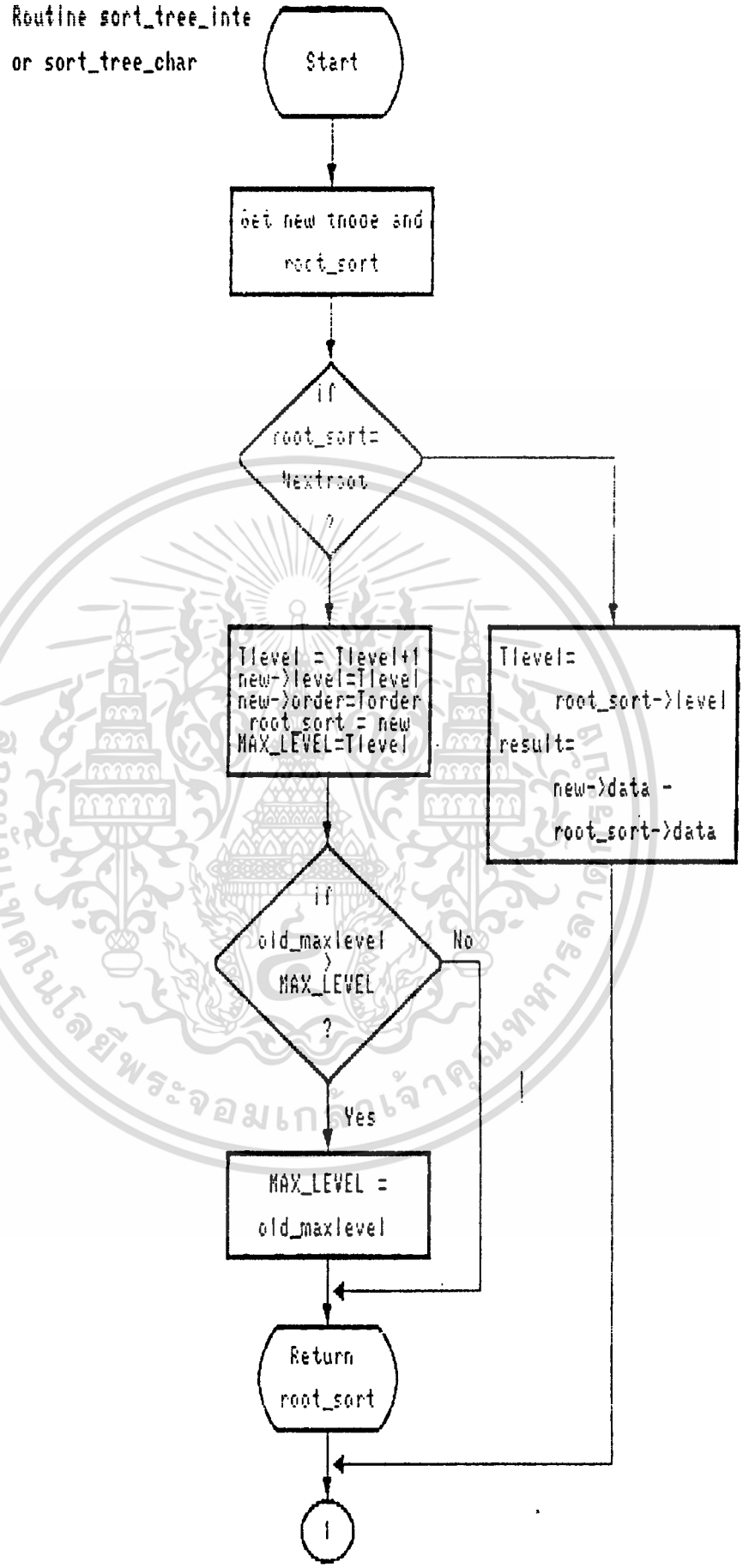


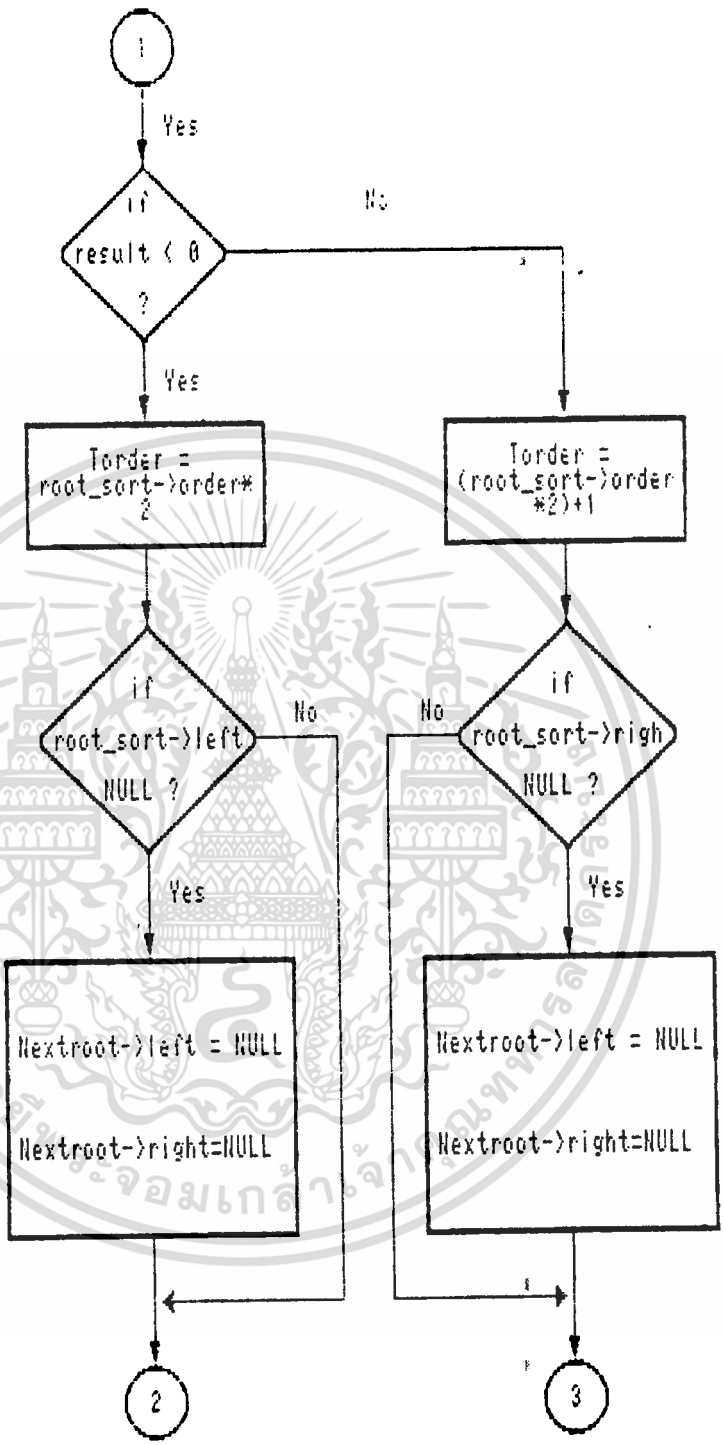


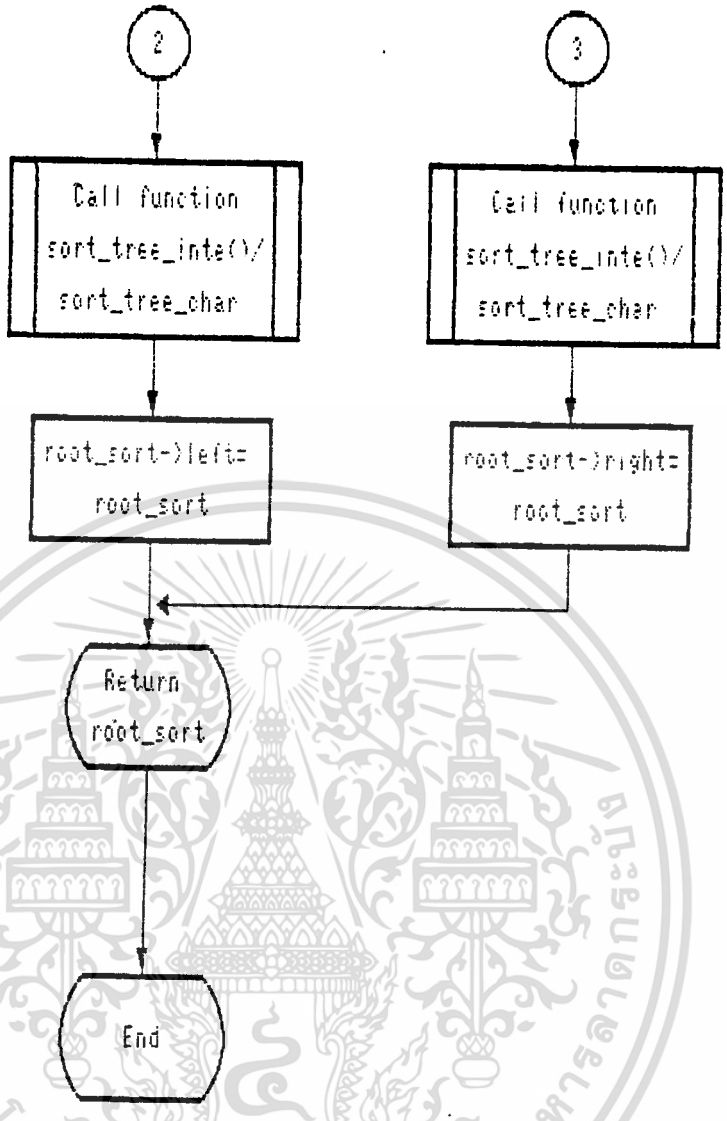
Routine sort_file



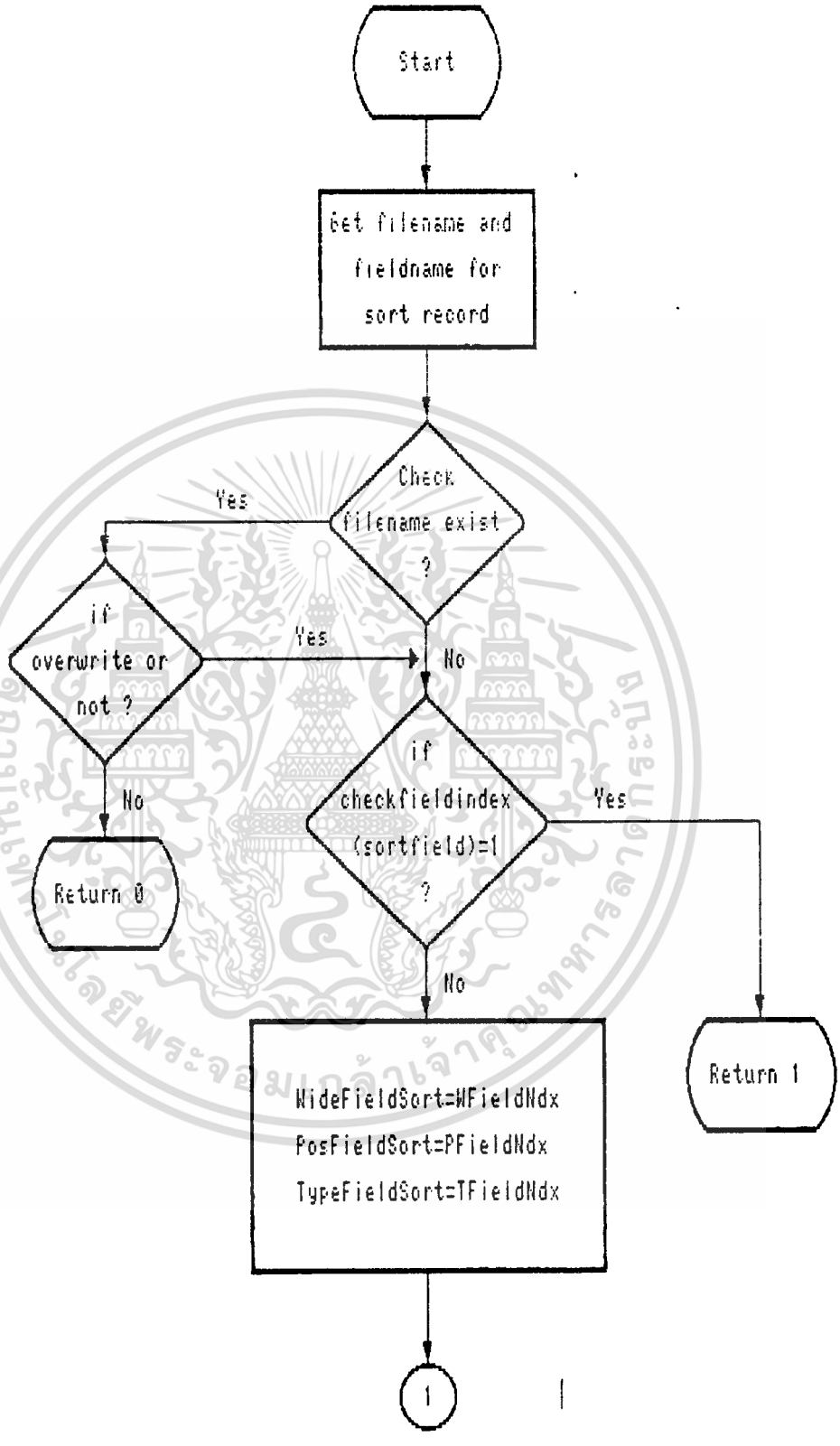


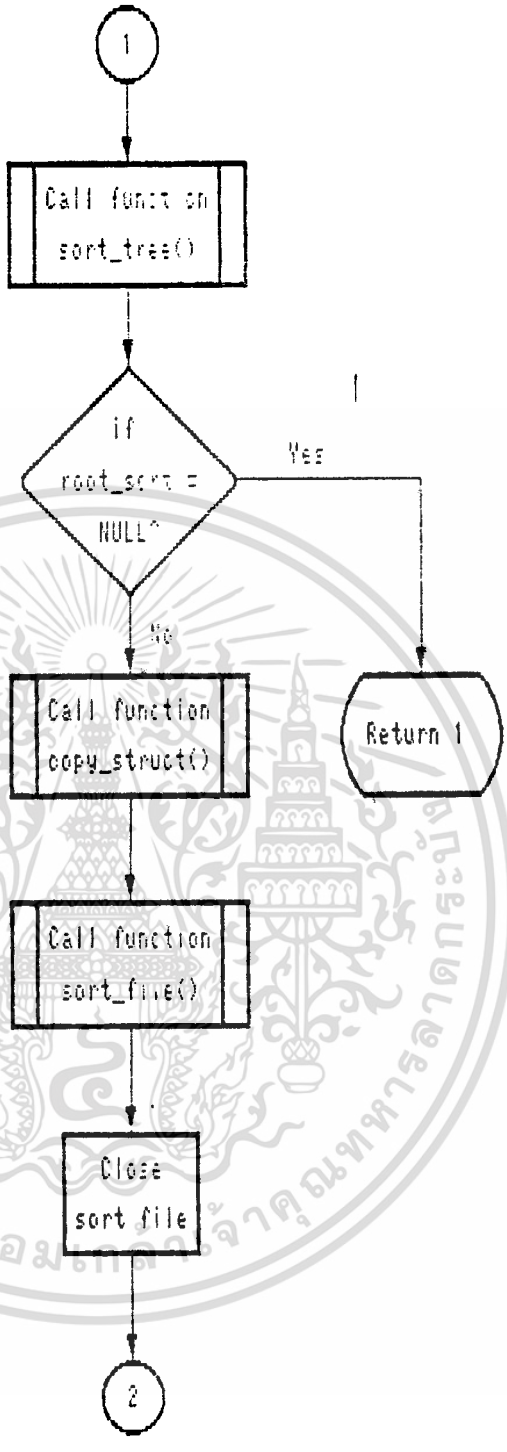


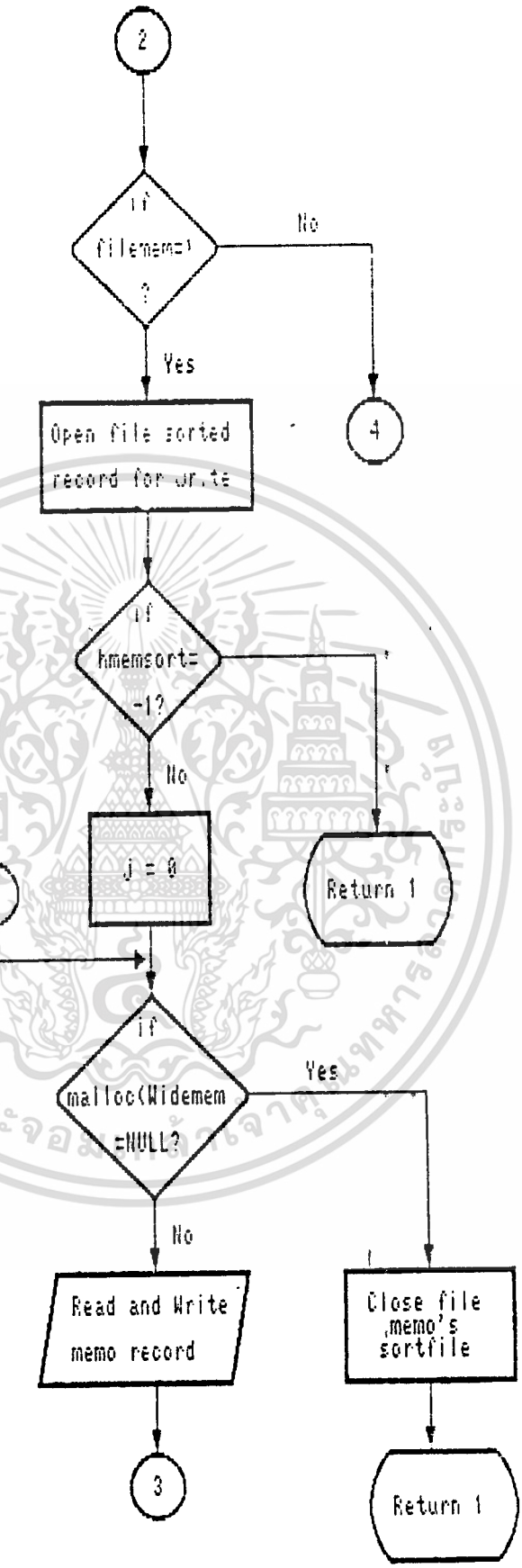


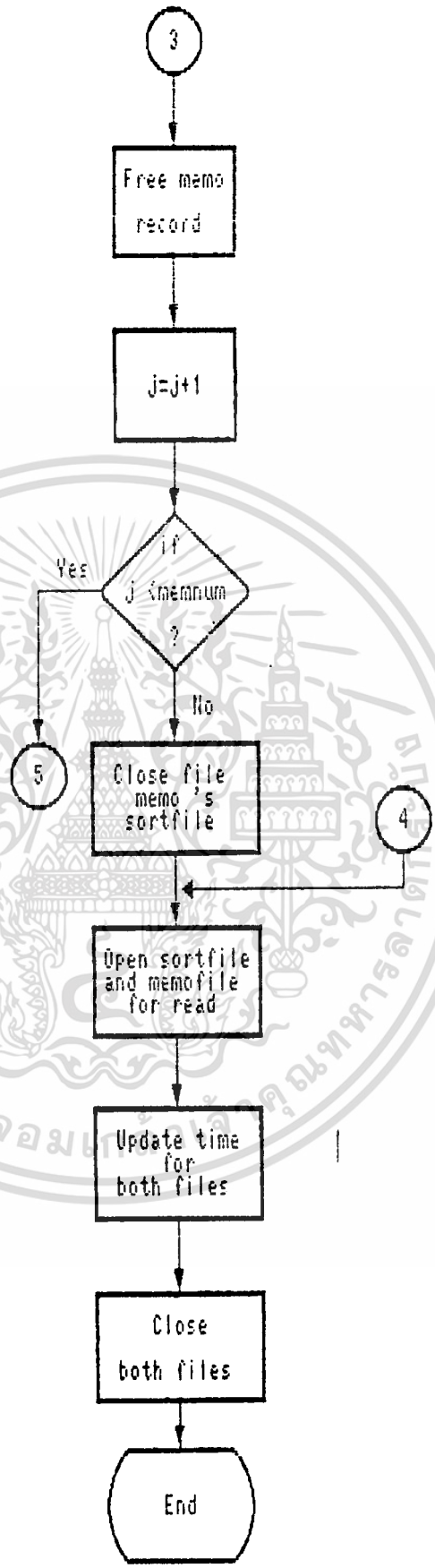


Routine sort

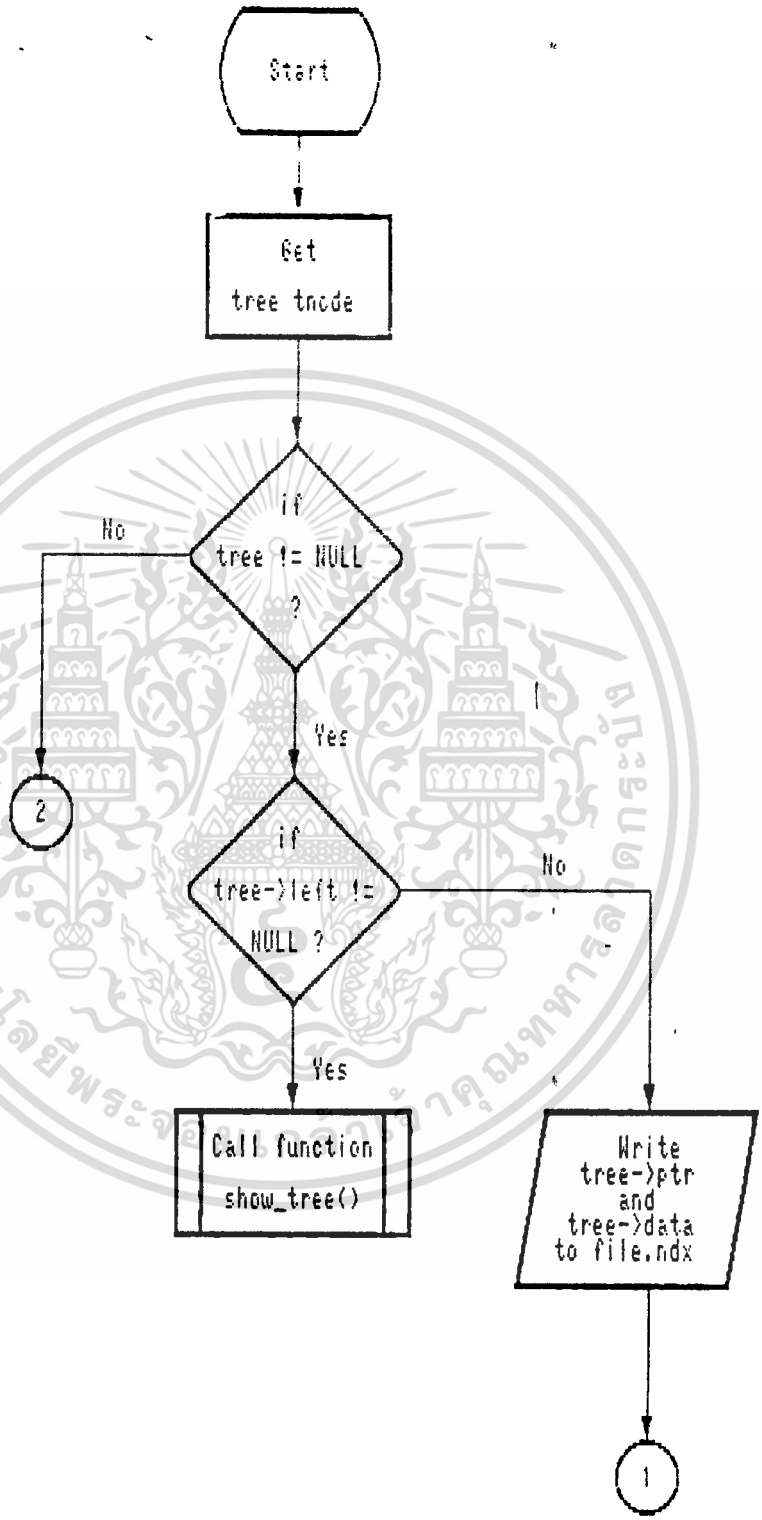


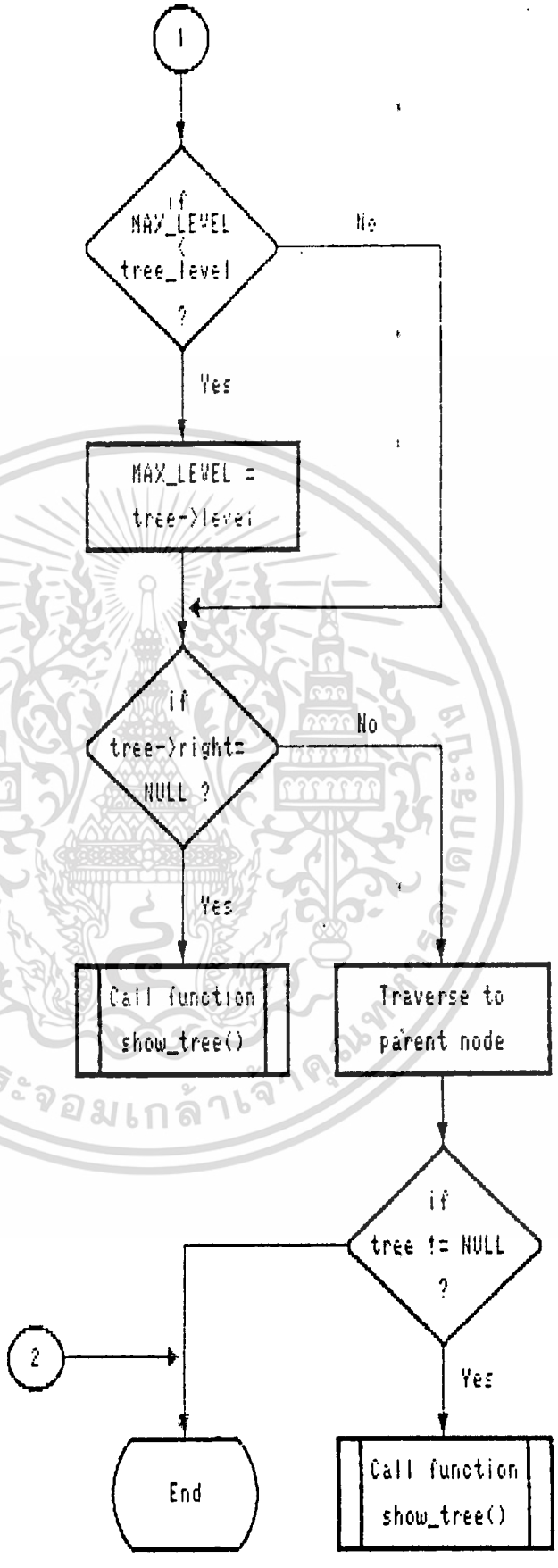




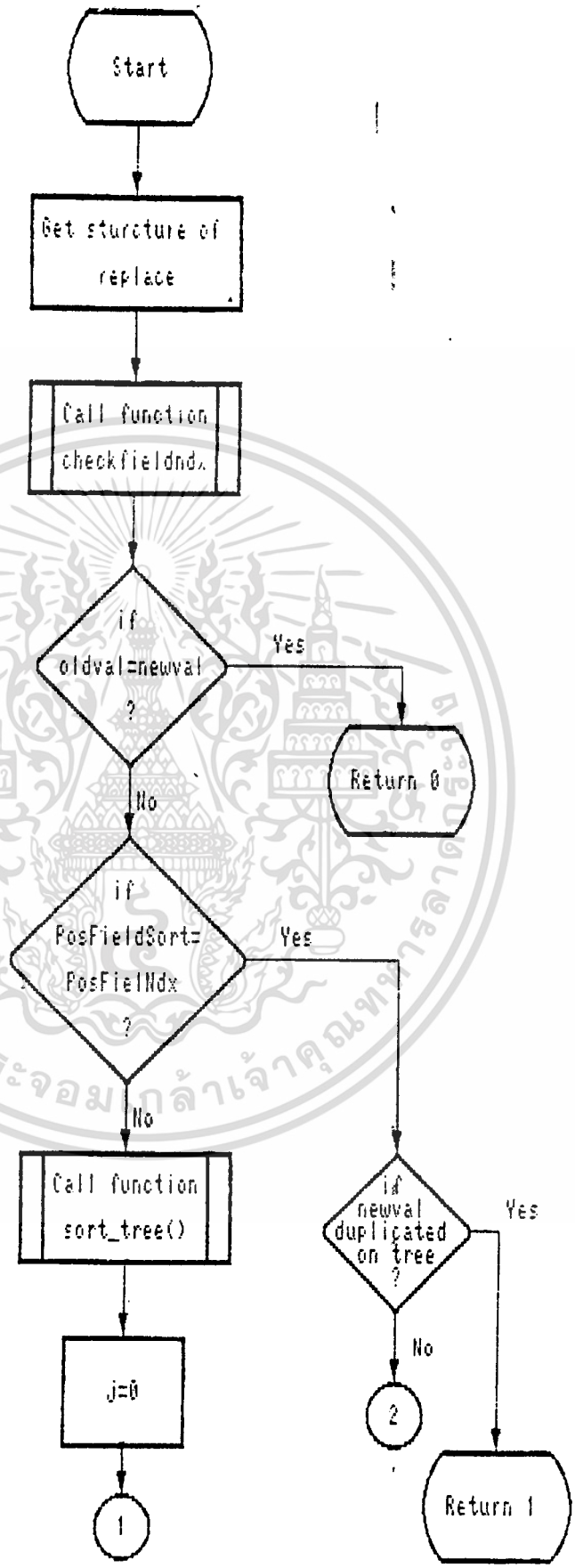


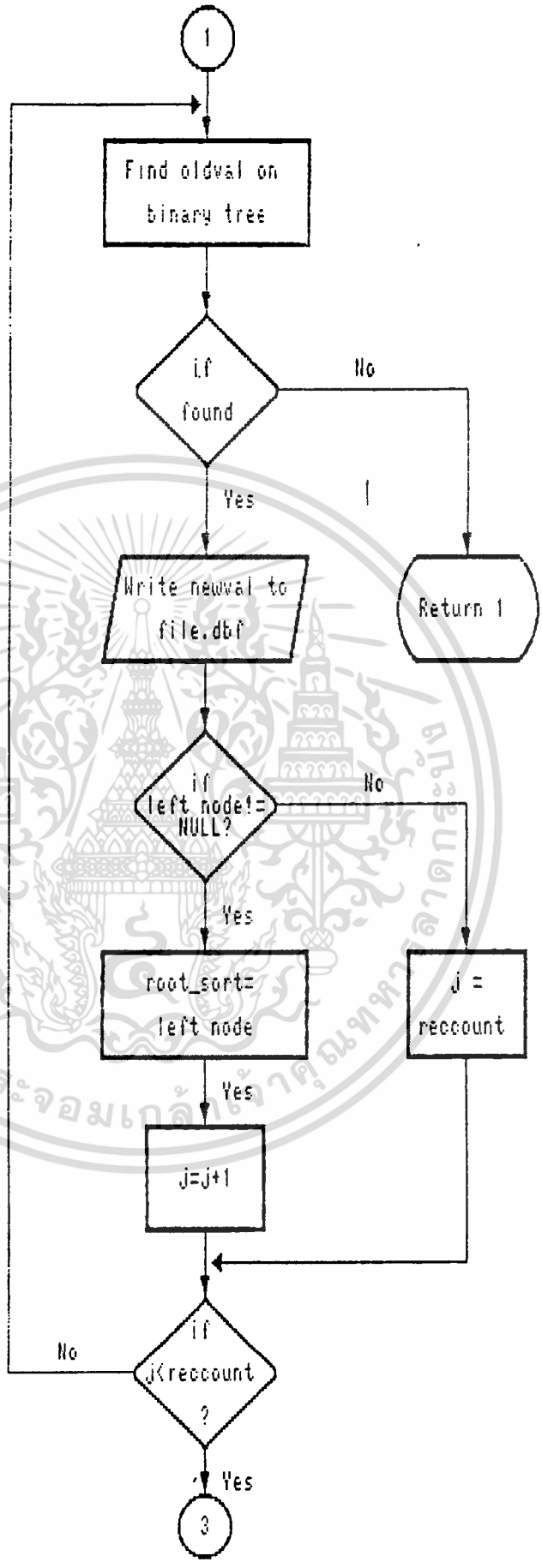
Routine show_tree

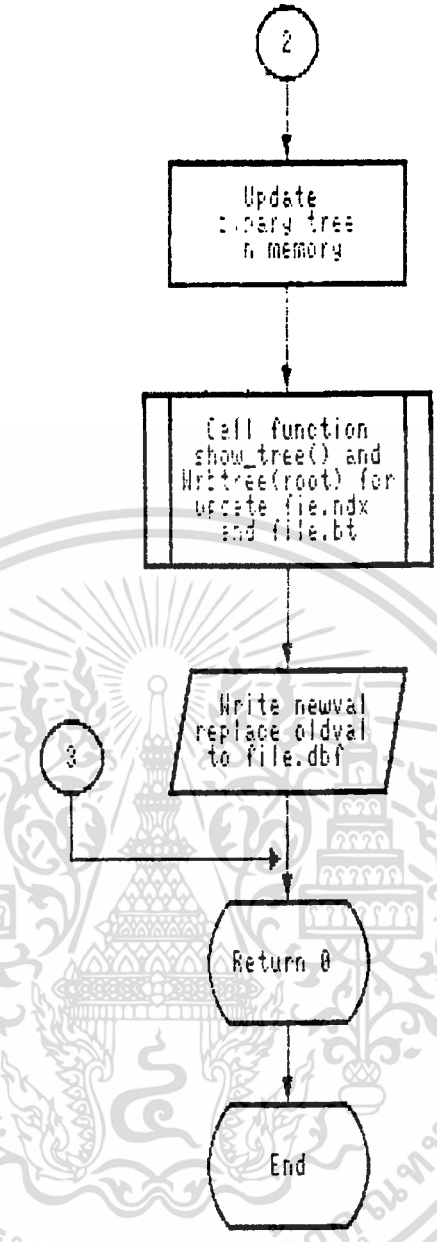




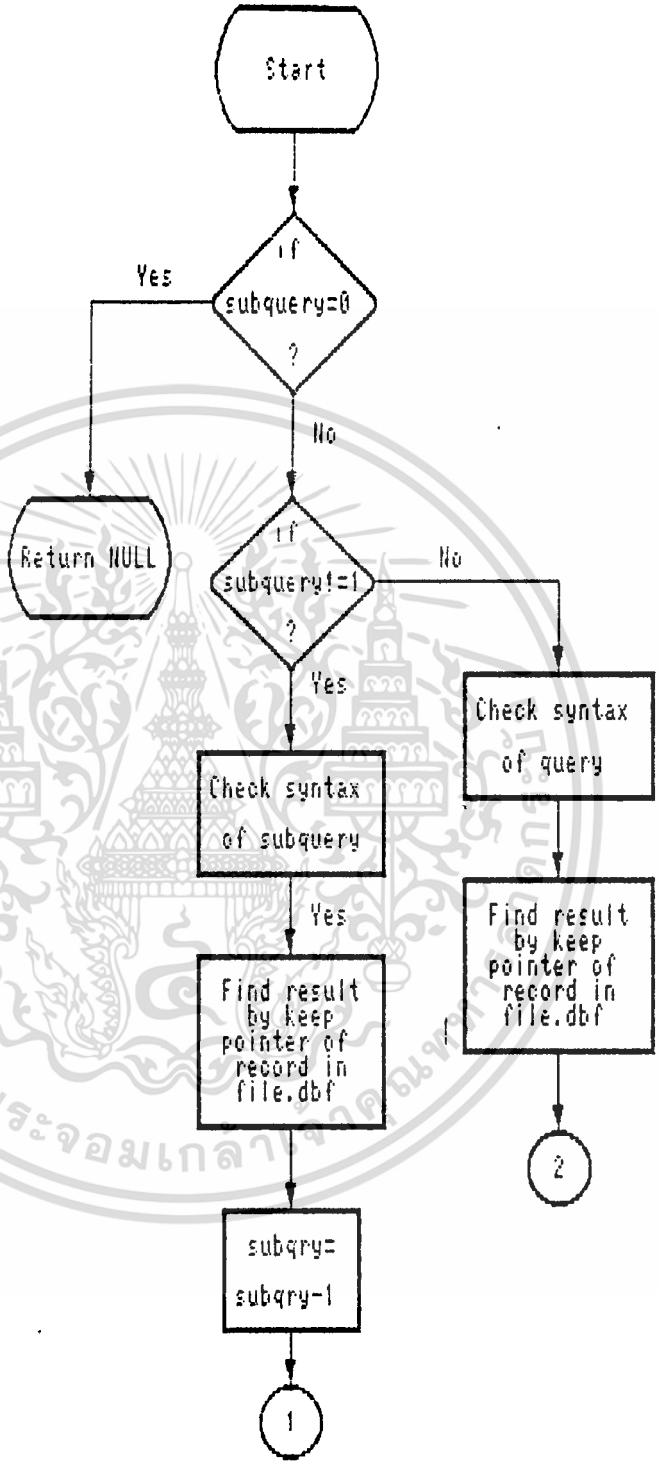
Routine Replace

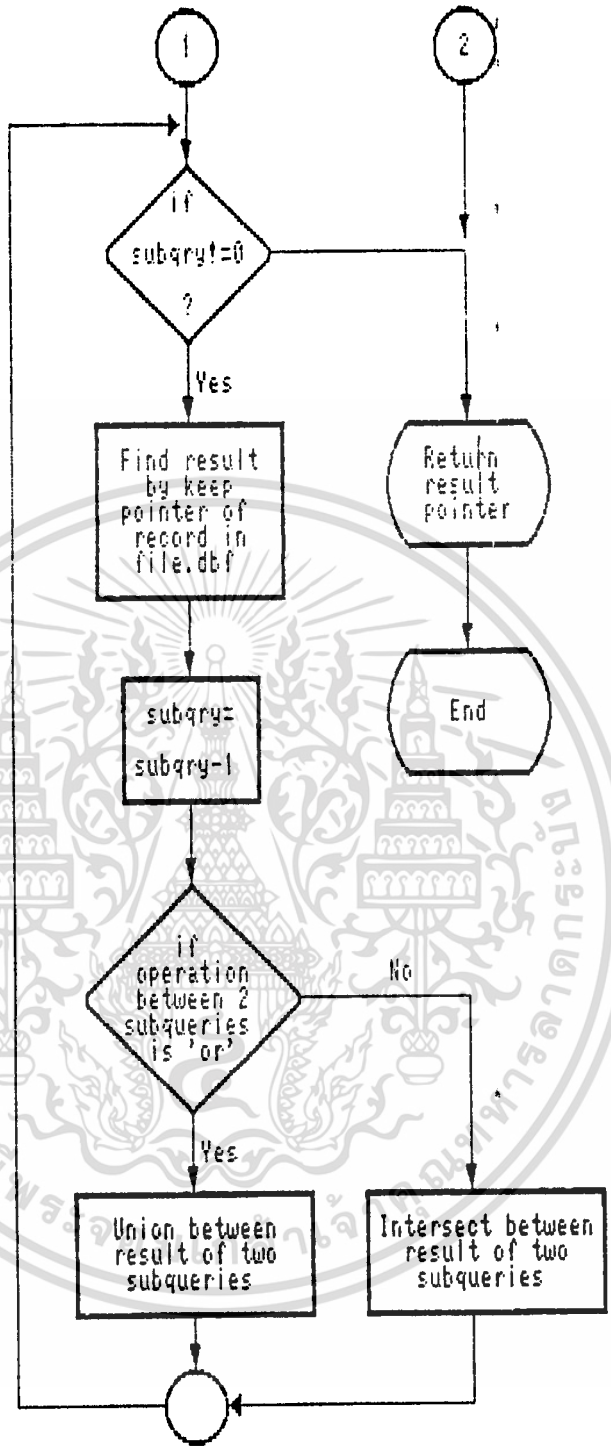




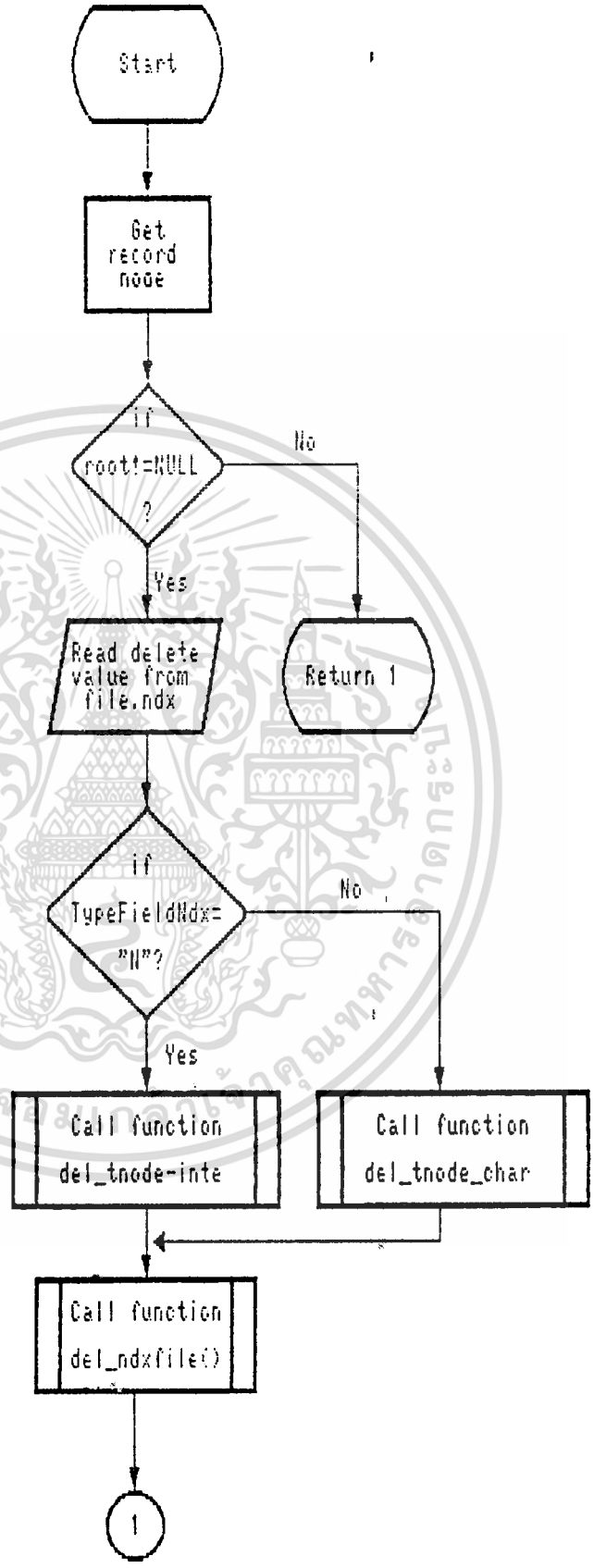


