**FILE PROCESSING (CSE4095)**

**Project 1**

학과 : 컴퓨터공학과

학번 : 20181593

이름 : 계인혜

**<목차>**

1. **프로젝트 설정상의 가정**
2. **클래스 다이어그램(standard UML specification)**
3. **파일 및 자료구조 설명**

3.1 iobuffer

3.2 userClass

3.3 dataGen

3.4 Etc

1. **프로그래밍 문제 요구 사항**

4.1 Basic Class 구현

4.2 Adding methods to basic classes

4.3 Using IOBuffer

4.4 Deleting and Updating Records

1. **6장 연습문제**

5.1 #21

5.2 #22

5.3 #23

5.4 #24

5.5 #25

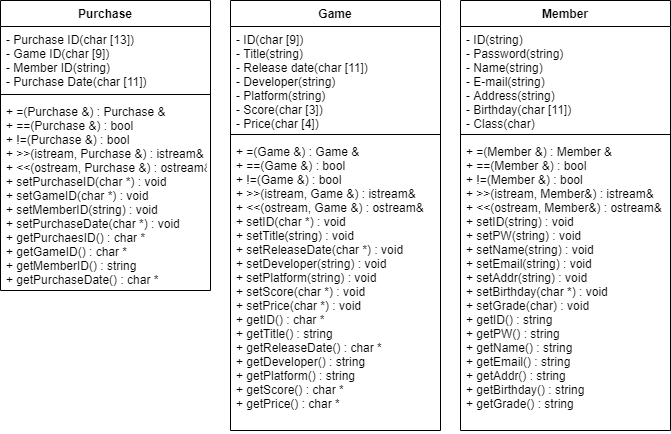
1. **프로젝트 설정상의 가정**

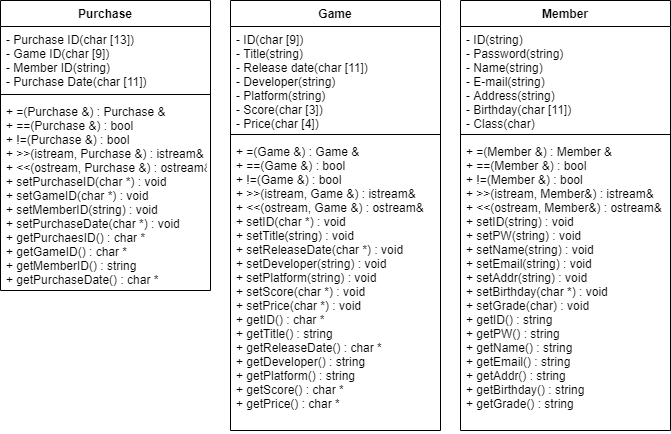
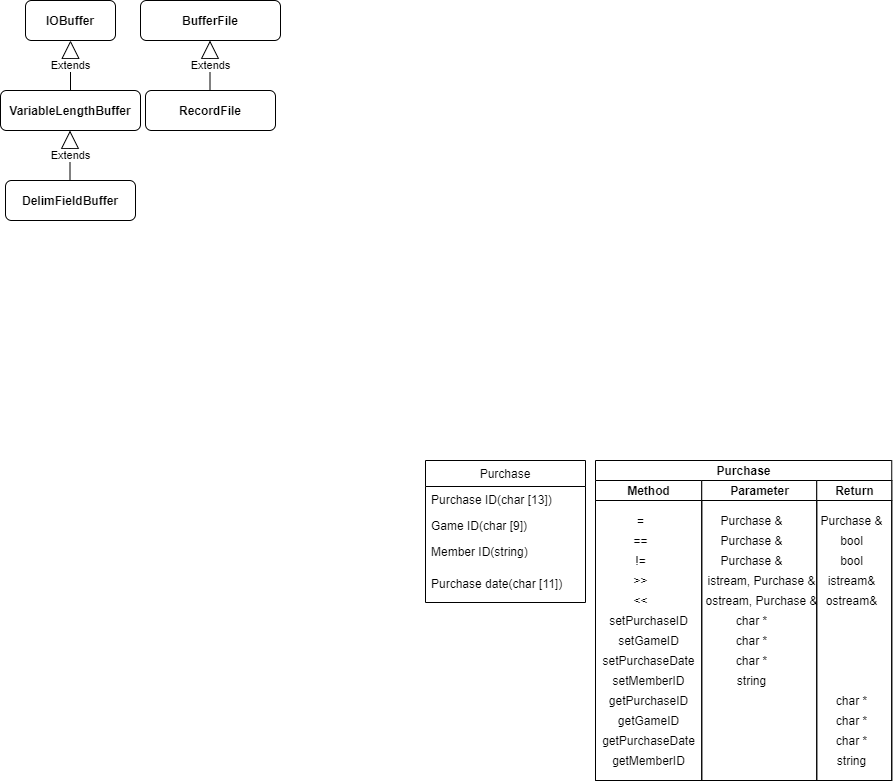
1.1 각 필드의 구분자는 ‘|’로 한다.

