2019-2

CSE4095-1

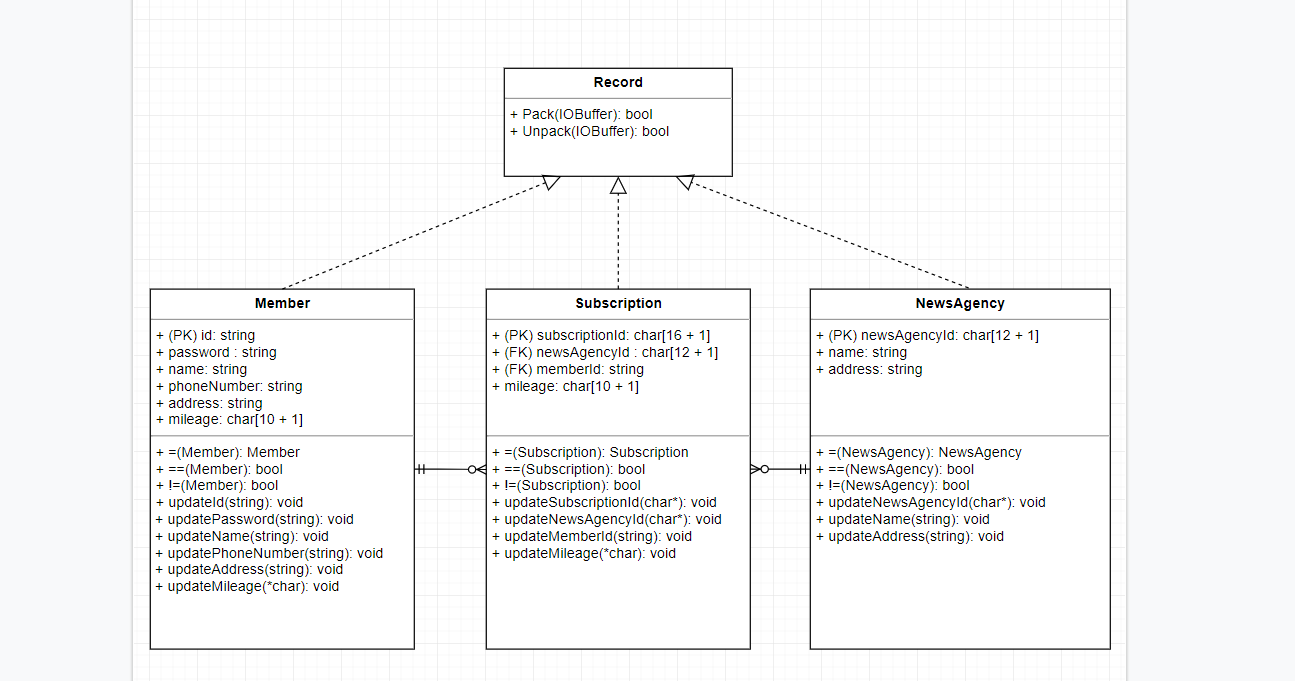
**File Processing**

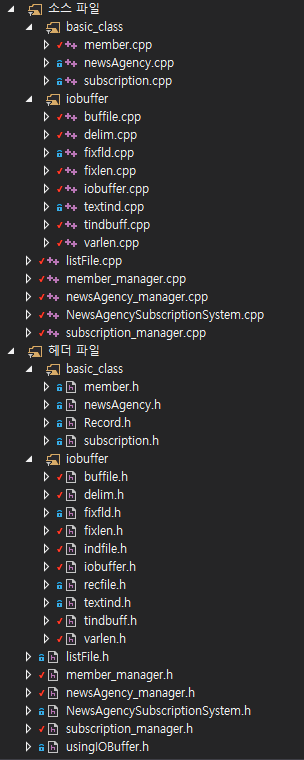
**Programming Project #2**

20140825

박진혁

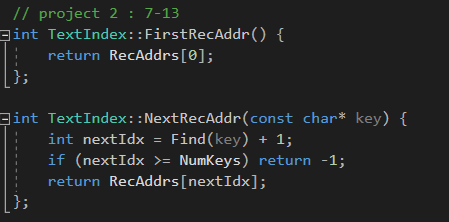
**INDEX**

1. **프로젝트 진행 중 필요한 가정**
2. **구현 클래스의 다이어그램(UML specification)**
3. **프로젝트 구성 파일 및 변경 및 추가된 자료구조**
   1. **basic class**
   2. **iobuffer**
   3. **class\_manager**
   4. **그 외 파일**
4. **프로젝트 요구 사항**
   1. **“1.1 관리자 모드”의 구현**
   2. **“1.2 인덱스”의 구현**
5. **연습문제 (7장 13,16,17,18번)**
6. **프로젝트 진행 중 필요한 가정**
   1. 레코드의 필드들은 ‘|’로 구분하여 기록한다.
   2. 입력 받는 레코드에는 \*나 공백문자가 포함되지 않는다.
   3. 입력 받는 레코드 중, 고정길이 필드에 대해서는 길이에 맞는 input이 들어온다.
   4. 입력 받는 레코드 중, 각 레코드 타입의 id에 해당하는 필드는 기존의 레코드들과 중복된 값이 입력되지 않는다.
7. **구현 클래스의 다이어그램(UML specification)**
8. **프로젝트 구성 파일 및 자료구조**