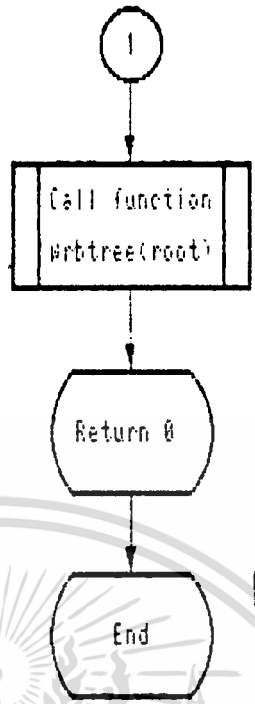
Routine Qry



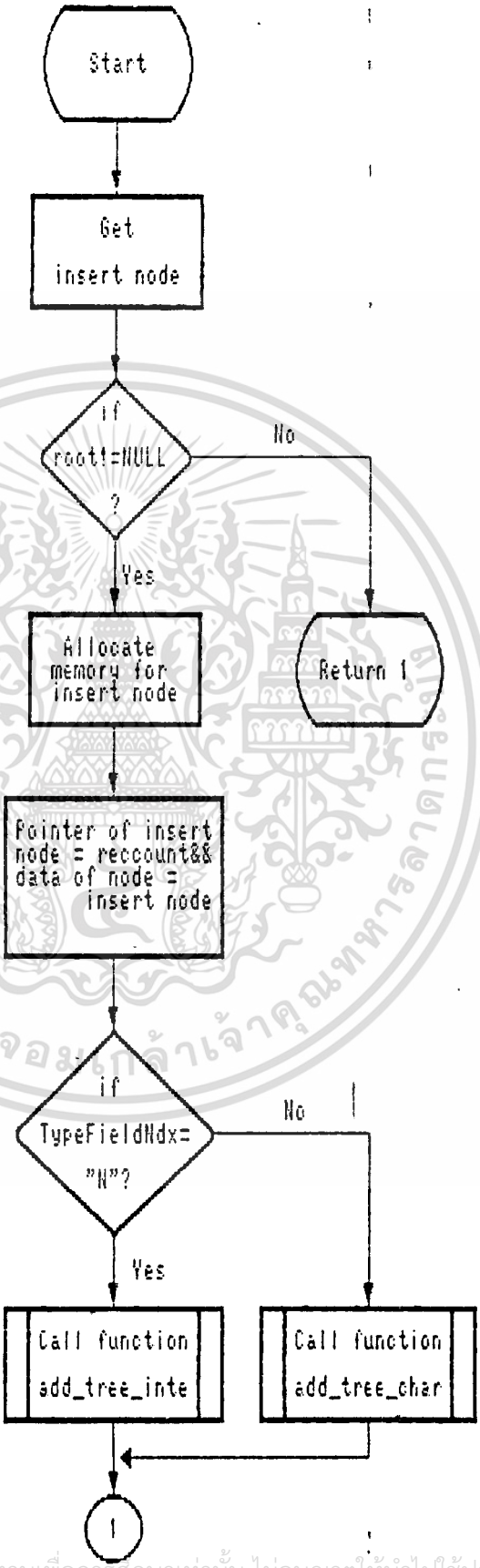


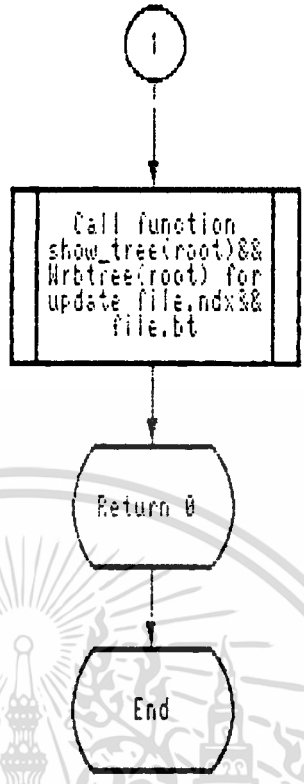
Routine pack_ndx



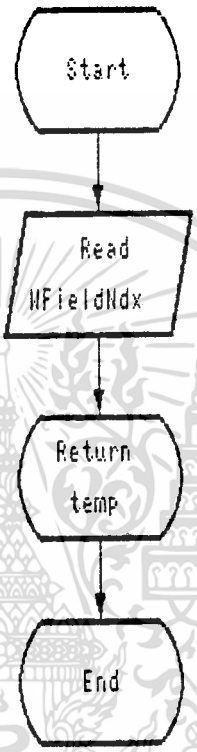


Routine insert

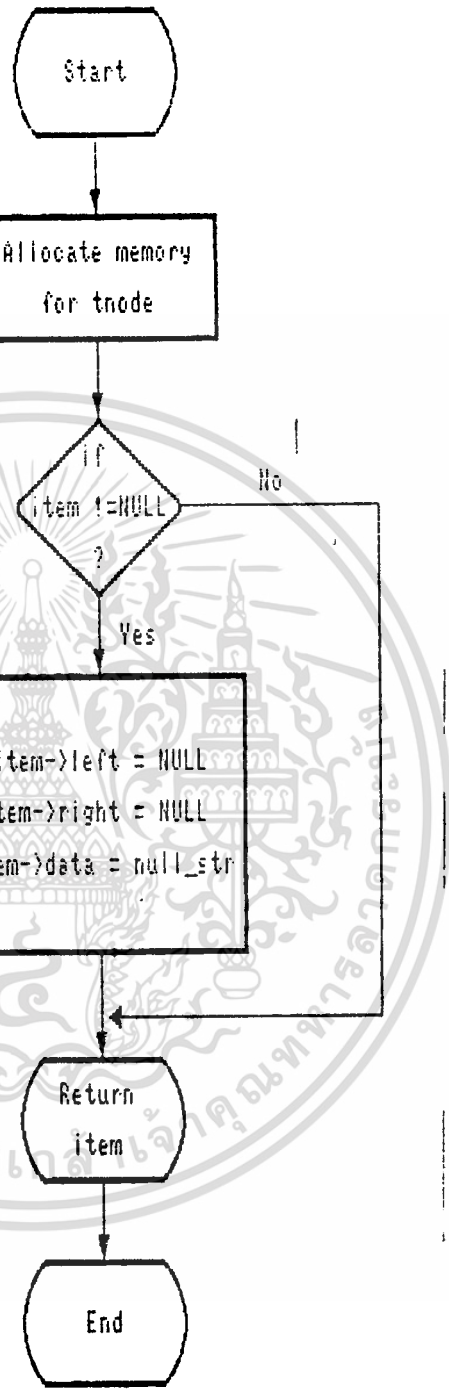




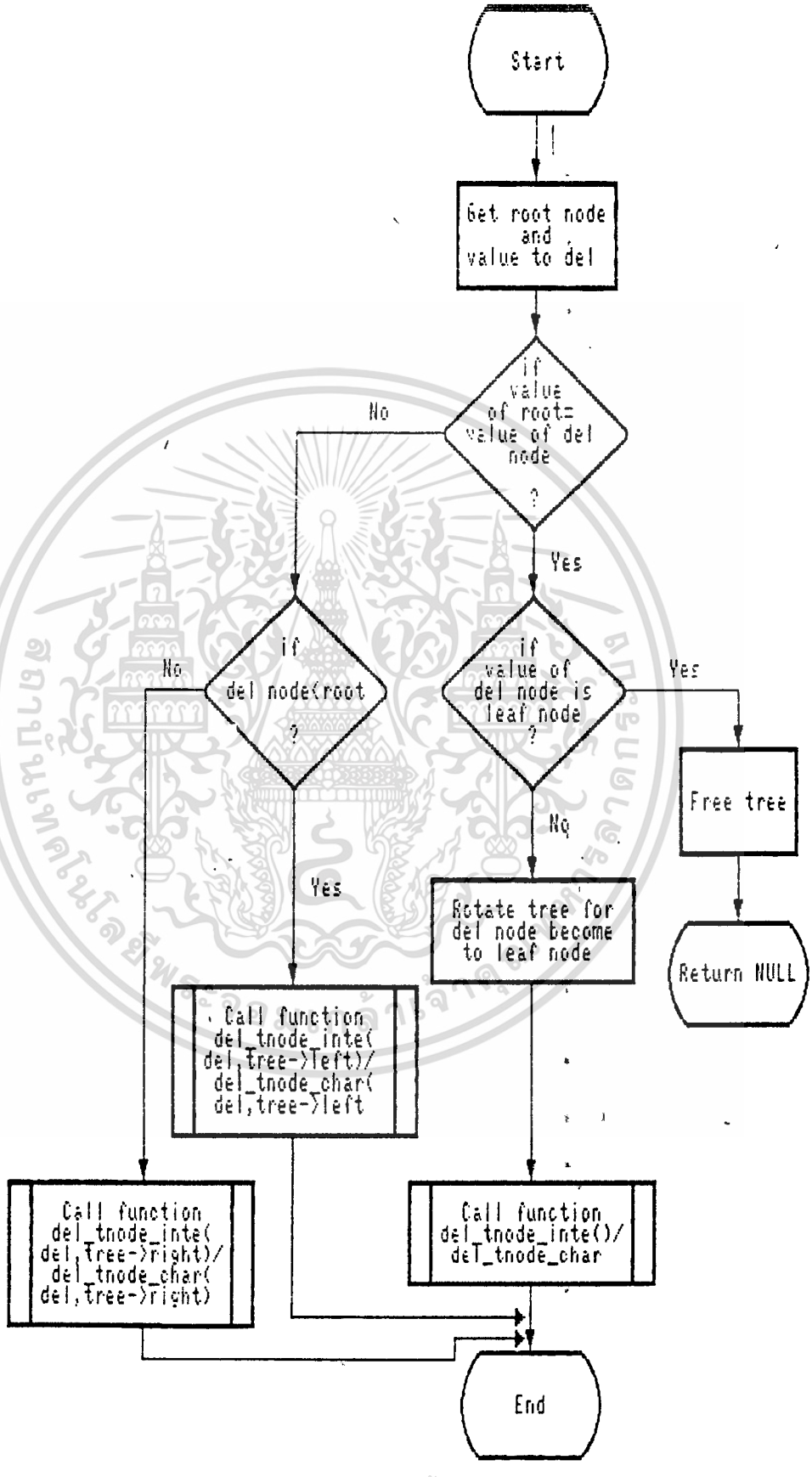
Routine getdat



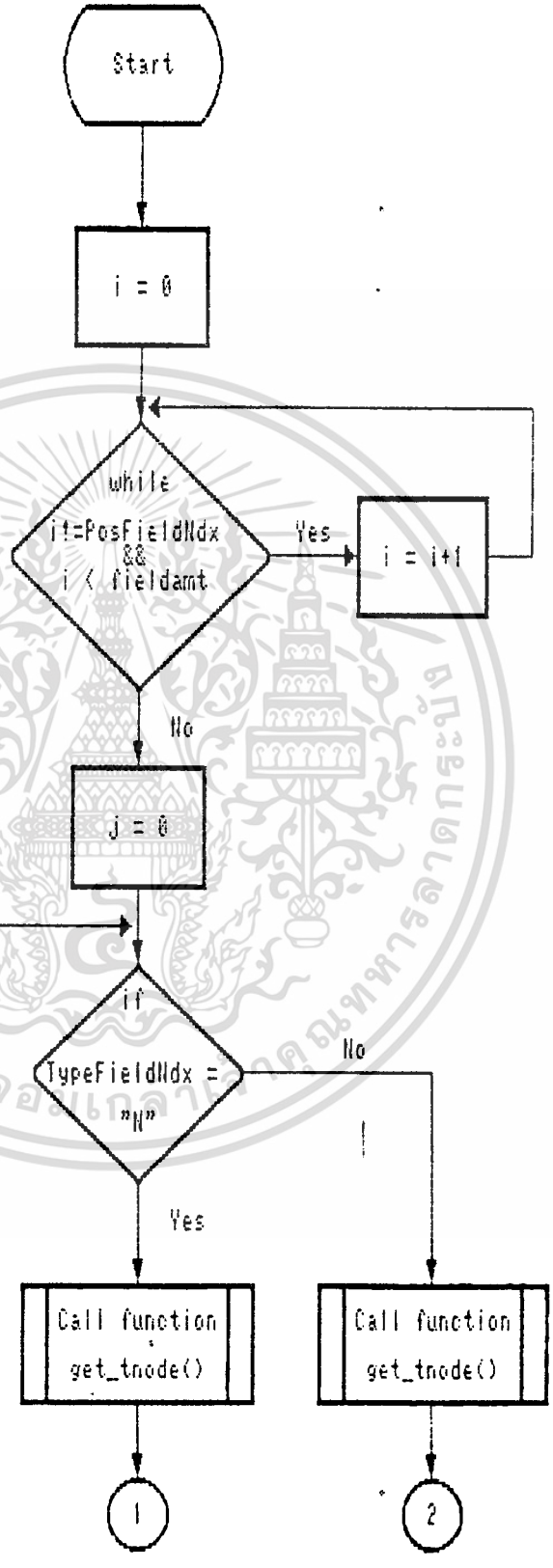
Routine get_tnode

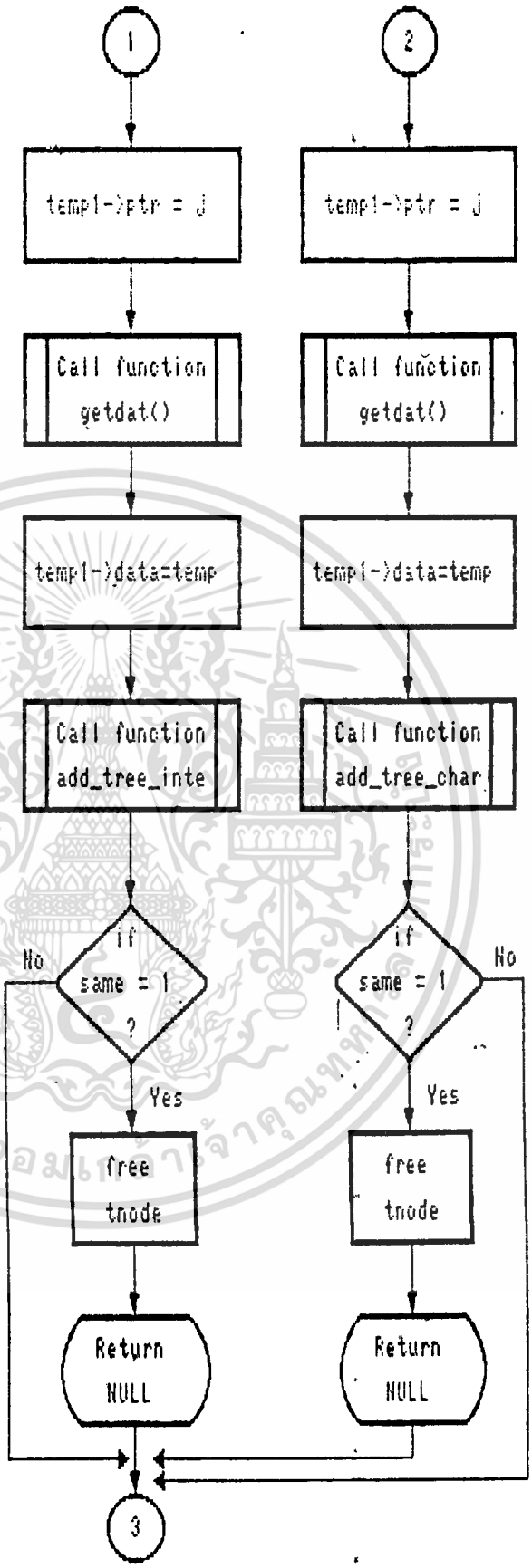


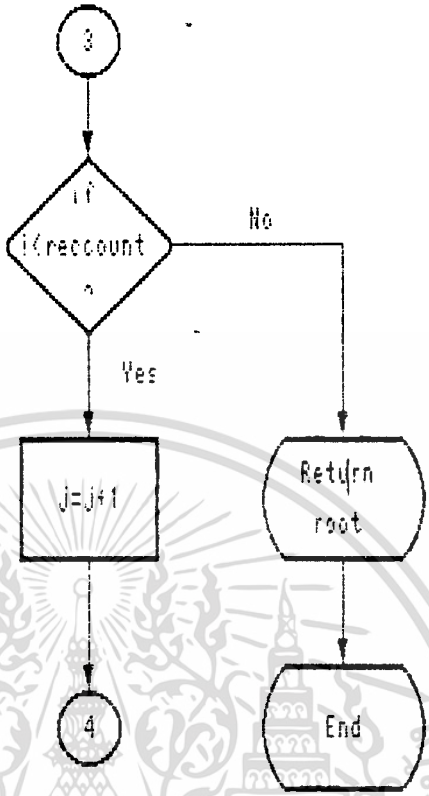
Routine del_ndx_inte/del_ndx_char



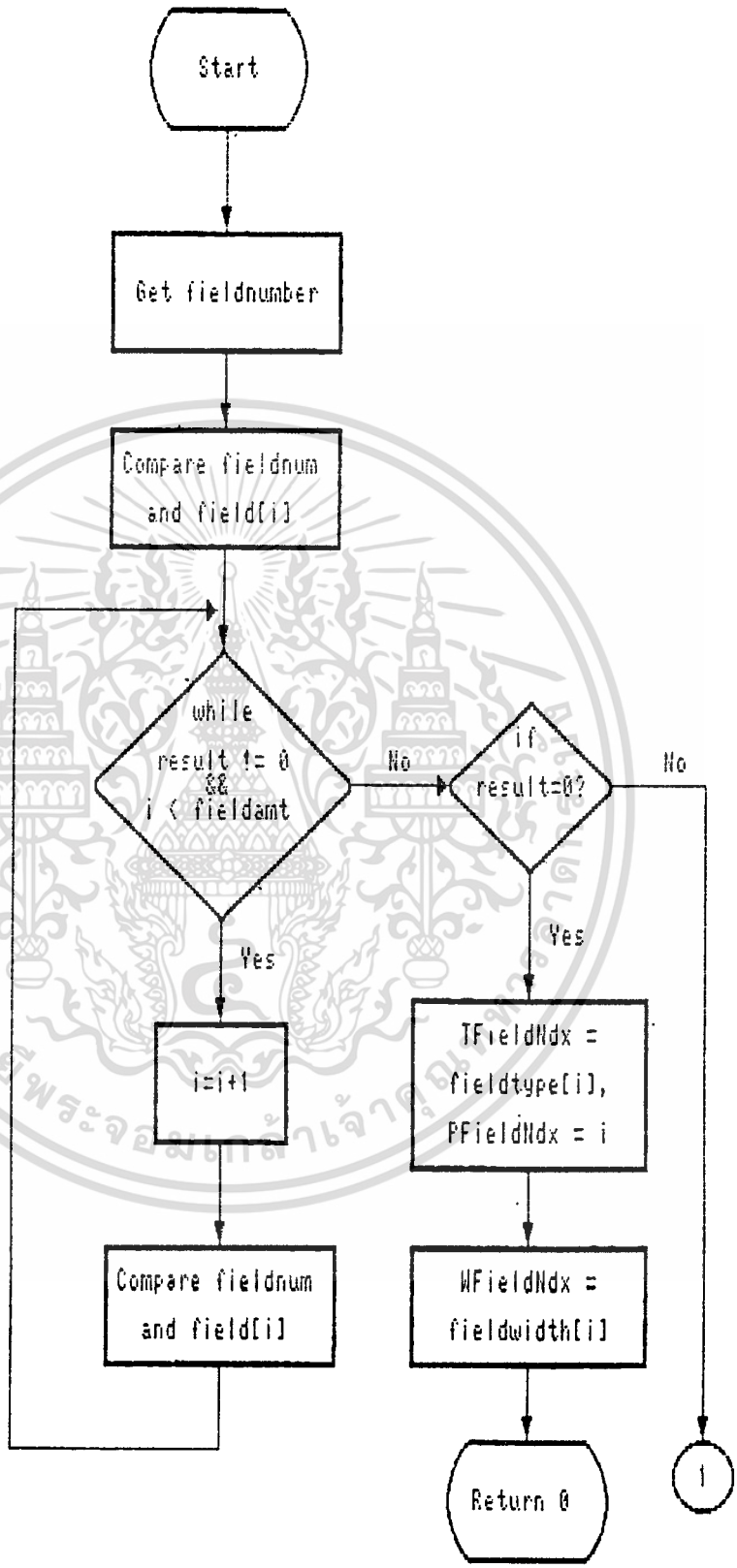
Routine create_tree

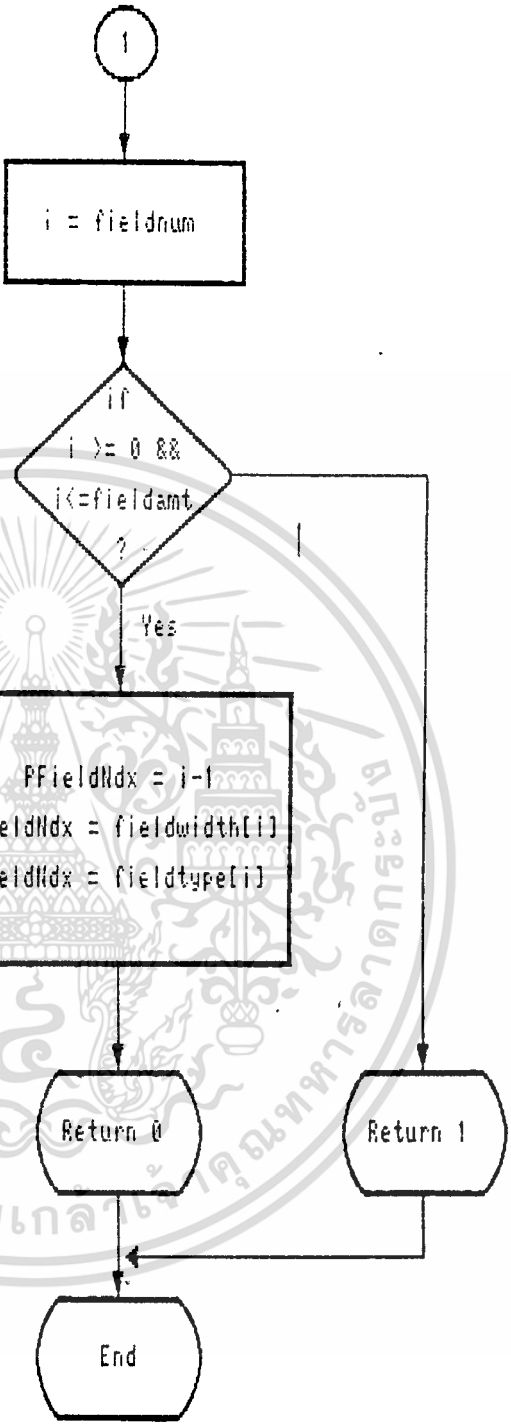




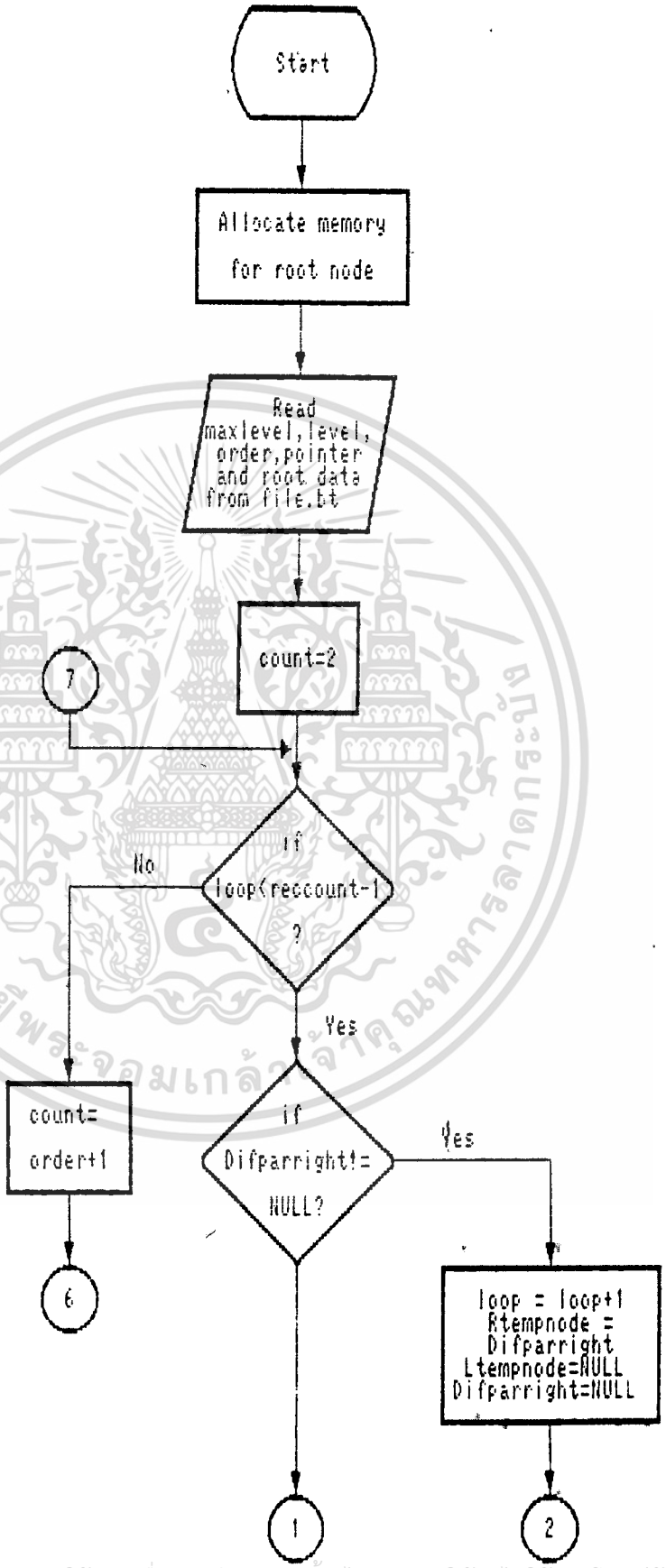


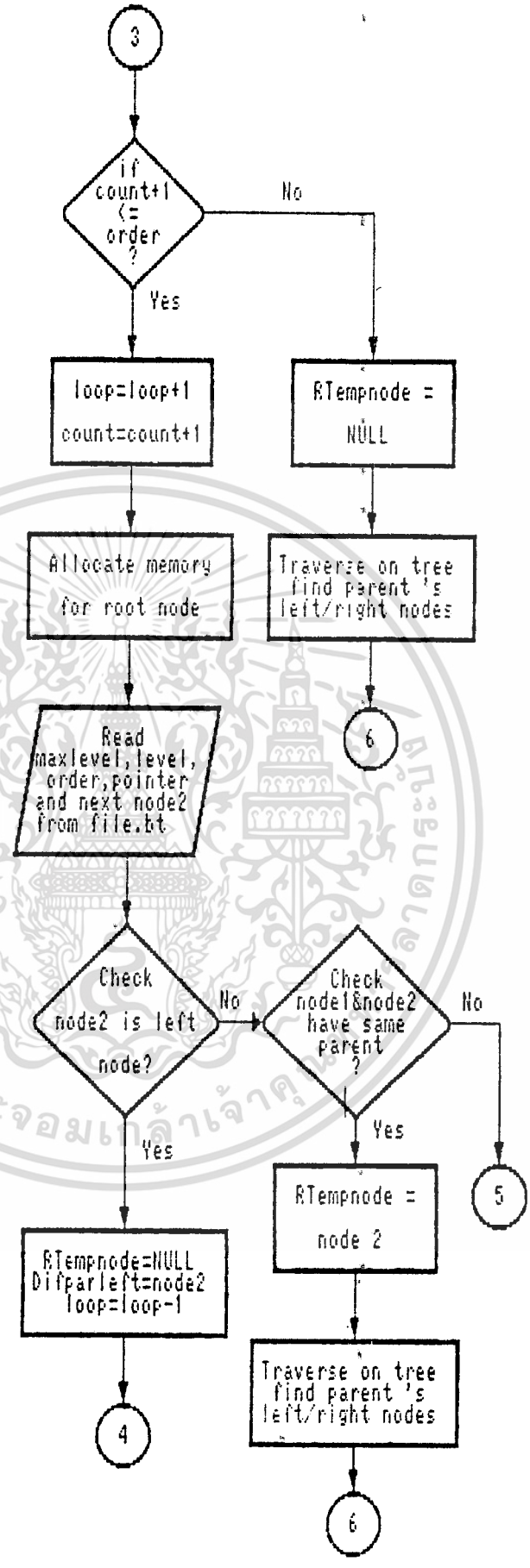
