File Processing

Programming Project #2

담당 교수 : 박석

학번 : 20161566

이름 : 권형준

목차

0. 프로젝트 필요하다고 생각되는 가정

① 개발 환경 가정

개발 환경은 Visual Studio 2019를 사용하였다.

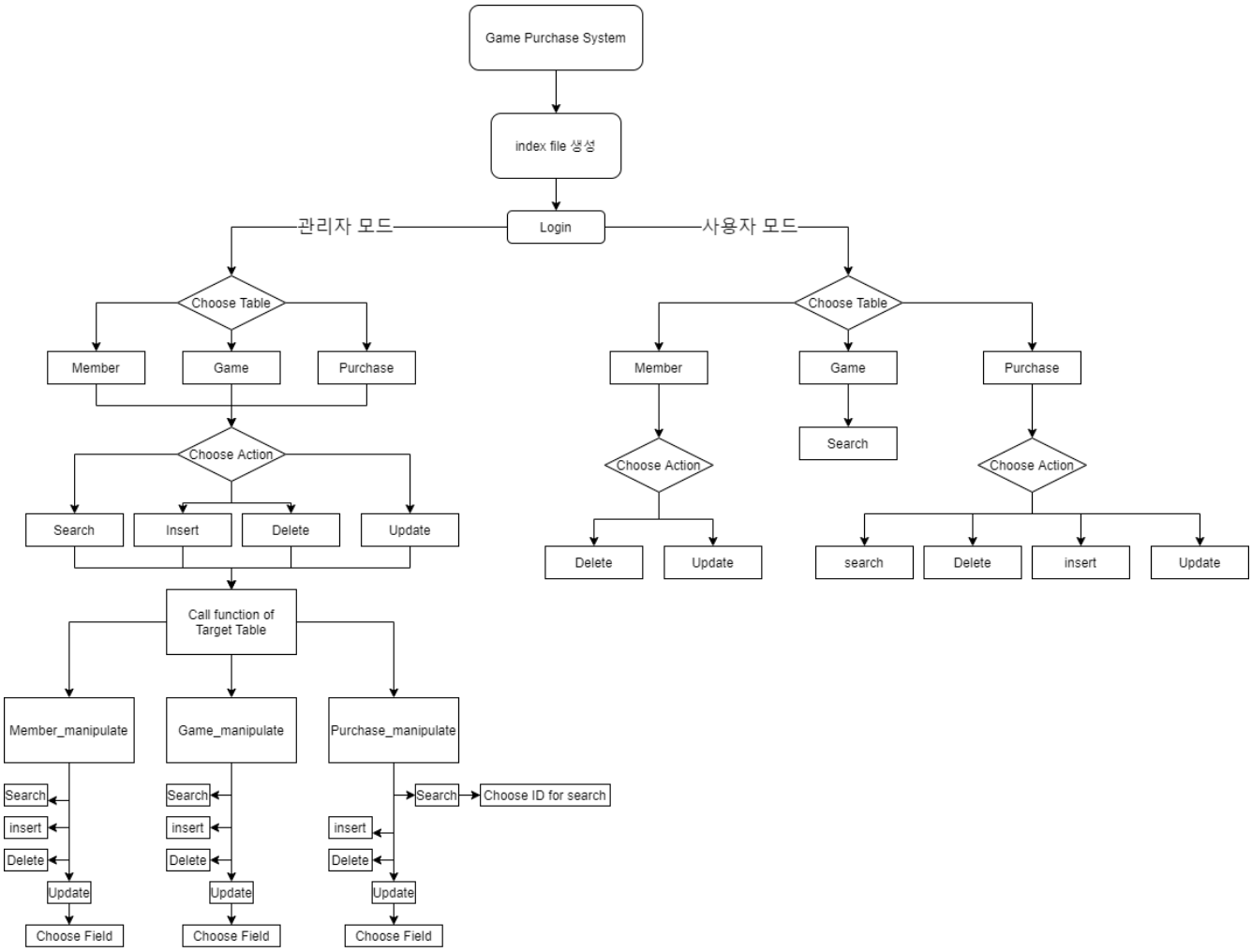
② Index파일에 대한 가정

Text Index는 처음에 생성할 때 최대 크기를 정하게 된다. 이 크기에 맞춰서 index file에 저장될 수 있는 key의 최대 값도 저장이 된다. 이번 프로젝트에서 생성하는 game과 member의 record개수는 1000개임으로 처음부터 추가를 할 수 있기 위하여 넉넉하게 2000을 최대 크기로 설정하였다.

