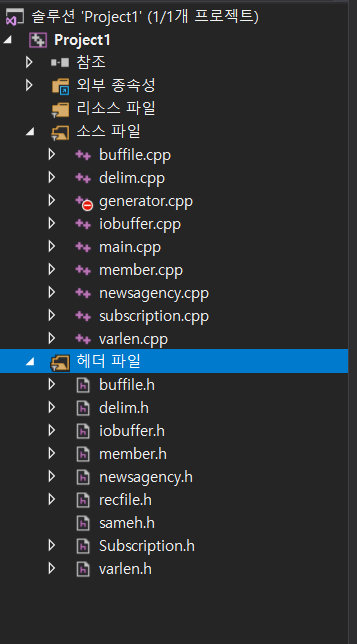
**File Processing Project #1**

*By 20150038 철학과 김재원*

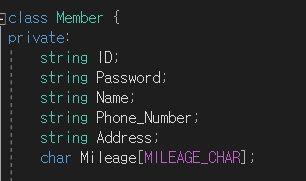
# **프로젝트 자료구조와 파일에 대한 설명**

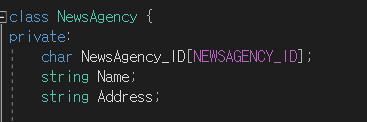


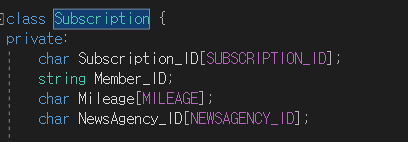
나의 프로젝트 소스 파일 구조는 위와 같다. generator.cpp는 각 클래스에 대한 List 파일을 생성하는 소스코드이므로 이번 프로젝트와 직접적인 연관성은 없다.

# **가. 프로젝트 요구사항 완료 보고**

## Basic class





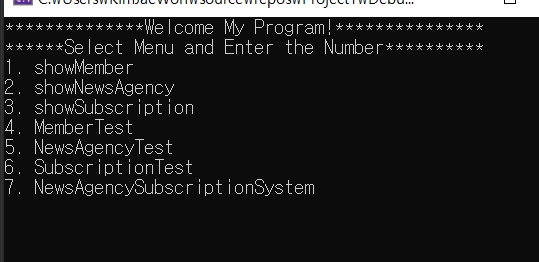


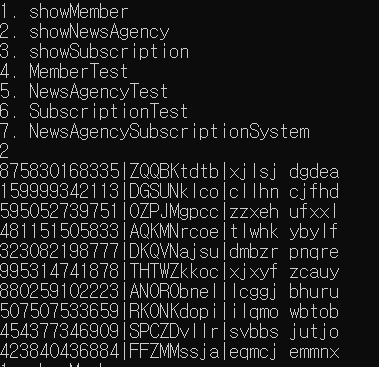
프로젝트의 기본적인 클래스 Member, NewsAgency, Subscription을 작성하였다.

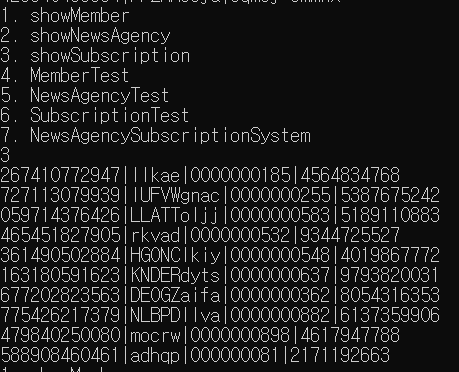
문서에서 요구한 자료형을 모두 포함했으며 char array의 경우 길이를 모두 알맞게 설정했다.

## Adding methods to basic classes

문서에서 요구하는 데로 구현하였다.