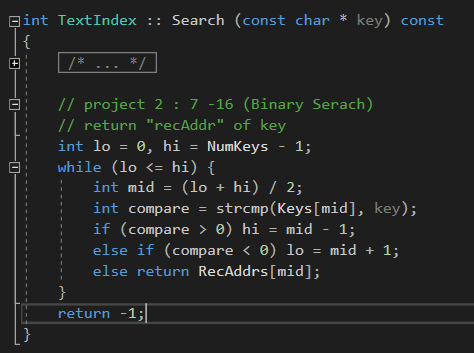
* 1. **basic\_class :** 프로젝트 내 별도의 ‘basic\_class’ 디렉토리를 생성해 관리
     1. **member.cpp과 member.h** 
        + Member 클래스의 멤버 변수로 char형 배열의 level을 추가하였다. level 필드의 기본 값은 9로, 이는 일반 사용자의 level 값이다. 또다른 멤버 변수로 string key를 추가하고, key 값을 생성하기 위한 char\* Key() 함수를 구현하였다. 이를 통해 TextIndex 관련 자료구조에서 해당 Member 레코드의 key값을 받을 수 있다.
     2. **newsAgency.cpp과 newsAgency.h**
        + NewsAgency 클래스의 멤버 변수로 string key를 추가하고, key 값을 생성하기 위한 char\* Key() 함수를 구현하였다. Member 클래스와 마찬가지로, 이를 통해 TextIndex 관련 자료구조에서 해당 Member 레코드의 key값을 받을 수 있다.
  2. **Iobuffer :** 프로젝트 내 별도의 ‘iobuffer’ 디렉토리를 생성해 관리
     1. **textind.cpp과 textind.h** 
        + TextIndex 클래스가 구현되어 있다. 기본적으로 문자열 배열 Keys와 정수형 배열 RecAddrs를 멤버 변수로 가지며, Key에 해당하는 레코드의 주소값이 매핑되어 있어, 레코드로의 빠른 접근을 돕는다.
     2. **tindbuff.cpp과 tindbuff.h**
        + TextIndexBuffer가 구현되어 있다. TextIndex에 있는 내용을 담는 Buffer로서의 역할을 한다. FixedFieldBuffer의 자식 클래스로, Key와 RecAddr를 각각의 필드로 갖는다.
     3. **indfile.h**
        + TextIndexedFile을 구현한 헤더 파일로, 레코드별로 자료형이 달라지는 템플릿 클래스이다. TextIndex 자료형의 Index, BufferFile 자료형의 IndexFile, TextIndexBuffer 자료형의 IndexBuffer, 그리고 RecordFileM<RecType>의 자료형인 DataFile을 변수로 가진다. 주어진 파일에 따라 레코드 내용이 담긴 RecordFile과 인덱스의 내용이 담긴 IndexFile에 접근할 수 있다.
  3. **class\_manager**
     1. **member\_manager.cpp과 member\_manager.h** : Member 타입의 레코드들과 관련하여 이를 관리하는 함수들을 구현. 기존의 RecFile을 이용한 레코드 처리에서, TextIndexedFile을 이용한 처리로 로직을 변경하였다.
        + **memberSerach** : TextIndexedFile의 Read를 사용하여, TextIndex의 Search를 이용한 이진탐색을 통해 찾고자 하는 레코드를 찾고, 이를 포인터 형태로 반환한다. 만약 찾지 못한다면 NULL을 반환한다.
        + **memberInsert** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 Member가 없을 경우에만, 추가적으로 정보를 입력받아 새로운 Member 객체를 만든 뒤, 이를 saveMemberRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다. 추가적으로 level 필드 값을 받을 수 있도록 수정하였다.
        + **memberUpdate** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 Member가 있을 경우에만, 기존의 Member 레코드를 삭제한 뒤, 추가적으로 정보를 입력 받는다. 이를 통해 새로운 Member 객체를 만들고 saveMemberRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다.
        + **memberDelete** : 파라미터로 string 타입의 id를 받아, 이와 일치하는 Member 객체를 찾는다. 만약 일치하는 레코드를 찾았다면, Delete 메소드를 호출하여 멤버변수인 DataFile에서 구현한 Delete를 사용해 레코드의 id 첫글자를 ‘\*’로 바꿔준다. DataFile의 Delete는 BufferFile 클래스에 구현한 Delete 메소드와, IOBuffer의 가상함수 Delete를 상속받아 실제로 구현한 VariableLengthField의 Delete를 호출해 작업을 수행한다.
        + **getMemberRecordLen** : Member 타입의 m을 파라미터로 받아, 해당 레코드가 실제로 .dat 파일에 Write 되게 될 때에 얼마만큼의 공간을 차지하게 될지를 계산하여 그 값을 반환한다.
        + **makeDummyMember** : Member 타입의 m과 만들어야 하는 레코드의 길이를 파라미터로 받아, 그 길이에 맞는 레코드가 .dat 파일에 쓰여지도록, Member의 가변 길이 필드 값을, 삭제 데이터임을 나타내는 ‘\*’로 채운다.
        + **saveMemberRecord** : Member 타입의 newMember 변수를 파라미터로 받은 뒤, 전체 DB(.dat 파일)을 순차 탐색하면서, newMember의 레코드가 들어갈 자리를 찾아 저장한다. 탐색 중 삭제된 record를 만나게 되고, 또 해당 삭제 레코드의 길이가 최소 단위의 dummy record와 newMember를 모두 포함할 수 있다면, 해당 위치에 레코드를 작성하고, 아니라면 DB의 마지막 레코드 다음 자리에 레코드를 기록한다.