1.2 사용자는 레코드 추가 시 중복된 값을 입력하지 않는다.

1.3 사용자는 입력 시 ‘|’ , ‘\*’, 공백을 제외한다.

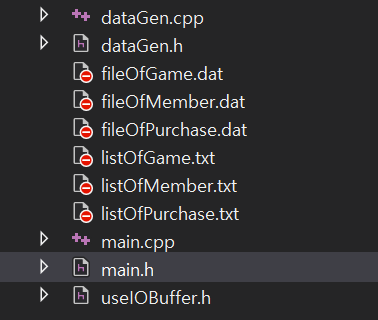
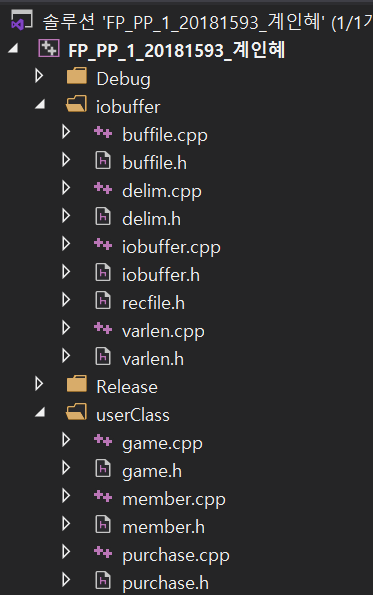
1. **클래스 다이어그램(standard UML specification)**

****

****

1. **파일 및 자료구조 설명**

파일 및 자료구조에 대한 전체적인 사진은 아래에 첨부하였다.

****

* 1. iobuffer

해당 클래스는 알 수 없는 오류로 인해 iobuffer라는 디렉토리안에 넣어 필요한 소스에 include하였다.

* + 1. iobuffer.h & iobuffer.cpp

IOBuffer 클래스가 구현되어 있고, 헤더 파일에는 가상 메소드들이 들어있다. cpp파일에는 IOBuffer 클래스를 상속받는 클래스들에서 사용되는 메소드들을 정의하고 있다.

* + 1. varlen.h & varlen.cpp

IOBuffer 클래스를 상속받는 VariableLengthBuffer 클래스가 구현되어 있고, 이 클래스는 가변길이 필드를 처리하는데 사용된다.

* + 1. delim.h & delim.cpp

VariableLengthBuffer 클래스를 상속받는 DelimFieldBuffer 클래스가 구현되어 있고, 이 클래스는 구분자(Delimiter)를 처리하는데 도움을 준다.

* + 1. buffile.h & buffile.cpp

IOBuffer와 fstream을 포함하는 Buffile 클래스가 구현되어 있고, 이 클래스는 파일과 관련된 연산을 수행한다. 이 연산에는 Read, Write, Open, Close, Create, Append 등이 있다.

* 1. userClass
     1. member.h & member.cpp

Member 클래스를 선언한 h 파일과 이에 해당하는 메소드들을 구현한 cpp 파일로 구성되어 있다. 보고서에 나온 데이터를 타입에 맞게 모두 멤버 변수로 가지고 있다.

* + 1. game.h & game.cpp

Game 클래스를 선언한 h 파일과 이에 해당하는 메소드들을 구현한 cpp 파일로 구성되어 있다. 보고서에 나온 데이터를 타입에 맞게 모두 멤버 변수로 가지고 있다.

* + 1. purchase.h & purchase.cpp

Purchase 클래스를 선언한 h 파일과 이에 해당하는 메소드들을 구현한 cpp 파일로 구성되어 있다. 보고서에 나온 데이터를 타입에 맞게 모두 멤버 변수로 가지고 있다.

