

순차 파일 프로그래밍

파일 처리

이태용 | 20143254 |컴퓨터소프트웨어공학과| 2018.11.15

내용

[순차파일이란? 2](#_Toc530047904)

[- 코드를 조직하는 가장 조직적인 방법 : 2](#_Toc530047905)

[레코드 저장 기준에 의한 종류 2](#_Toc530047906)

[1. 스트림 파일 : 2](#_Toc530047907)

[2. 입력 순차 파일 : 2](#_Toc530047908)

[3. 키 순차 파일 : 2](#_Toc530047909)

[순차 파일의 갱신 작업 3](#_Toc530047910)

[갱신 알고리즘 3](#_Toc530047911)

[순차 파일의 설계 3](#_Toc530047912)

[설계 시 고려 사항 3](#_Toc530047913)

[1. 레코드 내의 필드 배치는 어떻게 할 것인가? 3](#_Toc530047914)

[2. 키 필드는 어느 것으로 할 것인가? 4](#_Toc530047915)

[3. 적정 블로킹 인수는 얼마이어야 하는가? 4](#_Toc530047916)

[순차 파일의 생성 5](#_Toc530047917)

[프로그램 소스 6](#_Toc530047918)

[실행 결과 10](#_Toc530047919)

# 순차파일이란?

## - 코드를 조직하는 가장 조직적인 방법 :

파일 생성 시 레코드를 연속적으로 저장하기 때문에 레코드를 접근할 때에도 저장순서에 따라 연속적으로 접근하는 것이 효율적이다.

# 레코드 저장 기준에 의한 종류

1. 스트림 파일 :

* 연속적인 판독 연산을 통해 레코드가 파일에 저장되어 있는 순서에 따라 데이터를 접근하는 파일

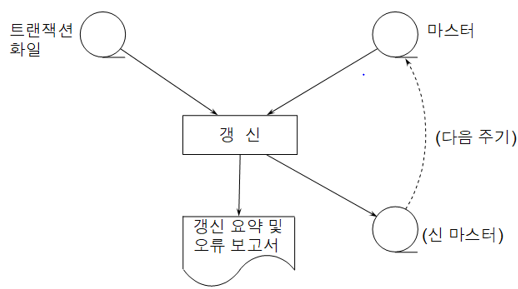
1. 입력 순차 파일 :

* 레코드가 입력되는 순서대로 레코드를 저장하는 파일

1. 키 순차 파일 :

* 레코드의 특정 필드 값 순서에 따라 레코드를 저장하는 파일
* 일반적으로 순차파일이라고 하면 키 순차 파일을 의미하는 경우가 대부분이다.

# 순차 파일의 갱신 작업



## 갱신 알고리즘

-트랜잭션 파일 레코드를 마스터 파일 레코드와 비교

-레코드 키 값이 두 파일에서 일치하면 갱신 코드에 따라 레코드를 수정 또는 삭제

-트랜잭션 레코드 키 값이 마스터 파일의 어떤 레코드의 것과도 일치하지 않으면, 새로 삽입 할 레코드로 판정하고 마스터 파일에 삽입한다.

# 순차 파일의 설계

## 설계 시 고려 사항

### 1. 레코드 내의 필드 배치는 어떻게 할 것인가?

- 데이터의 필드 활용도에 따라 활동 파일과 비활동 파일로 저장한다.

- 각 레코드 길이를 적절한 제어 정보와 함께 저장한다.

- 레코드 제어 정보는 각 레코드 앞에 있는 시스템용 필드에 저장된다.

### 2. 키 필드는 어느 것으로 할 것인가?

- 응용 요건에 따라 선정한다.

- 키 값은 순차 파일에서 레코드 접근 순서를 결정한다.

- 트랜잭션 파일은 마스터 파일과 같은 키 속성을 가진다.

- 보고서 파일은 출력 순서에 따라 레코드가 정렬될 수 있게 결정한다.