③ Index Key값에 대한 가정

Member의 경우 Key값인 ID가 string으로 존재하기 때문에 정확한 크기를 정하기 힘들었다. Index file은 fixed filed buffer을 사용함으로 최대 길이를 정해줘야 되는데, 이를 20으로 설정하였다.

1. GamePurchaseSystem 흐름도



2. 메뉴를 사용하는 방법

처음 프로그램을 실행하면 프로젝트1에 이어서 test, show등의 함수를 고를 수 있고 7번을 누르면 Game Purchase System이 실행된다. 이후 login할 수 있는 화면이 나오는데, 관리자 모드로 사용하고 싶다면

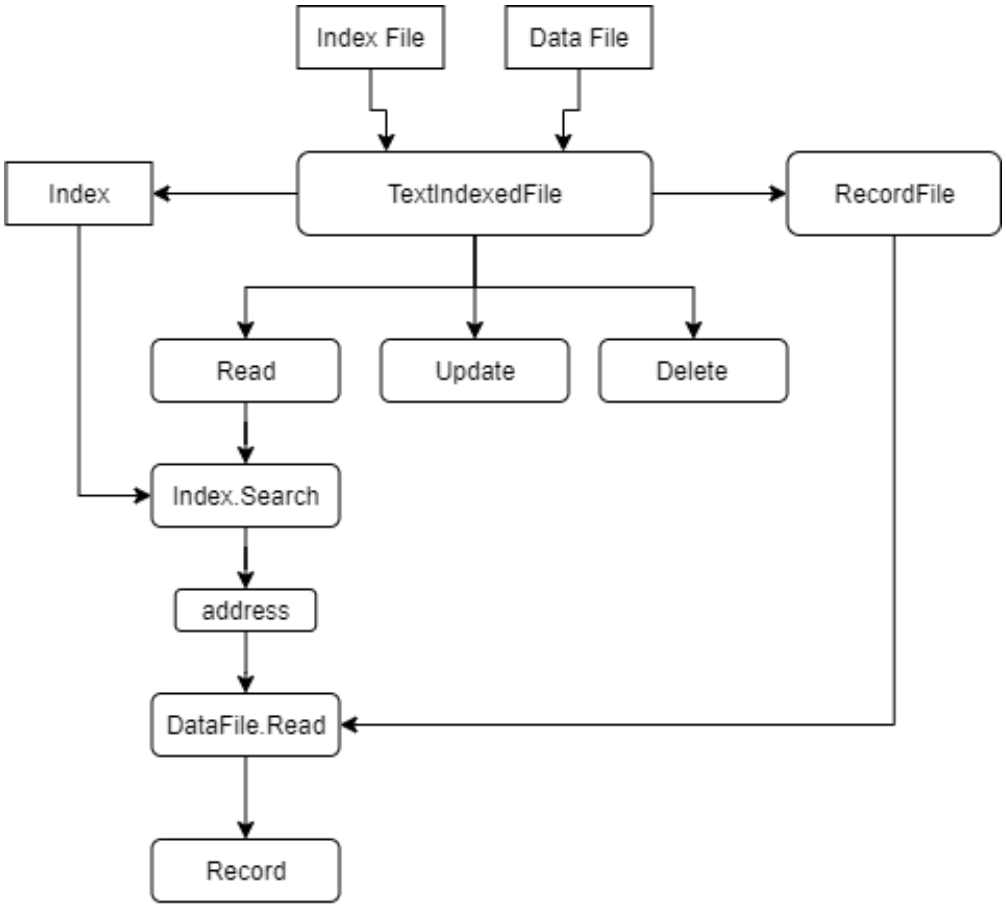
만약 그 외의 사용자 모드로 사용하고 싶다면, 사용하고 싶은 사용자의 ID와 password를 치면 된다. 프로젝트 명세서에 나온 것처럼 TestUser, T1234를 추가하였다. 만약 관리자 모드를 사용한 경우 프로젝트1의 Game Purchase System과 같은 화면이 나온다. Member, game, purchase중 하나를 선택하고 하고 싶은 행동을 선택한 이후 실행한다. 만약 사용자 모드를 사용하게 되는 경우 3가지 중 하나를 고를 수 있게 된다. 1. 자신의 member data를 삭제하거나 수정한다. 2. 모든 게임 중 검색을 실시한다. 3. 자신이 purchase한 목록에서 검색, 삭제, 추가, 수정을 한다. 각각의 메뉴 단계에서 말도 안 되는 입력을 받을 경우 여러 번의 반복문이 돌아서 오류가 발생할 수 있다.

