File Processing

Programming Project #1

컴퓨터공학과

20131574

오현석

**1. 프로젝트 목적**

본 프로젝트에서는 인터넷 강의의 회원 관리를 위해 회원 정보와 강의 수강 내역에 관한 정보를 처리하고 유지하는 정보 시스템을 구축한다.

**2. 요구사항.**

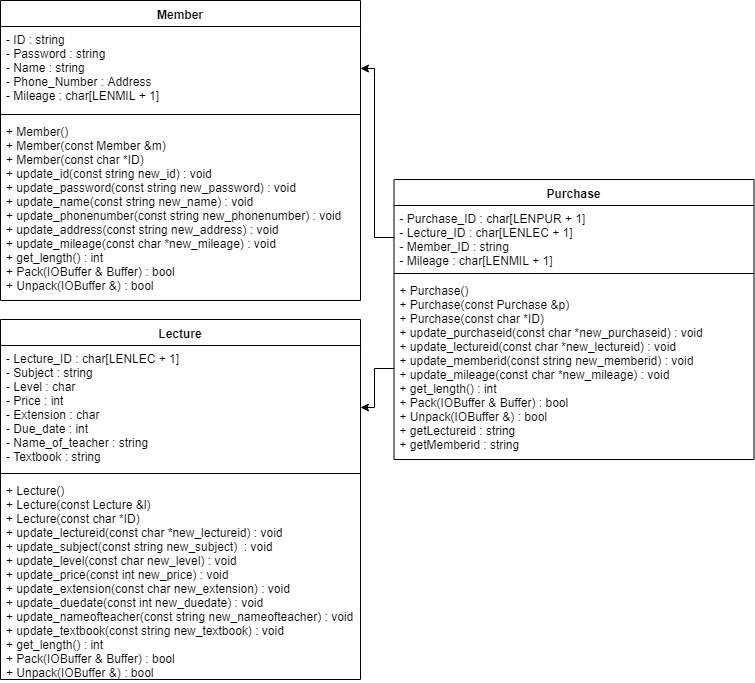
1) 프로젝트의 기본 클래스 작성

2) 기본 클래스 별 필요 속성과 메소드 정의 및 구현

3) 요구에 맞는 테스트 데이터 생성

4) 필요 기능 추가 및 구현

**3. 클래스 다이어그램**

****

**4. 추가된 파일에 대한 설명**

**[main.cpp]**

사용자 인터페이스를 담당하는 모듈이다. LecturePurchaseSystem가 구현되어 있다.

**[Member.h]**

Member Class의 Field 자료구조 및 각 Field의 setter가 구현되어 있다. 또한 해당 객체의 전체 길이를 반환해주는 함수도 구현되어 있다.

**[Member.cpp]**

Member Class의 객체간의 비교/복사 연산에 사용될 오버로드된 연산자와 Pack/Unpack method 및 object의 출력과 입력에 사용될 method가 구현되어 있다. 명세서에 제시된 show 함수들과 Test 함수들, Member data를 Search, Insert, Delete, Update 해주는 method도 구현되어 있다.

**[Lecture.h]**

Lecture Class의 Field 자료구조 및 각 Field의 setter와 getter가 구현되어 있다. 또한 해당 객체의 전체 길이를 반환해주는 함수도 구현되어 있다.

**[Lecture.cpp]**

Lecture Class의 객체간의 비교/복사 연산에 사용될 오버로드된 연산자와 Pack/Unpack method 및 object의 출력과 입력에 사용될 method가 구현되어 있다. 명세서에 제시된 show 함수들과 Test 함수들, Lecture data를 Search, Insert, Delete, Update 해주는 method도 구현되어 있다.

**[Purchase.h]**

Purchase Class의 Field 자료구조 및 각 Field의 setter와 getter가 구현되어 있다. 또한 해당 객체의 전체 길이를 반환해주는 함수도 구현되어 있다.

**[Purchase.cpp]**

Purchase Class의 객체간의 비교/복사 연산에 사용될 오버로드된 연산자와 Pack/Unpack method 및 object의 출력과 입력에 사용될 method가 구현되어 있다. Lecture\_ID와 Member\_ID를 가져오는 method, 명세서에 제시된 show 함수들과 Test 함수들, Purchase 객체를 Search, Insert, Delete, Update 해주는 method도 구현되어 있다.

