



비디오대여 프로그램 계획서

과 목	소프트웨어공학
담당 교수	권 세 진
학 번	201720970
학 과	소프트웨어·미디어·산업공학부
이 름	권 대 한

목 차

1. 비디오 대여 프로그램이란?

- 1-1) 전체적인 프로그램 구성 계획
- 1-2) 기능 구현에 대한 계획

2. 논리 구성도

- 2-1) 전체적인 프로그램 구성도
- 2-2) 프로그램 각 기능 구성도

3. 프로그램 실제 작성

- 3-1) 함수의 중요부분 설명

4. 프로그램 실행 화면

- 4-1) 개발된 프로그램의 문제점, 원인분석

5. 프로그램 유지 보수

- 5-1) 예외 처리문

1. 비디오 대여 프로그램이란?

프로그램의 사용자층 목표는 비디오 가게 주인이 수기로 작성한 비디오, 사용자, 대여 목록보다 사용하기 편해야 하며, 데이터 저장 후 2번째로 프로그램을 구동시켰을 때, 프로그램이 현재 보유 중인 비디오를 보여주면서 동시에 관리도 가능해야 하며, 실제 비디오 가게에서의 사용 환경에서도 문제가 없어야 한다고 생각했으며, 비디오 가게에서 이루어지는 아날로그적인 과정을 구현하기 위해, 프로그램의 기능 구현 부에 충실해야 한다고 생각해서, 비디오를 활용해서 사용할 수 있는 기능을 중점적으로 작성하였고, 코드 재사용을 중점으로 작성하였다.

- 전체적인 프로그램 구성 계획

먼저 비디오 가게에서 이루어지는 과정을 생각해 보았으며, 결국 필요한 기능은 각 사용자, 비디오의 추가, 검색, 수정, 제거, 출력, 마지막으로 대여, 반납, 저장 기능이라고 생각했으며, 실제로 이루어질 것이라고 생각했다. 프로그램 사용자가 실제 종이 형태의 List를 작성하는 것과 같이 프로그램에서의 데이터 작성 이후 항상 데이터가 최신으로 저장되어야 한다고 생각했다. 즉, 종료 전까지 작업했던 데이터베이스에 특별한 처리가 없다면, 휘발성 메모리인 램에 저장되기에, 데이터의 손실을 방지하기 위해 최소 한 번 프로그램 종료 전 비휘발성 메모리에 데이터를 저장해주는 기능이 존재해야 사용자가 프로그램을 재실행하고, 데이터를 관리하려고 할 때, 종이에서 관리하는 느낌, 프로그램의 목표에 가까워진다고 생각했다.

- 기능 구현에 대한 계획

(1) 프로그램 초기 실행 시

요구 점인 프로그램이 재실행될 때, 사전 저장된 List를 불러오기 위해 데이터를 저장할 공간을 선언하며, 파일 입력 스트림으로 저장된 데이터베이스의 정보를 가져오게 할 것이며, 로드가 완료되었다면, 사용자에게 기능을 입력받는 함수를 호출시켜 프로그램의 종료 전까지 사용자에게 받은 숫자 입력으로 원하는 기능을 사용하도록 유도할 것이다.

(2) 사전 저장된 List를 불러오려고 한다면

하나의 List에 사용자, 비디오, 대여 정보를 기록하는 것이 가능한 하지만, 비디오와 사용자가 늘어날수록 동시에 List를 색인하는 횟수가 늘어나면서 연산속도가 상당히 느려지기에, List를 나눠서 데이터를 저장하기로 하였다. 저장된 데이터는 파일 입력 스트림을 통해, 각 List에 로드될 것이며, 이를 통해 사용자가 프로그램을 켜면 항상 최신 DB를 유지한다는 요구 점을 충족할 것이다.

(3) 관리자가 사용자, 비디오를 추가하려고 한다면

관리자는 List에 이름, 휴대폰 번호를 입력해서 사용자를 추가하며, 이름 순으로 정렬해 데이터 가독성을 높일 것이다. 비디오를 추가할 때 비디오의 이름만 입력받아 데이터에 추가하며, 비디오 가게에서 한 종류의 비디오를 한 개씩만 보유하는 경우는 거의 없기에, 동시에 Random Index를 추가해 같은 이름을 가진 비디오를 구분한다. 사람의 이름 또한 Index를 통해 동명이인, 데이터 구분을 하려고 했는데, 사람마다 고유한 전화번호를 가지고 있으므로, Random Index를 사용하지 않도록 결정하였다.

(4) 관리자가 사용자, 비디오 정보를 검색하려고 한다면

List내에 저장되는 데이터가 각자 다르기 때문에, 예외 상황에 대처하기 위해서 List의 개수만큼 검색 함수를 만들어둔다. 그리고 검색, 수정, 제거, 대여, 반납함수에서 실질적으로 작동을 원하는 인덱스의 일부를 사용자에게 입력받아 검색함수를 통해 해당 지점을 찾아내고, 모든 작업이 실행되기에, 객체 검색 함수를 분리해서 작성하였다. 사용자에게 데이터를 입력받고 데이터가 구체화되어 있는 Class 내부의 재정의된 연산자를 통해 해당 데이터와 일치할 때의 데이터를 도출해서 사용자에게 이름, 전화번호, 고유 인덱스, 대여상태를 보여주게 되며, 검색을 마치게 되는데, 위와 같이 모든 기능을 호출만 하는 함수와 연산하는 함수를 분리해서 작성해, 추가를 제외한 모든 함수에서 사용가능해, 코드의 재사용률을 높임.