3. 로그인 구현 설명

프로젝트 1에서 구현한 "fileOfMember.dat"을 사용하였다. 새로운 필드 Level을 부여하여 관리자와 사용자를 구분하였다. 이번 프로젝트에서 구현한 Index 검색을 통하여 들어온 ID에 대한 record에 접근한다. 접근한 record가 존재한다면 입력 받은 password와 비교하여 같다면 login을 하고 다르다면 오류를 출력한다. 이후 해당 레코드가 사용자 모드로 들어왔는지, 관리자 모드로 들어왔는지 확인한다.

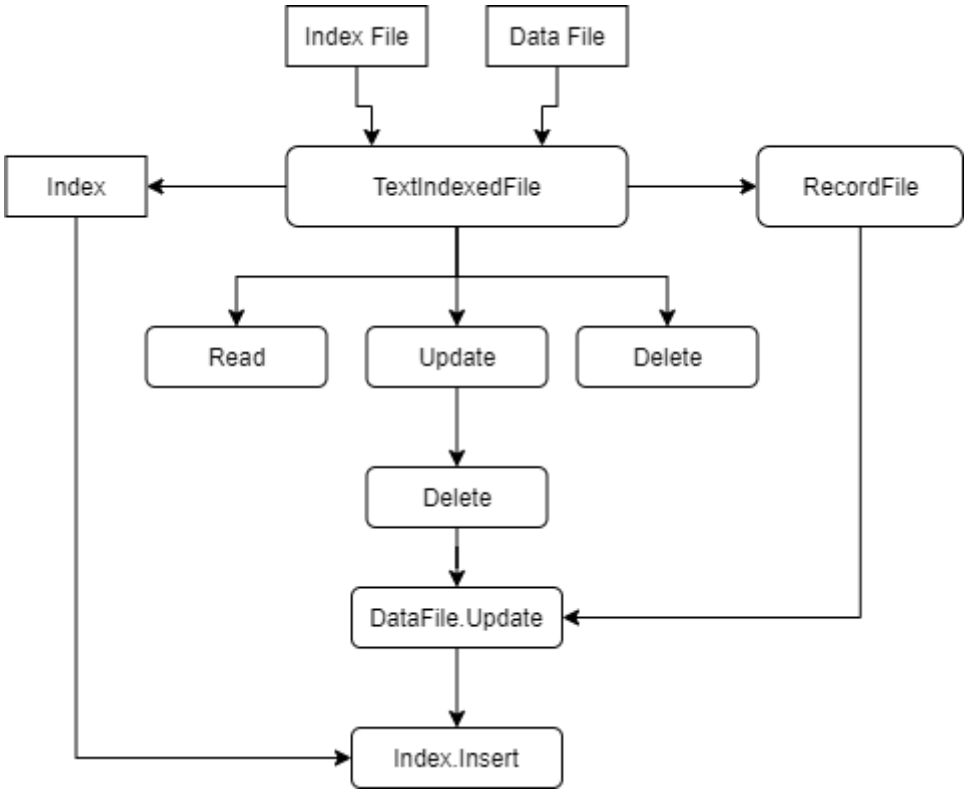
4. Index 사용한 검색, 삽입, 수정, 삭제 흐름

① 검색



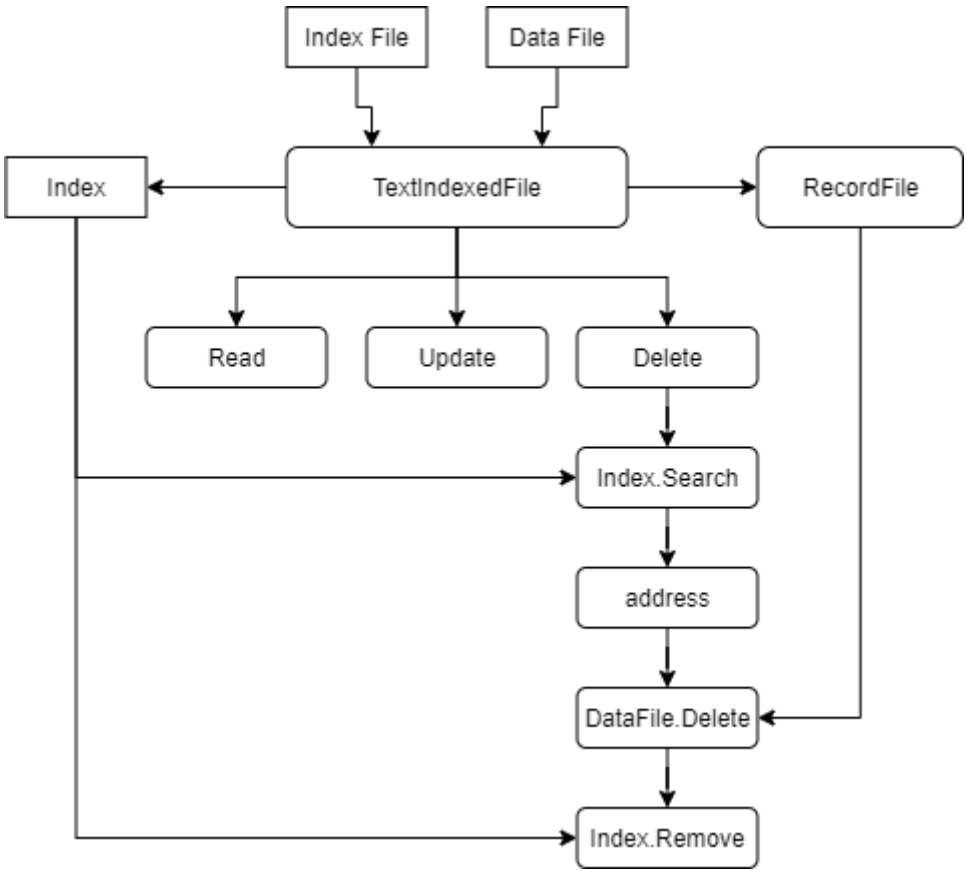
Textindexedfile에서 open을 실행하면 입력으로 들어온 이름의 index파일과 data파일을 다 open하게 된다. 이후 open한 파일들의 내용을 바탕으로 index와 recordfile을 생성한다. Read(검색)을 하게 될 시, textindex class에 구현되어 있는 search 메소드를 사용하여 index내에 해당 key값에 대응하는 주소를 읽어온다. 이때, 해당 key값에 해당하는 index가 없는 경우 -1을 반환한다. 읽어온 주소가 양수라면 TextIndexedFile 안의 RecordFile에서 프로젝트1에 구현한 Read를 호출한다. 주소에 대응하는 record파일을 읽어서 반환하게 된다. 이 경우에도 실패하면 -1을 반환하게 된다. 하지만 Index파일을 GamePurchase system을 실행할 때 생성하고 data 파일에 대한 접근은 모두 index와 동시에 발생하기 때문에 여기서 실패하는 경우는 없을 것이다.