**4. 각 클래스에 관한 설명<자료구조(Field) 및 Method>**

**Class <Member>**

이 객체는 각각 한 명의 회원에 대한 정보를 나타낸다. ID, Password, Name, Phone Number, Address, Mileage 의 private field와 이 object의 여러 형식의 생성자, field들을 수정하는 method, object간 비교/복사 연산에 사용될 오버로드된 연산자로 구성된다. 각 field들의 제한조건은 다음 표와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| Field(Data) | Type | update method  (setter) | getting method  (getter) |
| ID | variable string | update\_id | X |
| Password | variable string | update\_password | X |
| name | variable string | update\_name | X |
| Phone\_Number | variable string | update\_phonenumber | X |
| Address | variable string | update\_address | X |
| Mileage | 10 characters | update\_mileage | X |

위의 method 외에도 Pack/Unpack method와 Object 전체의 길이를 return 해주는 method, object의 출력과 입력에 사용될 method로 구성된다. 명세서에 제시된 show 함수들과 Test 함수들, Member data를 Search, Insert, Delete, Update 해주는 method도 구현되어 있다.

**Class <Lecture>**

이 객체는 각각 하나의 강의에 대한 정보를 나타낸다. Lecture ID, Subject, Level, Price, Extension, Due date, Name of teacher, Textbook의 private field와 이 object의 여러 형식의 생성자, field들을 수정하는 method, object간 비교/복사 연산에 사용될 오버로드된 연산자로 구성된다. 각 field들의 제한조건은 다음 표와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| Field(Data) | Type | update method  (setter) | getting method  (getter) |
| Lecture\_ID | 12 characters | update\_lectureid | X |
| Subject | variable string | update\_subject | X |
| Level | 1 characters | update\_level | X |
| Price | 1 integer | update\_price | X |
| Extension | 1 characters | update\_extension | X |
| Due date | 1 integer | update\_duedate | X |
| Name of teacher | variable string | update\_nameofteacher | X |
| Textbook | variable string | update\_textbook | X |

위의 method 외에도 Pack/Unpack method와 Object 전체의 길이를 return 해주는 method, object의 출력과 입력에 사용될 method로 구성된다. 명세서에 제시된 show 함수들과 Test 함수들, Lecture data를 Search, Insert, Delete, Update 해주는 method도 구현되어 있다.

**Class <Purchase>**

이 객체는 각각 하나의 상품 구매 내역에 대한 정보를 나타낸다. Purchase ID, Lecture ID, Member ID, Mileage의 private field와 이 object의 여러 형식의 생성자, field들을 수정하거나 갱신할 method, object간 비교/복사 연산에 사용될 오버로드된 연산자로 구성된다. 이때 Lecture ID와 Member ID는 실제 존재하는 정보에 기반해야 한다.(무결성유지) 각 field들의 제한조건은 다음 표와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| Field(Data) | Type | update method  (setter) | getting method  (getter) |
| Purchase\_ID | 16 characters | update\_purchaseid | X |
| Lecture\_ID | 12 characters | update\_lectureid | getLectureid |
| Member\_ID | variable string | update\_memberid | getMemberid |
| Mileage | 10 characters | update\_mileage | X |

위의 method 외에도 Pack/Unpack method와 Object 전체의 길이를 return 해주는 method, object의 출력과 입력에 사용될 method로 구성된다. 이 클래스는 특히 Lecture ID나 Member ID가 갱신되거나 삽입될 때, 기존의 Member 클래스나 Lecture 클래스에 해당 ID가 존재하여야 한다. 명세서에 제시된 show 함수들과 Test 함수들, Purchase data를 Search, Insert, Delete, Update 해주는 method도 구현되어 있다.

**7. showMember/showLecture/showPurchase**

**showMember()**

입력 스트림으로부터(listOfMember.txt) Member의 Field값을 읽고 출력 스트림에 Field값을 출력하기 위해 추가한 오버로딩 함수를 추가하였다는 확인하기 위한 Test Program이다. 작성된 레코드들을 읽은 뒤 표준 출력으로 앞의 10개의 레코드만 출력한다.

**showLecture()**

입력 스트림으로부터(listOfLecture.txt) Lecture의 Field값을 읽고 출력 스트림에 Field값을 출력하기 위해 추가한 오버로딩 함수를 추가하였다는 확인하기 위한 Test Program이다. 작성된 레코드들을 읽은 뒤 표준 출력으로 앞의 10개의 레코드만 출력한다.