     2. **newsAgency\_manager.cpp과 newsAgency\_manager.h :** NewsAgency 타입의 레코드들과 관련하여 이를 관리하는 함수들을 구현. 기존의 RecFile을 이용한 레코드 처리에서, TextIndexedFile을 이용한 처리로 로직을 변경하였다.
        + **newsAgencySerach** : 파라미터로 string 타입의 id를 받아 id와 일치하는 NewsAgency 객체를 탐색하며, 이는 .ind 파일에 대한 접근을 통한 이진 탐색으로 이루어진다. 일치하는 객체가 있을 경우 이를 포인터 형태로 반환하며, 없다면 NULL을 반환한다.
        + **newsAgencyInsert** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 NewsAgency가 없을 경우에만, 추가적으로 정보를 입력받아 새로운 NewsAgency 객체를 만든 뒤, 이를 saveNewsAgencyRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다.
        + **newsAgencyUpdate** : 콘솔 입력을 통해, id를 입력 받은 뒤, 이와 일치하는 NewsAgency가 있을 경우에만, 기존의 NewsAgency 레코드를 삭제한 뒤, 추가적으로 정보를 입력 받는다. 이를 통해 새로운 NewsAgency 객체를 만들고 saveNewsAgencyRecord 함수의 파라미터로 넣어주어 .dat 파일에 삽입한다.
        + **newsAgencyDelete** : 파라미터로 string 타입의 id를 받아, 이와 일치하는 NewsAgency 객체를 찾아, Member 클래스의 memberDelete와 마찬가지로 DataFile의 Delete 메소드를 이용해 해당 레코드에 대해 삭제처리를 해준다.
        + **getNewsAgencyRecordLen** : NewsAgency 타입의 n을 파라미터로 받아, 해당 레코드가 실제로 .dat 파일에 Write 되게 될 때에 얼마만큼의 공간을 차지하게 될지를 계산하여 그 값을 반환한다.
        + **makeDummyNewsAgency** : NewsAgency 타입의 n과 만들어야 하는 레코드의 길이를 파라미터로 받아, 그 길이에 맞는 레코드가 .dat 파일에 쓰여지도록, NewsAgency의 가변 길이 필드 값을, 삭제 데이터임을 나타내는 ‘\*’로 채운다.
        + **saveNewsAgencyRecord** : NewsAgency 타입의 newNewsAgency 변수를 파라미터로 받은 뒤, 전체 DB(.dat 파일)을 순차 탐색하면서, newNewsAgency의 레코드가 들어갈 자리를 찾아 저장한다. 탐색을 위해 수행한 TextIndexedFile의 Read 메소드 수행 결과로 반환된 recAddr값이 -1이라면, 이는 삭제된 레코드이거나 파일의 끝에 다다랐다는 뜻이다. 삭제된 레코드일 경우, 해당 삭제 레코드의 길이가 최소 더미 레코드의 길이와 현재 기록하고자 하는 레코드의 길이의 합보다 클 경우에 한해 해당 위치에 레코드를 기록해주고, 아닐 경우 다음 삭제 레코드를 찾거나 파일의 맨끝에 레코드를 기록한다.
     3. **subscription.cpp과 subscription.h** : 관리자 모드가 아닌 일반 회원 모드에서 수행해야 할 작업들을 위한 별도의 메소드들을 구현하였다.
        + **mySubscriptionInsert** : 사용자의 id를 받아 새로운 subscription 객체를 생성하고 이를 사용자와 매핑하여 db에 추가하는 메소드이다. 새롭게 생성할 subscription의 id가 다른 subscription 객체와 겹치지 않는지를 검사하고, 또 새롭게 매핑되고자 하는 NewsAgency 객체가 현재 존재하는 객체인지도 검사한다. 모든 조건을 만족하는 경우에만 객체를 db에 추가하고, 아닐 경우는 메소드를 종료한다.
        + **mySubscriptionUpdate** : 기존에 사용자와 매핑되어있던 subscription 객체에 한하여 그 내용을 수정해주고 이를 DB에 반영해주는 메소드이다. 이를 위해 수정하고자 하는 subscription의 멤버변수인 memberId가 사용자의 id와 일치하는지를 먼저 판단해 주어야 한다. 또한 mySubscriptionInsert와 마찬가지로, 새롭게 매핑하고자 하는 NewsAgency 객체가 현재 존재하는 객체인지를 판단해주어야 한다. 모든 조건을 만족하는 경우에 한해서만 객체의 수정사항을 DB에 반영한다. 이는 subscriptionDelete와 saveSubscriptioRecord 함수를 사용해 이루어진다.
  4. **그 외 파일**
     1. **usingIOBuffer.h**
        + tindbuff.h, textind.h, indfile.h를 링크해주기 위한 전처리기 부분을 추가해 주었다.
     2. **NewsAgencySubscriptionSystem.cpp** : 관리자 모드와 일반 회원 모드를 구분짓기 위한 함수 추가
        + **login** : id와 password를 input으로 받아, 해당 id에 해당하는 Member 인스턴스를 찾은 뒤, 해당 인스턴스의 멤버변수 password와 입력으로 들어온 password를 문자열 비교한 결과, 두 값이 일치하는 경우에 한해서만, NewsAgencySubscriptionSystem에 진입할 수 있다. 조건이 만족된다면 해당 Member 인스턴스를 포인터 형태로 반환한다.
        + **executeAdminSystem, executeNormalSystem** : 로그인 결과로 반환된 Member 인스턴스를 인수로 받아, 해당 인스턴스의 level에 따라 각각 관리자와 일반 회원모드의 system을 수행시킨다.
        + **renewIndexFile** : .dat 파일로부터 데이터를 읽어들여 새롭게 .ind 파일을 생성하는 메소드이다.