② 삽입, 수정



삽입과 수정은 동시에 구현하였다. 삽입과 수정의 유일한 차이는, 수정의 경우 이전에 존재하던 레코드를 삭제하고 새로 수정한 레코드를 삽입하는 구조를 가진다는 것이다. 즉, 수정은 삭제 후 삽입처럼 구현하였다. 처음에 들어온 key에 해당하는 index와 data file의 레코드를 삭제하였다. 이때 만약 처음에 들어온 key에 해당하는 index와 data file의 레코드가 존재하지 않는다면, 그것은 수정이 아닌 삽입이다. 이후 삽입하고자 하는 새로운 레코드를 프로젝트 1에서 구현한 update함수를 사용하여 진행하였다. 프로젝트1에서 삽입의 경우 가용리스트를 처음부터 탐색하는 (first-best-fit)을 사용하여 새로운 레코드를 삽입하였는데, 이 과정을 TextIndexFile에 있는 Update 메소드에 추가하여 그대로 사용하였다. 이 과정을 거쳤으면 data파일은 새로 record가 추가되었지만 Index에는 새로운 record에 해당하는 index가 들어오지 않았다. 따라서 index.insert를 사용하여 새로운 주소를 key와 함께 index에 추가하여 준다.

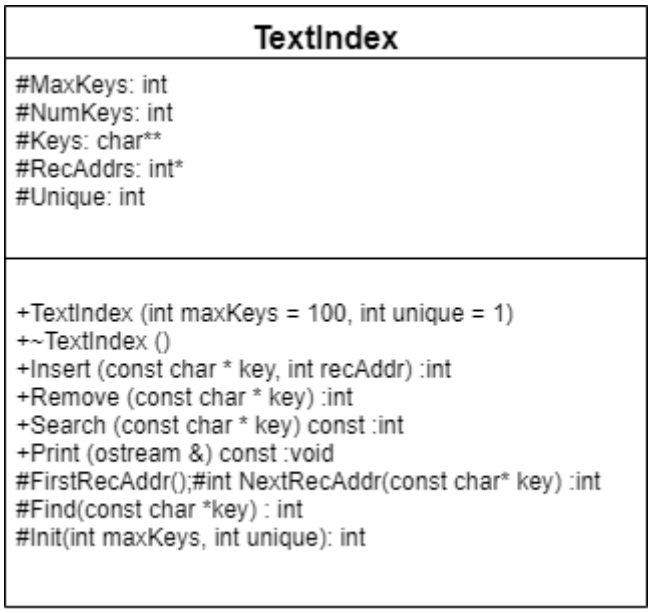
③ 삭제



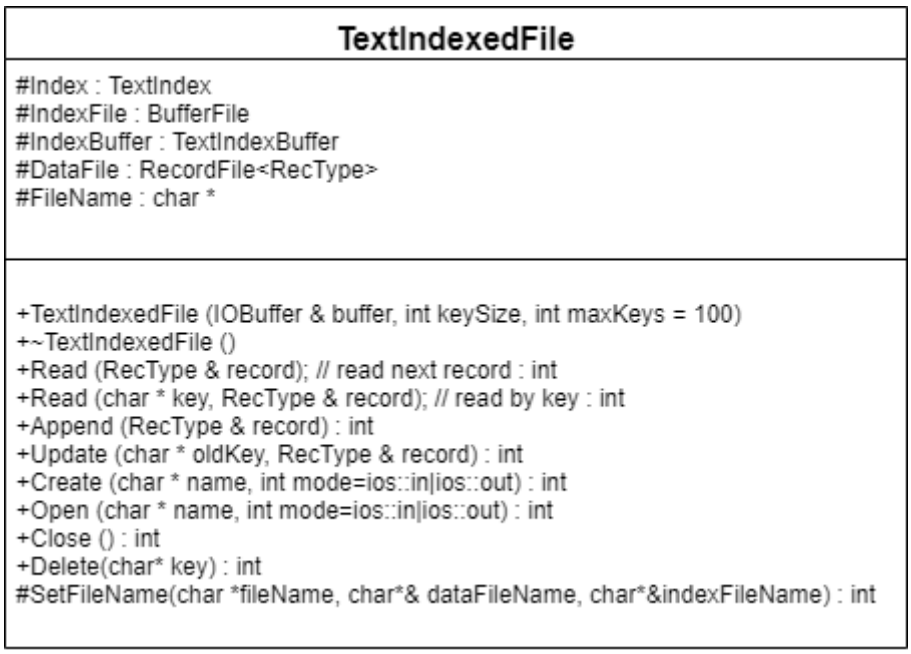
삭제하고자 하는 key에 해당하는 index를 검색한다. 만약 없다면 -1을 반환한다. 존재한다면 record file에서 구현한 delete 메소드를 사용하여 data file의 record를 삭제한다. 이 과정에서 가용리스트가 생성되고, data file header에 가용리스트의 header가 생성된다. 이제 삭제한 record에 대응하는 index 파일의 key와 주소도 삭제해주기 위해 Index.Remove를 호출한다.