### 3. 적정 블로킹 인수는 얼마이어야 하는가?

- 가능한 한 블록을 크게 하는 것이 바람직함

- 버퍼의 크기와 운영 체제가 지원하는 페이지 크기에 의해 제한

# 순차 파일의 생성

파일 생성

- 데이터 저장 장치에 레코드들을 순서대로 입력하여 생성

- 키 순차 파일의 갱신은 트랜잭션 파일을 이용

- 트랜잭션 파일은 데이터 수집, 레코드 형식으로 변환, 레코드 편집, 정렬 과정을 거쳐 생성

파일 편집

- 트랜잭션 파일 생성 과정에서 입력되는 데이터 값에 오류가 있는지 검사하는 과정

순차 파일의 갱신

1.순차 파일에서 검색

2.레코드의 저장 순서에 따라 연속적으로 검색

3.레코드 검색 순서에 따라 레코드 입력 순서를 결정

# 프로그램 소스

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct Record{//파일을 한 줄씩 받아줄 Record 생성

int no; //순차파일의 Key값 no

int amount; //은행 잔고

}Record;

Record mas; //old\_master 레코드 생성

Record tra;

int main() {

FILE \*old\_master, \*transaction, \*new\_master, \*error;

if((old\_master = fopen("master.txt", "r")) == NULL){ //만약 열 파일이 없을 시 발생하는 예외처리

printf("file open error\n");

}

if ((transaction = fopen("transaction.txt", "r")) == NULL){

printf("file open error\n");

}

new\_master = fopen("new\_master.txt", "w"); //프로그램으로 인해 기록될 파일을 쓰기 모드로 열거나 생성함.

error = fopen("error.txt", "w");

int m\_file\_pointer; //마스터 파일의 라인을 저장할 변수

int t\_file\_pointer;

char error\_type;

m\_file\_pointer = fscanf(old\_master,"%d %d", &mas.no, &mas.amount); //마스터 파일을 한 라인씩 읽어들임.

t\_file\_pointer = fscanf(transaction,"%d %d", &tra.no, &tra.amount);

void get\_master\_record(){ //마스터 파일의 키 값이 EOF가 아닐 시 다음 라인을 읽어들이는 함수.

if (m\_file\_pointer == EOF){

mas.no = EOF; //파일의 끝을 포인터가 가르킬 시 키 값을 EOF로 지정

}else {

printf("%d %d\n",mas.no,mas.amount); //프로그램이 정상적으로 돌고 있는지 로그 기록

m\_file\_pointer = fscanf(old\_master,"%d %d", &mas.no, &mas.amount);

}

}

void get\_transaction\_record(){

if (t\_file\_pointer == EOF){

tra.no = EOF;

}else

t\_file\_pointer = fscanf(transaction,"%d %d", &tra.no, &tra.amount);

}

while(1){

if(mas.no < tra.no){ //transaction 파일의 키 값이 old\_master의 키 값보다 클 시.

fprintf(new\_master, "%d %d\n", mas.no, mas.amount); // 처리할 transaction이 없으므로

get\_master\_record(); //new\_master에 레코드 그대로 복사.

continue; //키 값 가산으로 인한 오류를 막기 위해 다음 반복문으로 이동함.

}

if (mas.no == tra.no) { //처리할 transaction이 있을 때.

if ((mas.amount + tra.amount) < 0){ //은행잔고가 부족하다면

error\_type = 'L'; //L에러를 출력.

fprintf(error,"%c %d %d\n",error\_type, tra.no, tra.amount); //에러파일에 에러 기록

fprintf(new\_master,"%d %d\n",mas.no, mas.amount); //new\_master에 old\_master 레코드

get\_master\_record(); //그대로 복사. 다음 old\_master 레코드 읽어옴

get\_transaction\_record(); //다음 transaction 레코드 읽어옴.

continue;

}else{ //은행잔고가 부족하지 않을 경우 실행되는 구문.

mas.amount = mas.amount + tra.amount; //old\_master 잔고에 transaction 금액을 더하고

fprintf(new\_master,"%d %d\n",mas.no, mas.amount); //new\_master에 기록

get\_master\_record();

get\_transaction\_record();

continue;

}

}

//old\_master의 키 값이 transaction의 키 값을 지나쳤을 때.

if(mas.no > tra.no && tra.no != EOF){ //tra.no가 EOF일 때 마지막으로 한 번 실행되는 오류 제거

error\_type = 'U'; //old\_master 파일에는 없는 transaction의 Key값이므로

fprintf(error,"%c %d %d\n",error\_type, tra.no, tra.amount);//에러메세지 출력.

fprintf(new\_master,"%d %d\n",mas.no, mas.amount); //다음 old\_master와 transaction 레코드를 불러옴.

get\_master\_record();

get\_transaction\_record();

continue;

}if (mas.no == EOF && tra.no == EOF) break; //파일 포인터들이 각 파일의 끝에 도달했을 시

} //반복문 탈출

fclose(old\_master);

fclose(transaction);

fclose(error);

fclose(new\_master);

return 0;

}

# 실행 결과

