**File Processing**

**Programming Project #1**

**컴퓨터공학과**

**20141602**

**황기덕**

1. **프로젝트 목적**

* 본 프로젝트에서는 뉴스 구독 서비스의 회원 관리를 위해 회원 정보와 뉴스 구독 내역에 관한 정보를 처리하고 유지하는 정보 시스템을 구축한다.

1. **프로젝트 기본 가정**

* 기본 입력으로 주어진 .txt파일의 record들의 id는 서로 중복되지 않는다.
* .txt 파일에서 newsagency와 member는 다 대 다 관계, member와 subscription, newsagency와 subscription은 1 대 다의 관계를 가진다.
* 각각의 field는 ‘|’ 를 이용하여 구분한다.
* 처음 프로그램을 실행 시킨 후 4, 5, 6을 누르거나, 7을 누르면 txt파일을 통해 .dat 파일이 생성된다.

1. **클래스 field, method**

* **Member**

Member class의 각 instance는 한 명의 회원에 대한 정보를 나타낸다. Member class는 회원의 id와 비밀번호, 이름, 전화번호, 주소, 마일리지를 포함한다.

|  |  |  |
| --- | --- | --- |
| Data | Type | update |
| ID | Variable string | Update\_id |
| Password | Variable string | Update\_password |
| Name | Variable string | Update\_name |
| Phone number | Variable string | Update\_phoneNum |
| Address | Variable string | Update\_address |
| mileage | 10 characters (Numeric) | Update\_mileage |

Get\_id => instance 의 primary key(ID)를 반환한다.

Get\_size => instance의 절대 size를 반환한다.

Pack/Unpack => .dat에 쓰고 읽기를 위함.

* **NewsAgency**

NewsAgency class의 각 instance는 하나의 언론사 정보를 나타낸다. NewsAgency class는 언론사 고유 Id, 언론사 이름, 언론사 주소를 포함한다.

|  |  |  |
| --- | --- | --- |
| Data | Type | update |
| NewsAgency ID | 12 characters (Numeric) | Update\_newsAgencyId |
| Name | Variable string | Update\_name |
| address | Variable string | Update\_address |

Get\_id => instance 의 primary key(NewsAgency ID)를 반환한다.

Get\_size => instance의 절대 size를 반환한다.

Pack/Unpack => .dat에 쓰고 읽기를 위함.

* Subscription

Subscription class의 각 instance는 하나의 언론사 구독 정보를 나타낸다. Subscription class는 구독 고유 ID, 뉴스 언론사 고유 ID, 구독하는 회원의 ID, 추가된 마일리지를 포함한다.

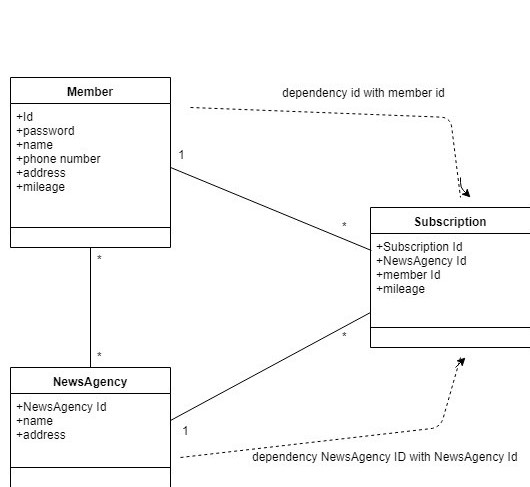
|  |  |  |
| --- | --- | --- |
| Data | Type | update |
| Subscription ID | 16 characters (Numeric) | Update\_subscriptionId |
| NewsAgecny ID | 12 characters (Numeric) | Update\_newsAgencyId |
| Member ID | Variable string | Update\_memberId |
| Mileage | 10 characters (Numeric) | Update\_mileage |

Get\_id => instance 의 primary key(Subscription ID)를 반환한다. Argument가 있는데 적절한 값을 주면 member id, newsagency id도 뽑아낼 수 있다.