* 1. dataGen

getTXT : 클래스 타입에 맞춰 txt 파일을 생성한다.

getDAT : 클래스 타입에 맞춰 dat 파일을 생성한다.

randomGen : size에 맞는 임의의 숫자를 string 타입으로 리턴한다.

formatting : int(num)를 size 자리 수에 맞춰 string으로 변환한다.

memberGen : 프로젝트 명세서에 기록된 Member 클래스의 멤버 변수를 타입에 맞게 listOfMember.txt 파일에 임의로 저장한다. 이때 각 레코드는 중복되지 않으며, 첫 부분에 레코드의 개수를 같이 저장한다.

gameGen : 프로젝트 명세서에 기록된 Game 클래스의 멤버 변수를 타입에 맞게 listOfGame.txt 파일에 임의로 저장한다. 이때 각 레코드는 중복되지 않으며, 첫 부분에 레코드의 개수를 같이 저장한다.

purchaseGen : 프로젝트 명세서에 기록된 Purchase클래스의 멤버 변수를 타입에 맞게 listOfPurchase.txt 파일에 임의로 저장한다. 이때 각 레코드는 중복되지 않으며, 첫 부분에 레코드의 개수를 같이 저장한다.

* 1. Etc

useIOBuffer.h : iobuffer와 관련된 파일들을 한꺼번에 include 할 수 있도록 만든 헤더파일이다.

이하는 메인 함수 내부에 구현된 메소드들에 대한 설명이다.

<T>showData : 클래스 타입 T에 해당하는 레코드를 10개까지 화면에 출력한다.

<T>dataTest : 클래스 타입 T에 해당하는 txt 파일을 읽어 dat 파일에 저장한다. 이때 10개의 레코드는 화면에 출력한다.

displayMenu : 프로그램 실행 초기에 뜨는 메뉴화면을 출력한다.

GamePurchaseSystem : 프로젝트 명세서에 나온 함수로 Member, Game, Purchase 레코드의 검색, 삽입, 삭제, 수정을 지원하는 대화식 프로그램이다.

searchRecord : 레코드의 검색을 위한 함수로 클래스 타입에 맞게 작동한다.

insertRecord : 레코드의 삽입을 위한 함수로 클래스 타입에 맞게 작동한다.

deleteRecord : 레코드의 삭제를 위한 함수로 클래스 타입에 맞게 작동한다.

modifyRecord : 레코드의 수정을 위한 함수로 클래스 타입에 맞게 작동한다.