Routine checkfieldindex

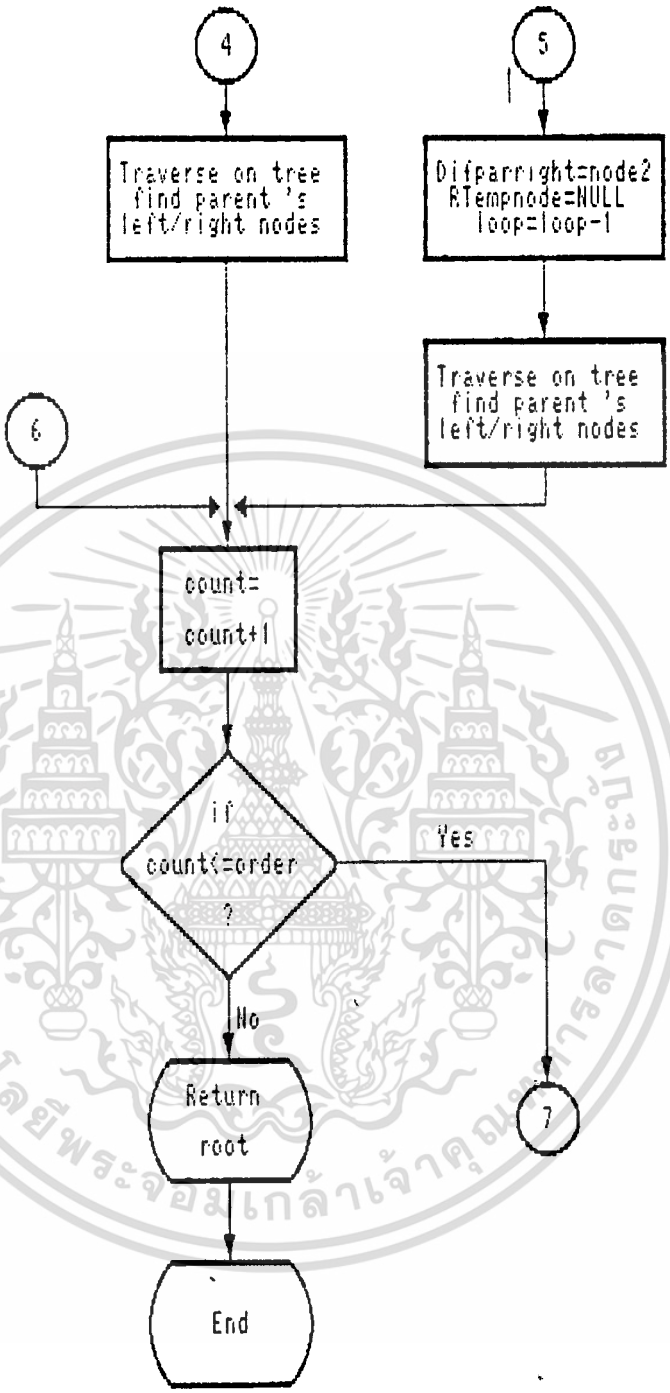




Routine Loadbt







ภาคผนวก ค

Reference Guide

Function rotate_tright
เป็นฟังก์ชั่นที่ทำเกี่ยวกับการหมุน node ใน tree ไปทางขวา

Syntax struct tnode *rotate_tright (struct tnode *temprt)

Prototype index.h

Return Value temprt

Note temprt คือ pointer ของ node ที่จะถูกหมุนไปทางขวา

Function rotate_tleft
เป็นฟังก์ชั่นที่ทำเกี่ยวกับการหมุน node ใน tree ไปทางซ้าย

Syntax struct tnode *rotate_tleft (struct tnode *rtree)

Prototype index.h

Return Value rtree

Note rtree คือ pointer ของ node ที่จะถูกหมุนไปทางซ้าย

Function tree_leaf

เป็นฟังก์ชั่นที่ใช้เช็คว่า node ใน tree เป็น leaf node หรือไม่