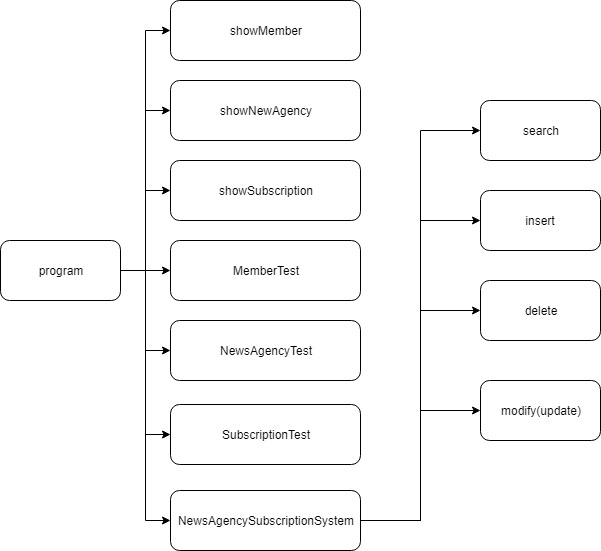
Get\_size => instance의 절대 size를 반환한다.

Pack/Unpack => .dat에 쓰고 읽기를 위함.

1. **클래스 다이어그램**

****

1. **Flow chart**



1. **함수 설명**

**showMember/showNewAgency/showSubscription**

* 각 class에 맞는 txt 파일을 연 다음, overriding 한 >> , << 연산을 통하여 읽고 화면에 출력해준다. 이 때 맨 앞에 저장되어 있는 10개의 데이터만 출력해준다. 이 함수들로 인해 class가 적절히 생성되었고, 읽고 쓰는 것이 잘 된다는 사실을 확인할 수 있다.

**MemberTest/NewsAgencyTest/SubscriptionTest**

* 각 class에 맞는 txt 파일을 열고 각 class에 맞는 .dat 파일을 생성한다. 그 후 txt 파일에서 한 레코드씩 읽고 dat 파일에 pack 하여 작성한다. 그렇게 dat 파일을 만든 이후 필요하다면 10개의 레코드를 보여준다. 이 함수로 인해 데이터 파일의 모든 레코드들이 적절하게 저장되었다는 사실을 볼 수 있다.

**NewsAgencySubscriptionSystem**

* 파일로부터 레코드를 삭제하고 파일에 있는 레코드를 수정할 수 있는 함수를 추가한다. 각 class에 대해 검색, 삽입, 삭제, 수정을 지원할 수 있는 대화식 프로그램이다. Primary key인 id를 제외한 모든 필드를 수정할 수 있고, 모든 경우에서 참조 무결성을 유지하게끔 프로그램을 작성해 주었다.
* **Search**

사용자로부터 id를 input받아서 그 id가 있는지 확인하고, 있다면 record를 사용자에게 출력해준다. Subscription data에서 search할 경우에는 subscription id뿐만 아니라 member id, newsagency id를 통한 검색도 추가해 주었다.

이후 insert, delete, modify 에서도 이 검색을 사용하기 때문에 재사용 가능하게끔 함수화 시켜놓았다

**방법** : index등을 사용하지 않았기 때문에 dat 파일에서 읽어와 하나하나 비교하는 방법을 채택하였다.

* **Insert**

사용자로부터 id를 input 받아서 그 id가 있는지 search를 통해 확인하고, 이미 있다면 사용자에게 출력, 없다면 id 말고 다른 field값들을 받아서 새롭게 추가해 주었다. Field 형식에 맞지 않게 데이터가 들어오면 형식이 맞을 때까지 다시 입력 받았다.