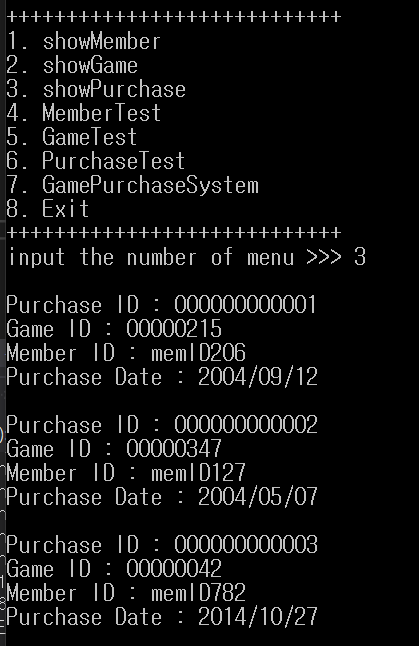
1. **프로그래밍 문제 요구 사항**

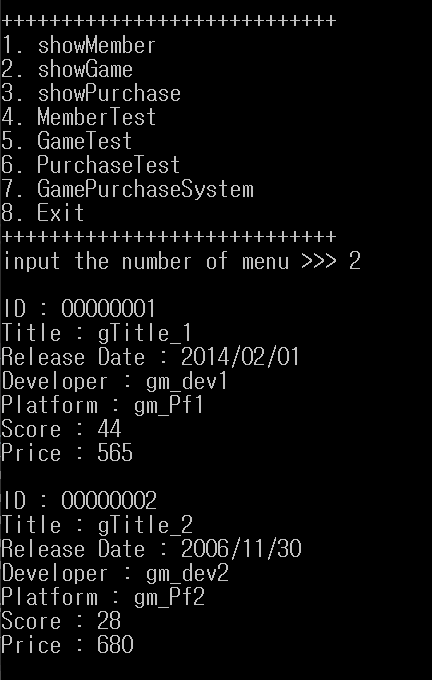
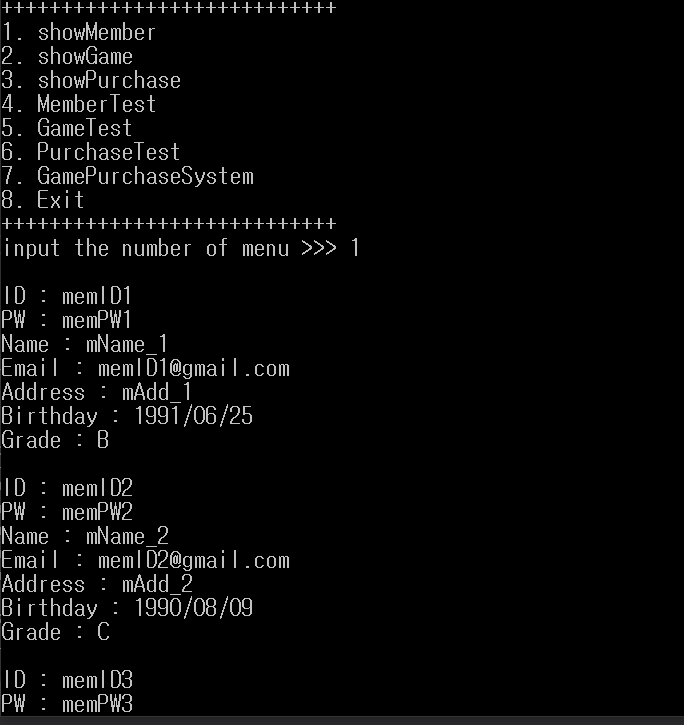
4-1. Basic Class 구현

Member, Game, Purchase 클래스를 구현하였으며, 명세서에 표기된 데이터 타입, 길이, 생성자, 연산자 오버로드 및 필드 값 갱신 함수를 포함하였다. 클래스의 헤더 및 소스 파일은 관리의 편의성을 위하여 userClass라는 디렉토리에 넣었다.

4-2. Adding methods to basic classes

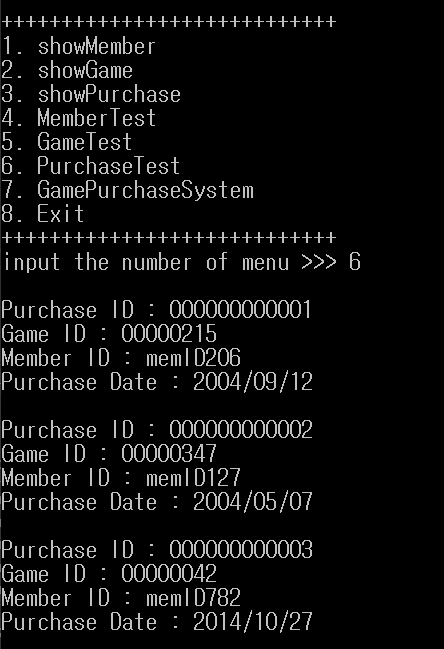
Operator >>, <<를 오버로드하는 함수를 추가하였으며 성공적으로 잘 작동하는 것을 확인하였다. 또한 이를 콘솔창에 출력하여 확인할 수 있는 showData 함수 역시 추가하였다. 이 함수는 showMember, showGame, showPurchase를 따로 구현하는 대신 클래스의 타입에 맞게 함수 하나로 세 클래스를 모두 출력할 수 있다.