5. (새로)구현한 클래스의 다이어그램

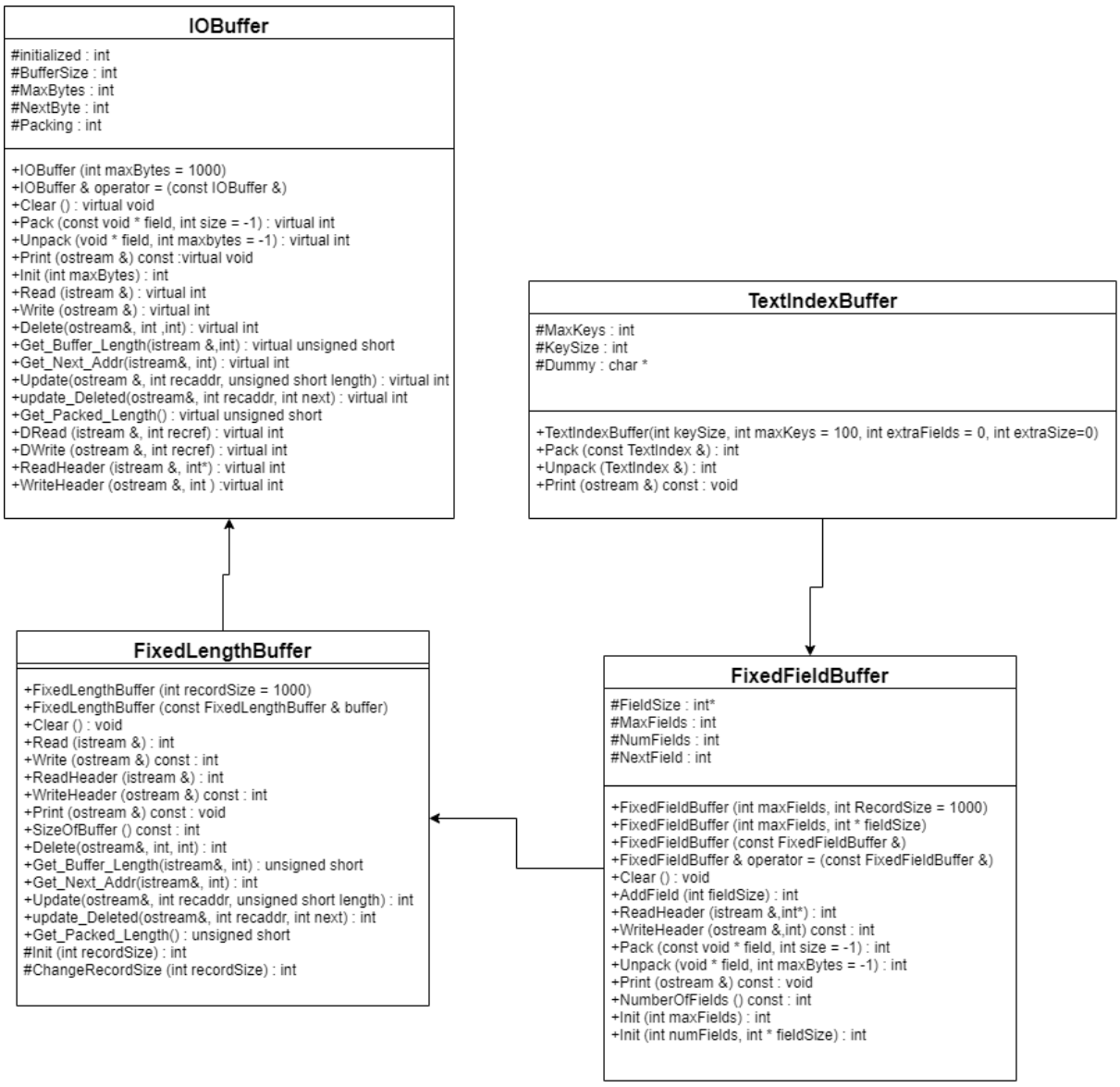
① TextIndex



② TextIndexFile



③ TextIndexBuffer



6. 각 파일에 대한 기능 설명

➔ Textind.h/textind.cpp

Text index 클래스와 관련된 method를 담고 있는 파일이다. Text index는 어떠한 data파일의 접근, 또는 정렬을 더 편리하게 하게 해준다. 담고 있는 정보는 index의 최대 크기, 현재 크기, key와 주소를 담고 있는 배열이 있고, method는 새로운 index를 넣는 Insert, 있는 index를 제거하는 Remove, 찾고자 하는 key에 해당하는 index가 있는지를 위한 Search 등등이 있다.

➔ Indfile.h

프로젝트1에서는 data file과 buffer을 함께 담고 있는 buffer file과, 특정한 record를 지정할 수 있는 record file 등의 class를 사용하였다. 이제는 한단계 더 확장하여 index까지 포함하고 다룰 수 있는 text index file을 정의하는 파일이 바로 "indfile.h"이다. 클래스 내부에는 index, buffer file, text index buffer, record file등의 변수들이 다 포함되어 있다. Text indexed file의 역할은 index 파일과 data 파일을 열어서 같이 검색, 수정, 삭제, 삽입을 할 수 있게 한다. 이전에 구현한 data파일의 delete, read등의 method를 그대로 사용할 수 있는 장점이 있다.

➔ Tindbuff.h/tindbuff.cpp