**방법** : search 함수를 활용해 검색하고, 찾지 못했다면 새로운 레코드를 만들어 dat파일 마지막에 붙여 주었다.

* **Delete**

사용자로부터 id를 input받아 그 id가 있는지 확인하고, 없다면 사용자에게 없다는 message를 출력해 주었다. 만약에 있다면 그 파일에서 해당 레코드를 삭제해 주었고, 참조 무결성을 유지하기 위하여 member, newsagency의 id가 삭제된다면 subscription 데이터에서 해당 id를 가진 레코드들도 동일하게 삭제해 주었다.

**Delete compaction + 방법** : search 함수를 활용해 검색하고, 찾았다면 dat file을 모두 읽으며 메모리상에 저장하다가 삭제할 record는 저장하지 않고 넘어간다. 그 후 dat file을 다시 생성해 메모리상에 저장해둔 데이터를 다시 작성해 준다. 추후 삽입 시에는 record의 맨 마지막에 삽입해 주었다.

이후 index등이 추가된다면 이 방식을 더 좋은 방법으로 개선하여야 할 것이다.

* **Modify**

사용자로부터 id를 input받아 그 id가 있는지 확인하고, 없다면 사용자에게 없다는 message를 출력해 주었다. 만약 있다면 그 class에 해당하는 나머지 field값들을 입력 받고 dat 파일에 다시 써 주었다.

**방법** : insert와 delete를 섞어 놓은 방식으로 구현하였다. 먼저 해당 class의 instance 하나를 새로 만들어 id와 다른 field값들을 저장해준다. 그 후 delete 함수를 통하여 해당 id를 dat 파일에서 삭제해 준 후, 저장한 새로운 instance를 dat 파일에 새로 작성해 주었다.

1. **6장 연습문제**

* **21**

처음 진행할 때에는 삭제된 레코드 앞에 특별한 문자 (‘\*’) 을 정하여 붙이고, 읽지 않게 처리했다. 그러나 추후에 가변길이 레코드를 처리할 때, 삭제할 레코드를 뺀 나머지 레코드를 새롭게 저장하는 방식으로 했다. 그렇게 함으로써 중간에 빈 메모리가 남지 않게끔 해 주었고, 결과적으로 shift 한 것처럼 보이게 되었다.

* **22**

삭제된 레코드를 리스트로 만들어 관리하였다. 이후에 레코드가 삽입될 때에는 이 리스트에서 적당한 공간이 있는지 확인 후 있다면 그곳에 삽입하고, 존재하지 않으면 맨 뒤에 삽입하였다. 다만 이후에 가변길이 레코드를 처리할 때 단편화가 발생하기 때문에 최종적으로는 삭제한 공간에 뒷 데이터를 shift를 하는 방식으로 구현했다.

* **23**

가변길이 레코드 처리 시 \*과 같은 문자를 추가하여 처리하면, 매번 해당 레코드의 전체 길이를 구해야 한다. 그리고 index로 정렬도 되지 않았으므로 어차피 검색하려면 순차적으로 탐색을 해야 하기 때문에 삭제한 레코드의 공간을 낭비없이 사용하기 위해 shift를 통해 중간에 남는 공간을 없애고, 추가되는 것은 뒤에 append 하는 방식을 이용했다. 만약 나중에 index가 추가되면 수정해야 할 것으로 보인다.

* **24**

가변길이 레코드 delete 시 남는 공간이 존재하지 않도록 shift 해서 구현하였기 때문에 append 수정할 필요가 없다. 하지만 Index가 추가된다면 정렬 되어있기 때문에 함부로 데이터를 delete 할 때 shift하면 안된다. 그러므로 delete 연산을 바꾸어야 하고, 그렇다면 자연스럽게 append 연산도 변경하여야 한다.

* **25**

현재 shift를 통하여 delete를 수행하였으므로 내부 fragment에 대해선 생각할 필요가 없다. 그리고 variable length string을 입력할 때 원래의 크기보다 작든 크든 data compaction과 append에 대해서 문제가 발생하지 않는다.