1. **프로젝트 요구 사항**
   1. **“1.1 관리자 모드”의 구현**
      1. NewsAgencySubscriptionSystem.cpp에 login 함수를 구현하고, main 함수에서 이를 실행하여 관리자와 일반 회원을 구분하기 위한 로그인 기능을 구현하였다.
      2. 로그인 결과에 따라 모드를 나누고, 해당 모드에 맞는 메뉴를 띄워주고 시스템을 수행해주기 위해, 기존의 printMenu와 executeSystem 함수를 각 모드별로 세분화 하였다. (printAdminMenu, printNormalMenu, executeAdminSystem, executeNormalSystem)
      3. 일반회원의 권한을 제한하기 위하여, 별도의 메소드를 subscriptionManager 클래스 내부에 구현하였다.
      4. 함수 내부 문제와 시간 제한으로 인해 참조 무결성 문제를 해결하지 못하였다.
   2. **“1.2 인덱스”의 구현**
      1. NewsAgencySubscriptionSystem.h에 renewIndexFile을 구현하여 새로운 .ind 파일을 갱신할 수 있도록 하였다.
      2. TextIndexedFile을 사용하여, .dat 파일과 .ind 파일에 동시에 접근할 수 있도록 하였다. 이를 위해 TextIndexedFile내에 새롭게 메소드를 구현하여 기존에TextIndexedFile의 다른 멤버변수 클래스에서 구현한 메소드를 사용하였다.
2. **연습문제 (7장 13,16,17,18번)**
   1. 13번