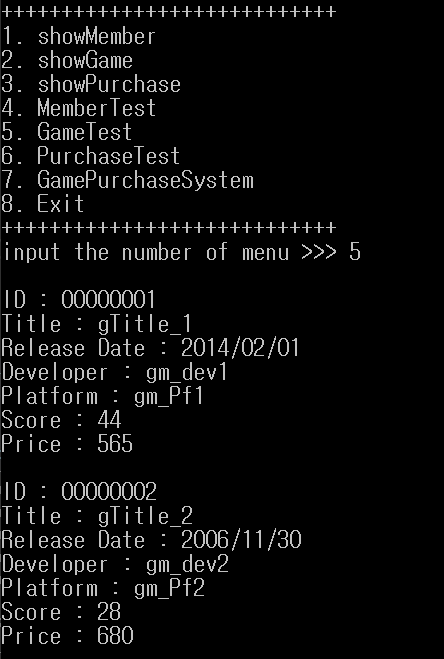
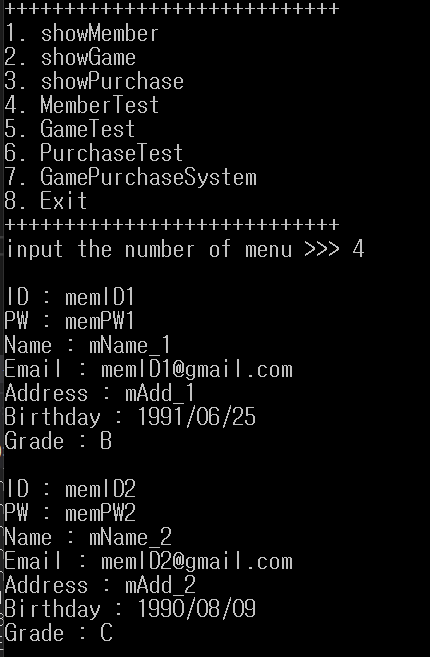




4-3. Using IOBuffer

dataTest 함수를 이용하여 memberTest, gameTest, purchaseTest 함수를 각각 구현하는 대신 클래스의 타입에 맞게 함수 하나로 구현하였다.





4-4. Deleting and Updating Records

Member, Game 클래스의 레코드를 삭제하는 경우 이에 해당하는 Purchase 레코드 역시 삭제하여 참조 무결성을 유지하였다. 레코드의 삭제를 표시하기 위해 필드의 맨 첫번째 글자를 ‘\*’로 변환하여 저장하였다. 레코드의 수정은 클래스의 키를 제외한 모든 필드에서 가능하고 수정 시에도 참조 무결성을 유지한다. 삭제 후 dat 파일을 다시 읽어보면 삭제된 레코드 앞에 ‘\*’ 표시가 있어 레코드의 삭제를 확인할 수 있다.

1. **6장 연습문제**

5-1. #21

고정길이 레코드는 삭제된 레코드의 필드 값을 ‘\*’로 채워 삭제를 표시한다. 길이가 고정되어 있기 떄문에 삭제된 레코드를 만나면 레코드의 길이만큼 뒤로 가서 탐색을 이어갈 수 있다.

5-2. #22

레코드를 sequential search 할 때 키 필드가 ‘\*’를 포함하고 있으면 이 레코드는 삭제된 것으로 판단한다. 따라서 새 레코드를 삭제된 레코드에 배치하기 위해서 이 표시를 이용하여 Append 한다.

5-3. #23

가변길이 레코드의 삭제 시 필드 값의 첫 글자를 ‘\*’로 변환하여 저장한다. 이 표시를 이용하여 삭제된 다음 레코드의 탐색을 할 수 있다.

5-4. #24

레코드를 sequential search할 때, 키 값이 ‘\*’로 시작한다면 해당 레코드는 삭제된 것으로 판단하고 삽입될 레코드와 삭제된 레코드의 크기를 비교한다. 이때 삽입될 레코드가 삭제된 레코드보다 크기가 작다면 삭제된 레코드의 자리에 레코드를 삽입한다.

5-5. #25

수정될 레코드와 기존 레코드의 크기를 비교하여 수정될 레코드가 기존 레코드보다 크다면 기존 레코드의 필드에 ‘\*’를 넣어 삭제를 하고, 데이터의 맨 끝에 수정된 레코드를 다시 삽입해야 한다. 만약 수정될 레코드가 기존 레코드보다 작다면, 남는 부분은 공백으로 채우고 필드 값을 수정하면 된다. 예를 들어 수정될 레코드의 값은 [p,r,o,j,e,c,t]이고, 기존 레코드의 값은 [f,i,l,e, ,p,r,o,c,e,s,s,i,n,g]이라면 수정될 레코드의 길이가 기존 레코드의 길이보다 작기 떄문에 기존 레코드의 남는 부분을 공백으로 대체한다. 즉, 기존 레코드는 [f,i,l,e, ,p,r, , , , , , , , ]가 된다. 이후 수정될 레코드의 값을 기존 레코드에 넣으면 레코드의 수정([p,r,o,j,e,c,t, , , , , , , ,])을 할 수 있다.