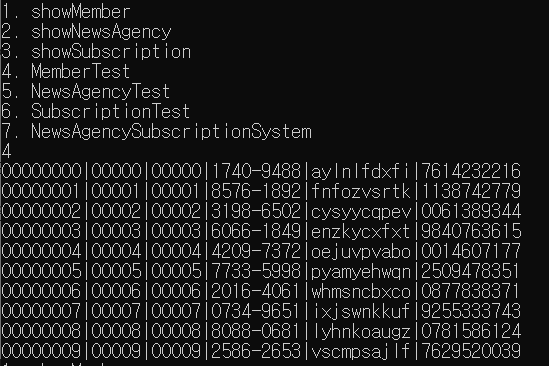


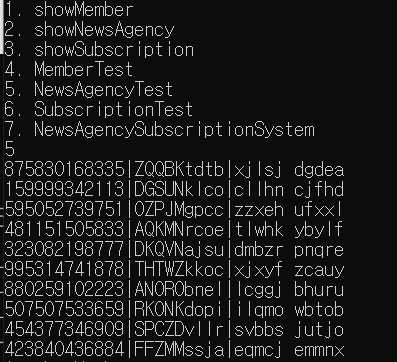


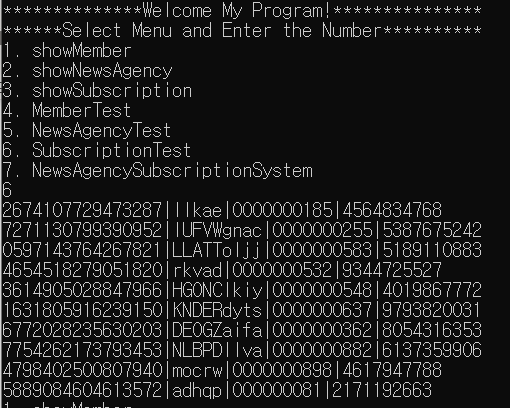


위 캡쳐들은 요구사항에서 요구한 showMember, showNewsAgency, showSubscription에 대한 구현 결과이다. 보면 요구사항에 알맞게 잘 작동되는 것을 알 수 있다.

## Using IOBuffer







위는 요구사항에 알맞게 MemberTest, NewsAgencyTest, SubscriptionTest를 구현한 것을 실행한 결과이다.

## Deleting and Updating Records

이번 프로젝트에서 검색, 삽입, 삭제, 수정을 모두 지원하는 대화식 프로그램을 작성하는 것이 중요했다. 수정은 키를 제외한 모든 필드가 가능하고, 모든 경우에 참조 무결성을 유지해야 했다. 예를들어, Member 레코드가 삭제되면 Member ID와 연관된 Subscription record도 수정되어야 했다.

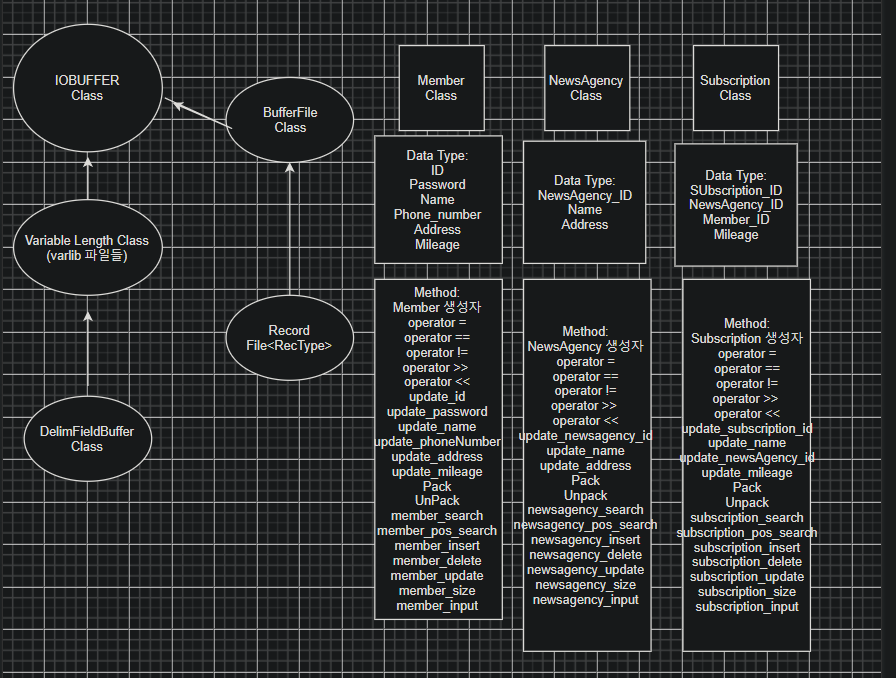
그리고 삽입과 삭제, 수정 등 파일 연산을 최대한 IOBuffer를 활용해서 dat 파일에 작업하였다. 제시된 요구사항에 등장하는 레코드파일이 .dat를 의미한다고 생각했기 때문이다.