**showPurchase()**

입력 스트림으로부터(listOfPurchase.txt) Purchase의 Field값을 읽고 출력 스트림에 Field값을 출력하기 위해 추가한 오버로딩 함수를 추가하였다는 확인하기 위한 Test Program이다. 작성된 레코드들을 읽은 뒤 표준 출력으로 앞의 10개의 레코드만 출력한다.

**8. MemberTest/LectureTest/PurchaseTest**

**memberTest()**

Member 클래스에 추가한 Pack과 Unpack 함수를 확인하기 위한 Test Program이다. listOfMember.txt로부터 데이터를 읽어 들인 후, Pack(IOBuffer)함수를 사용하여 fileOfMember.dat 파일을 작성한다. 작성된 fileOfMember.dat 파일에서 Unpack 함수를 사용하여 저장된 레코드들을 읽은 뒤 표준출력으로 앞의 10개의 레코드만 출력한다.

**LectureTest()**

Lecture 클래스에 추가한 Pack과 Unpack 함수를 확인하기 위한 Test Program이다. listOfLecture.txt로부터 데이터를 읽어 들인 후, Pack(IOBuffer)함수를 사용하여 fileOfLecture.dat 파일을 작성한다. 작성된 fileOfLecture.dat 파일에서 Unpack 함수를 사용하여 저장된 레코드들을 읽은 뒤 표준출력으로 앞의 10개의 레코드만 출력한다.

**purchaseTest()**

Purchase 클래스에 추가한 Pack과 Unpack 함수를 확인하기 위한 Test Program이다. listOfPurchase.txt로부터 데이터를 읽어 들인 후, Pack(IOBuffer)함수를 사용하여 fileOfPurchase.dat 파일을 작성한다. 작성된 fileOfPurchase.dat 파일에서 Unpack 함수를 사용하여 저장된 레코드들을 읽은 뒤 표준출력으로 앞의 10개의 레코드만 출력한다.

**9. LecturePurchaseSystem**

각 데이터파일(fileOfMember.dat, fileOfLecture.dat, fileOfPurchase.dat)을 통해서 검색, 삽입, 삭제, 수정을 지원하는 대화식 프로그램. 검색(Search)/삽입(Insert)/삭제(Delete)/수정(Update)의 과정들은 각각의 ID들을 통해 이루어진다. 모든 경우에서 참조 무결성을 유지해야하며, 수정에서는 ID를 제외한 모든 필드가 가능하다.

**<Search>**

**record의 ID로 검색 후 검색한 레코드를 출력한다. Purchase의 경우 Purchase ID뿐만 아니라 Member ID나 Lecture ID를 통한 검색도 지원한다. 해당 함수를 호출할 때 플래그 값을 파라미터로 넘겨주어 특정 ID에 맞는 Record를 검색해야 할 때 사용할 수 있게 하였다.**

**Search에 적용한 방법.**

**:** 순차검색을 통해 해당 ID에 관련된 Record들을 검색한다.

**<Insert>**

record의 ID를 사용하여 검색 후, 만약 존재하지 않는 ID라면 해당 record에 대한 삽입을 진행할 수 있게 한다. Insert과정에서 형식에 맞지 않는 data를 입력하려 하면 에러 메세지를 출력한다.

**Insert에 적용한 방법.**

**:** 위의 Search함수를 활용하여 해당 ID를 순차적으로 탐색하고, 이를 찾지 못했을 때, 해당 ID에 맞는 데이터들을 삽입할 수 있게 한다.

**<Delete>**

record의 ID를 사용하여 검색 후, 참조 무결성을 유지하면서 해당 ID에 대한 record들을 삭제한다.

**Delete에 적용한 방법.**

**:** 위의 Search함수를 활용하여 해당 ID를 순차적으로 탐색하고, 이를 찾았을 때 해당 ID를 삭제하고 Member 또는 Lecture를 삭제한 경우, 해당 ID가 있는 Purchase의 record도 삭제한다.

**<Delete Compaction 전략>**

순차적으로 탐색 후, 해당 ID에 관련된 Record를 삭제했을 때는 그 공간만큼을 Shift하여서 메모리 중간의 빈 공간을 없앤다. 추후 삽입 시에는 Record의 맨 뒤로 삽입한다.