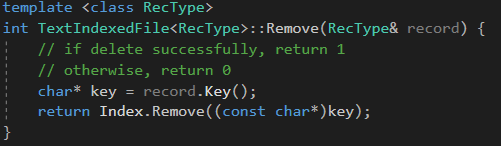
textind.cpp에 해당 두 함수를 구현하였다

* 1. 16번



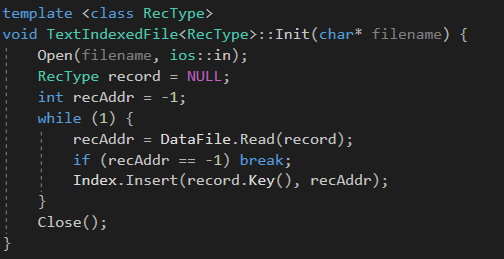
textind.cpp의 Search 함수를 수정하여 이진탐색을 수행하도록 구현하였다.

* 1. 17번

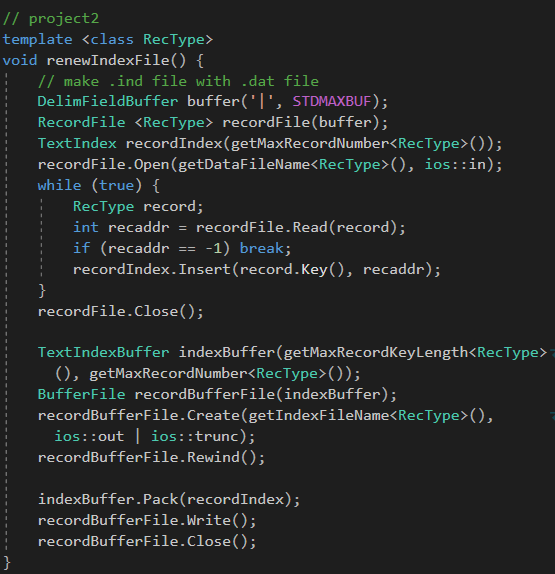


indfile.h의 TextIndexedFile에 Remove 함수를 구현하였다. Index의 remove 함수를 이용하여 Key에 해당하는 레코드를 찾아 해당 Index에서 지워준다.

* 1. 18번



indfile.h의 TextIndexedFile에 Init을 구현하였지만 정상 작동하지는 않는다.



대신, 같은 기능을 NewsAgencySubscriptionSystem.h에 renewIndexFile을 사용할 수 있다.