Text index 클래스로 생성한 index를 파일에 기록하기 위해서는 특정 record형태와 buffer형태를 갖추어서 기록해야 된다. 이번 프로젝트에서 사용하는 index는 fixed length, fixed filed임으로, 이를 상속한 새로운 class textindexbuffer을 사용하여 더 편리하게 pack, unpack을 하게 해준다. 만약 이 class가 없다면, 모든 index를 각각의 buffer에 넣어서 하나 하나씩 fixed filed record처럼 기록해야 되는 불편함이 있다. 간단하게 말해서 Pack, unpack method를 overload하기 위한 목적이 가장 크다. 실제로 pack, unpack부분의 코드를 살펴보면 다음과 같이 반복문을 통하여 fixedfiled의 pack연산을 수행하는 것을 알 수 있다.

```
for (int i = 0; i < index.NumKeys; i++)
{// note only pack the actual keys and recaddrs
    result = result && FixedFieldBuffer::Pack (index.Keys[i]);
    result = result && FixedFieldBuffer::Pack (&index.RecAddrs[i]);
}
```

➔ Make_index_file(main.cpp)

처음 game purchase system을 수행 시작하였을 때, 존재하는 data file을 기반으로 index파일을 생성하는 함수이다. 이미 index파일이 존재하더라도 새로 생성한다. 다른 함수의 호출로 인하여 data file은 수정이 되었는데 index파일은 수정이 안될 수 있기 때문이다. (예: membertest, gametest등등) 따라서 참조 무결성을 위하여 매번 새로운 index파일을 data파일 기준으로 생성한다. 이때 index파일을 생성할 때, indexfiledbuffer class를 사용하지 않고 bufferfile class를 사용한다.