처음에는 수업시간에 배운 것처럼 '\*'와 같은 Special Character를 통하여 삭제된 Record를 표기하고 Available List를 만들려고 하였으나, Member, Lecture, Purchase Class 모두 가변길이 Record가 포함된 Class들이기 때문에 길이 검색이 어려웠고, 또한 길이를 검색하여 그만큼의 공간을 저장해둔다 하더라도, Memory상의 쓰레기 값들이 발생하여 처리하기가 지나치게 어려웠다. 또한 Available List에 지나치게 쪼개진 메모리들이 남아서 이를 관리하기가 매우 어려웠다. 따라서 지금은 Shift하는 방식으로 구현하였지만,(실질적으로는 파일에 다시 읽고 쓰는 방식이지만) 추후 프로젝트가 진행되며 개선될 것이라 생각된다.

**<Update>**

record의 ID를 사용하여 검색 후, 존재하는 ID라면 해당 record의 대한 정보를 수정할 수 있게 한다. Update 과정에서 형식에 맞지 않는 data를 입력하려 하면 에러 메세지를 출력 한다.

**10. 6장 연습문제**

**<21>**

처음 프로젝트 관련 프로그래밍을 했을 때는, 삭제된 레코드 앞에 ‘\*’를 붙여서 처리했다. Read를 하였을 때 처음이 ‘\*’인 레코드는 인식하지 않도록 처리하였다. (그러나 이 구현은 추후 가변길이 레코드를 처리할 때, 삭제된 공간만큼 Shift하는 것으로 처리하였다.)

**<22>**

삭제된 레코드만큼의 길이를 갖는 만큼을 available\_list로 편성하여 관리하였다. 추후 레코드가 삽입될 때는 이 available\_list에서 먼저 적절한 공간이 있는지 확인한 후, 존재한다면 그 곳에 삽입하고, 존재하지 않는다면 맨 뒤에 삽입하는 것으로 구현하였었다. 그러나 이 구현은 추후 가변길이 레코드를 처리할 때 Memory상의 쓰레기 값들이 발생하여 처리하기가 지나치게 까다롭고, 더하여 Available List에 지나치게 쪼개진 메모리들이 남아서(앞으로도 영원히 사용되지 않을) 최종적으로는 Shifting하는 것으로 구현하였다.

**<23>**

가변길이 레코드를 처리할 때, 삭제된 레코드 앞에 '\*'와 같은 Special Character를 추가하여 처리하면, 매번 해당 레코드의 전체 길이를 구하여 처리해야 한다. 특히 현재 Record들은 특정 Index에 의해 정렬된 것이 아니기 때문에, 삭제하고 싶은 Record를 찾기 위해서는 어차피 순차탐색이 이루어져야 한다. 따라서 여기서 Shifting하여 빈 공간을 없애는 방식으로 처리하여도 괜찮다고 판단된다. 추후 Record에 특정 Index가 추가되고 이를 통해 정렬되어야 한다면 Delete의 구현을 조금 더 수정해야 할 것으로 보인다.

**<24>**

가변길이 레코드의 Delete를 Shifting을 통해 구현하였기 때문에 현재는 Append를 수정할 필요가 없지만, 만약 Record에 특정 Index가 추가되고, 이 Index를 통해 정렬되어 저장되어 있다면 Delete연산이 수정되어야 하며 Delete연산이 수정된다면 Append연산도 이에 따라 수정되어야 한다. 맨 처음 가변길이 레코드를 ‘\*’를 통해 처리했을 때, Memory상의 쓰레기 값 문제, Fragmentation 문제, 정렬되지 않은 문제가 발생하여 Shift를 통해 Delete를 수행하고, 레코드모음의 맨 뒤로 삽입하는 방식으로 구현했기 때문이다.

**<25>**

update 연산을 추가하였다. 현재는 Shifting방식을 통해 구현하였기 때문에 variable length string을 입력할 때는 문제가 없고 또한 원래의 string보다 작은 크기의 string이 입력되어야 한다 하더라도 Data Compaction에 대해서는 문제가 발생하지 않는다. 즉, 최적화 되어있다. 또한, 정해진 형식이 있는 데이터는 별도의 분기점을 두어서, 정해진 형식과 일치하지 않는다면 update가 수행되지 않도록 처리하였다.