**삭제 연산**은 특정 레코드 앞에 특수기호(\*)를 삽입하여 삭제 상태를 표현했다. Dat 파일을 Read할 때 삭제 표시된 레코드는 읽지 않도록 작업했다. 원본 .txt 파일을 작업한 후 새롭게 .dat 파일을 만드는 과정을 반복할 수도 있지만 비효율성이 워낙 극심해서 좋지 않은 방법이라 판단했다. 이 방법을 사용하면 연산을 최대한 적게 하면서 레코드가 삭제된 효과를 낼 수 있다. 단, 삭제된 레코드가 여전히 파일의 공간을 차지하고 있기 때문에 공간에 민감한 프로그램이라면 이 방법은 부적절하다. 그 외 대부분의 경우에 이 방법은 유효하다. 파일에 대한 접근과 연산을 최소화하기 때문이다.

**삽입 연산**은 우선 삭제된 레코드, 즉 가용 공간 중 삽입하려는 공간에 최초로 적합한 공간을 발견하면 그곳에 삽입하도록 했다. 왜냐하면 레코드 탐색 기법으로 ‘순차 탐색 기법’을 채택했기 때문이다. 삽입 후 남는 공간은 삭제 레코드처럼 수정해서 다음에 활용할 수 있도록 만들었다. 파일의 레코드가 정렬되어 있지 않고 새로운 레코드가 정렬되어 입력되는 지도 알 수 없기 때문에 순차탐색 기법을 채택했다. 삽입 연산을 하다가 적절한 공간을 찾지 못하면 자동으로 파일 끝에 추가되도록 했다.

**수정 연산**은 우선 사용자로부터 수정 대상의 ID를 입력받은 후 순차탐색을 거쳐서 대상 레코드를 찾는다. (찾지 못하면 찾지 못했다고 알려주고 연산을 종료한다.) 사용자가 수정하려고 하는 정도가 기존에 있던 레코드의 길이보다 길다면 파일에 저장된 레코드를 삭제 레코드 처리하고 사용자가 수정한 레코드를 파일의 끝에 추가한다. 만약 사용자 수정 레코드가 기존 레코드보다 짧다면 사용자 수정 레코드를 기존 레코드의 자리에 삽입하고 남는 공간은 삭제 레코드로 만들어서 공간 재사용성을 높혔다.

# **클래스 다이어그램**



클래스 다이어그램은 위와 같다.

# **교과서 프로그램 연습문제**

* 1. 21 고정길이 레코드, 레코드를 삭제하려면 물리적으로 삭제하거나, 논리적으로 삭제할 수 있다. 논리적으로 삭제하는 방법이 연산 비용이 상대적으로 저렴하며 물리적 삭제와 굉장히 유사한 효과를 낼 수있다. 레코드의 비활동상태(삭제)를 나타내기 위해 \* 표시를 레코드에 추가했다. Buffer Read를 할 때 레코드에 \*이 있으면 읽지 않도록 IOBuffer 클래스를 수정했다.
  2. 22 고정길이 레코드, Read와 마찬가지로 Append를 할 때도 \* 표시를 보고 삭제된 레코드임을 인지한 후 추가 작업을 수행할 수 있다. 그 레코드 위에 덮어쓰는 방식으로 Append 작업을 수행할 수 있다.
  3. 23 가변길이 레코드, 가변길이 레코드에서도 고정길이 레코드와 유사한 방식으로 삭제 연산을 수행할 수 있다. 다만 가변길이 레코드는 길이가 일정치 않아서 잘못 덮어씌면 다른 레코드의 영역을 침범해서 큰 오류를 야기할 수 있다. 그러므로 삭제 레코드의 리스트를 크기 순으로 정렬해서 보관하는 것이 필요하다.
  4. 24 가변길이 레코드, 크기 순으로 정렬되어 있는 삭제 리스트를 보고 최초로 적합한 리스트에 Append 연산을 수행한 후 그 남는 리스트는 다시 삭제 리스트로 반환한다. 이런 식으로 단편화를 최소화할 수 있다. 만약 적합한 공간이 없으면 파일의 끝에 새로운 레코드를 추가한다.
  5. 25 Update 연산을 수행할 때 Update 되는 레코드와 새로 넣으려는 레코드의 길이가 상이할 수 있다. 이역시 Append와 마찬가지로 다른 레코드의 영역을 침범하는 문제를 야기하거나 유휴 공간을 남발해서 나중에 심각한 메모리 누수를 야기할 수 있다. 이를 관리하기 위해서 레코드를 수정한 후 남는 공간은 삭제 레코드 처리해서 삭제 리스트로 돌려준다. 이런 방식으로 유휴공간을 최소화할 수 있다.