7. 7장 연습문제

① 13번

➔ Int FirstRecAddr()

Index는 처음 insert할 때, create할 때, 이미 정렬이 된 상태로 key와 주소를 저장하고 있다. 따라서 가장 작은 키에 대한 참조는 Recaddr[0]을 반환하면 되는 것이다.

➔ Int NextRecAddr()

문제의 조건이, "키 순서로 인덱스를 통한 반복을 지원"이었기 때문에, 다른 key의 값을 필요로 하지 않으면서 index파일을 순회할 수 있어야 한다. 따라서 "int curr"이라는 변수를 생성하여 이전까지 순회한 index를 저장한다. 처음 Init을 할 때, 0으로 초기화한다. NextRecAddr()을 호출할 때마다, 이 값을 증가시키고 해당하는 RecAddrs을 반환한다. 만약 NumKeys를 초과할 경우 -1을 반환한다.

② 16번

이진 탐색을 구현하기 위하여 Search메소드를 수정하려고 했으나, Search에서 결국 Find메소드를 호출하여 사용함으로, Find를 수정하였다. 기본적으로 이진 탐색을 하기 위해서는 어떠한 배열, list는 정렬이 된 상태여야 된다. Index는 항상 strcmp함수를 기준으로 정렬이 되어있기 때문에 바로 사용할 수 있었다. First, last변수를 선언하여 first<last인 동안 두 값의 중간 지점인 middle을 설정하여 key[middle]과 찾고자 하는 key를 strcmp로 비교하여 first=middle, 또는 last=middle로 바꿔 가면서 점점 범위를 줄여간다. 만약 성공하면 해당하는 index를 반환하고, 실패하면 -1을 반환한다. 그러면 search로 가서 Recaddrs[index]를 반환하여 찾고자 하는 주소를 반환하게 된다.

③ 17번

TextIndexedFile의 Delete method는 구현한 것을 위에서 흐름도와 설명을 적었다. 여기서 다시 정리하면 다음과 같다. TextIndexedFile 클래스 안에는 index, datafile등이 열려 있는 상태이다. 따라서 바로 index에서 search메소드를 호출하여 원하는 주소를 찾고, datafile에 구현되어 있는 delete를 사용하여 해당하는 주소의 data record를 삭제하고, 마지막으로 대응하는 index항목도 삭제해준다. Texed indexed file을 닫으면 자동으로 index파일도 update가 된다. Data file에 구현되어 있는 delete는 프로젝트1에서 구현한 가변길이, delimed buffer 레코드이다.

④ 18번

Textindexedfile에 구현되어 있는 method는 현재 생성되어 있는 index파일과 data파일을 열어서 조작하거나, 둘 다 새로 생성하는 method밖에 없다. 문제에서 요구하는 것은 data파일은 open하고 index파일은 생성하는 것이다. 따라서 새로운 method "create_from_data"를 추가하였다. 기존에 존재하던 create에서 Datafile.create 대신 open을 실시하였고. 실습과, make_index_file에서 실시한 것처럼 index파일을 bufferfile클래스를 사용하여 생성하였다. Open한 data파일에서 차례대로 읽어와서 textindexbuffer에 저장하고 마지막에 pack을 한 이후 write해주었다. 이후에 TextIdexedfile의 맴버들을 초기화하기 위하여 열려있는 data와 index를 닫고 textindexedfile의 open 메소드를 호출하였다. Open method에서는 알아서 모든 index, bufferfile등을 설정한다.