Syntax int tree_leaf (struct tnode *item)

Prototype index.h

Return Value Return 0 on success , 1 on error

Note เป็น node ที่จะถูกนำมาเช็คว่าเป็น leaf node หรือไม่

Function createdb

เป็นฟังก์ชั่นที่ใช้สร้าง database file ใหม่

Syntax int createdb (char file[])

Prototype head.h

Return Value Return 0 on success , 1 on error

Note file คือ ชื่อของ database file

Function write_header_file

เป็นฟังก์ชั่นที่ทำหน้าที่การเก็บ header file ลงใน database file

Syntax write_header_file(int h,int total,int fieldamt)

Prototype head.h

Return Value Return 0

Note h คือ file handle ของ database file

total คือ ความกว้างของ record

fieldamt คือ จำนวน field ของ database file

Function write_header_field

เป็นฟังก์ชั่นที่ทำหน้าที่การเก็บ header field ลงใน database file

Syntax write_header_field (int h,char *field)

Prototype head.h

Return Value Return 0

Note h คือ file handle ของ database file

field คือ pointer ของค่าต่าง ๆ ของ field ใน database file

Function read_header

เป็นฟังก์ชั่นที่ทำหน้าที่ในการอ่าน header ของ database file

Syntax int read_header (char filename[])

Prototype head.h

Return Value Return 0 on success , 1 on error

Note filename คือ ชื่อของ database file

Function check_mark

เป็นฟังก์ชั่นที่ทำหน้าที่ในการเช็คว่าเป็น database file หรือไม่

Syntax check_mark ()

Prototype head.h

Return Value Return 0 on success , 1 on error

Function list_struc

เป็นฟังก์ชั่นที่ทำหน้าที่การอ่าน structure ของ database file

Syntax list_struc()

Prototype head.h

Return Value Return 0 on success , 1 on error

Function point_record

เป็นฟังก์ชั่นที่ทำหน้าที่ record number

Syntax point_record (long num)

Prototype head.h

Return Value Return 0

Note num คือ หมายเลขของ record ที่ต้องการจะชี้ไป

Function next_record

เป็นฟังก์ชั่นที่ทำให้ชี้ record number ถัดไป

Syntax next_record()

Prototype head.h

Return Value Return 0

Function pre_record

เป็นฟังก์ชั่นที่ทำให้ชี้ record number ก่อนหน้า record number ปัจจุบัน

Syntax pre_record()

Prototype head.h

Return Value Return 0

Function skip_record

เป็นฟังก์ชั่นที่ใช้ชี้ record number โดยข้าม record number ปัจจุบัน ไปเท่ากับ จำนวน record ที่ skip ไป

Syntax skip_record (long skip)

Prototype head.h

Return Value Return 0

Note skip คือ จำนวน record ที่ต้องการจะ skip ไป

Function first_record

เป็นฟังก์ชั่นที่ใช้ชี้ record number แรก

Syntax first_record()

Prototype head.h

Return Value Return 0

Function last_record

เป็นฟังก์ชั่นที่ใช้ชี้ record number สุดท้าย

Syntax last_record()

Prototype head.h

Return Value Return 0

Function Operate_AND

เป็นการค้นหาค่าที่เหมือนกันใน setqry1 และ setqry2

Syntax long *Operate_AND (long *setqry1,long *setqry2)

Prototype index.h

Return Value Return 0

Note setqry1 คือ pointer ของ long integer ที่จะทำการเปรียบเทียบตัวแรก

setqry2 คือ pointer ของ long integer ที่จะนำมาเปรียบเทียบกับ setqry1

Function check_pointer

Syntax long check_pointer (long recno)

Prototype index.h,head.h

Return Value Return pointer to point data record

Note recno คือ หมายเลขของ record

Function check_ndx

Syntax long cheok_pointer (long num,long *qry)

Prototype index.h,head.h

Return Value Return sort pointer to point data record

Note num คือ จำนวน record

qry คือ pointer ของ long integer ที่เป็นหมายเลข record

Function	list_struc
Syntax	list_struc()
Prototype	head.h
Return Value	Return 0 on success , 1 on error

Function	mem_pack
Syntax	mem_pack (long memrec)
Prototype	head.h
Return Value	Return 0 on success , 1 on error
Note	memrec คือ หมายเลขของ memo record