(5) 관리자가 사용자, 비디오 정보를 수정하려고 한다면

수정함수가 사용자를 수정하기 원한다면, 이름, 전화번호, 대여 상태(현 Account 상태)를 수정할 것인지 입력받고, 비디오를 원한다면, 이름, 대여상태 중 무엇을 수정할 것인지 입력받는다. 위 과정이 끝나고 객체 검색 함수를 통해 데이터의 인덱스 색인을 진행하며, 만약 같은 데이터가 여러 개 존재한다면, 검색 결과를 출력시켜 수정하고자 하는 데이터를 선택하게 하며, 검색 연산 함수가 도출한 인덱스 값을 참조해서 사용자가 원하는 값으로 수정하게 한다.

(6) 관리자가 사용자, 비디오 정보를 제거하려고 한다면

사용자, 비디오 List 중 제거할 데이터를 선택하게 하며, 제거할 인덱스를 판별하기 위해 사용자의 이름, 비디오의 이름을 입력받아 검색기능 함수를 통해 해당 인덱스를 찾아내며, 해당 인덱스를 제거하고 제거함수가 종료되는 것이 일반적이다.

그러나 이는 실물을 대여하고, 관리하는 일이기에, 항상 정상적으로 재고, 사용자가 제거될 일이 적다. 고객이 비디오를 파손시켜 폐기하는 상황에서 관리자가 비디오를 제거하려고 한다면, 비디오는 제거되겠지만, 사용자, 대여 List내에 대여 정보가 존재하기에, 데이터 보장성이 떨어진다. 언제나 이런 케이스가 발생할 수 있기에 제거하려는 정보를 입력했을 때, 데이터가 대여 List에 존재하는지 확인해야 하며 존재한다면, 비디오, 대여자, 대여 List의 대여상태를 변경시키고 종료시켜야 한다. 반대로 대여 중인 비디오가 있는데, 사용자를 제거하려고 하는 경우가 있을 수 있으므로, 제거함수가 실행되면 대여 List의 사용자를 색인해서 비디오의 인덱스를 확인하고, 모두 반납처리 후에 사용자를 제거한다.

(7) 관리자가 사용자, 비디오를 출력하려고 한다면

- 전체 사용자, 비디오가 얼마나 있는지 알기 위해

관리자가 전체 사용자를 파악하기 위해서는 출력 기능함수가 실행될 때, 사용자가 저장된 List에서 사용자의 이름, 전화번호, 대여상태를 불러오면서 사용자가 몇 명인지, 대여 중인 사람은 몇 명인지 표시해줄 것이다.

- 사용자, 비디오의 대여상태를 알기 위해 // 검색함수 내에서 처리

관리자는 비디오 가게를 운영하는 견해에서, 언젠가 특정 비디오를 찾을 일이 있어 재고를 검색해야 할 일이 있을 것이며, 사용자가 대여 중인지 확인할 수 있어야 하기에, 각 List에 정의된 대여상태 인덱스를 통해 이 사용자가 현재 대출 중인지, 연체 중인지 파악하고, 대여 중인 비디오를 관리자에게 보여주며, 비디오 관리를 더 편하게 할 것이며, 비디오가 검색되었을 때 각 비디오의 고유 인덱스, 대여상태, 대여자를 보여주면서 비디오의 현 상태를 관리자가 파악할 수 있게 할 것이다.

- 비디오의 재고가 얼마나 있는지 알기 위해

관리자가 비디오의 재고를 출력하는 함수를 호출할 때, 비디오 List에 저장된 비디오의 이름, 비디오의 고유 인덱스(Random), 대여상태를 출력하며, 총 비디오의 개수, 대여 가능한 비디오의 개수를 관리자에게 보여주면서 비디오 관리를 도와줄 것이다.

(8) 비디오를 대여하려고 한다면

대여하고자 하는 비디오를 객체 검색 함수를 이용해 찾아내고, 대여상태 인덱스를 참조해 만약 재고가 있다면, 대여자의 정보를 입력받은 후, 대여상태를 변경하고, 대여 List에 대여자, 비디오, 대여자 인덱스, 비디오 인덱스, 대여일, 반납일을 추가하며, 대여 함수가 종료된다.

(9) 비디오를 반납하려고 한다면

반납하고자 하는 비디오 이름을 입력받아, 해당 인덱스를 객체 검색 함수를 통해 대여 중인 비디오의 고유 인덱스를 받아오며, 대여 List를 참조해 사용자 List, 비디오 List의 인덱스를 찾아내고, 대여상태를 초기화한다. 마지막으로 대여 List의 정보를 삭제하며 반납이 완료된다.

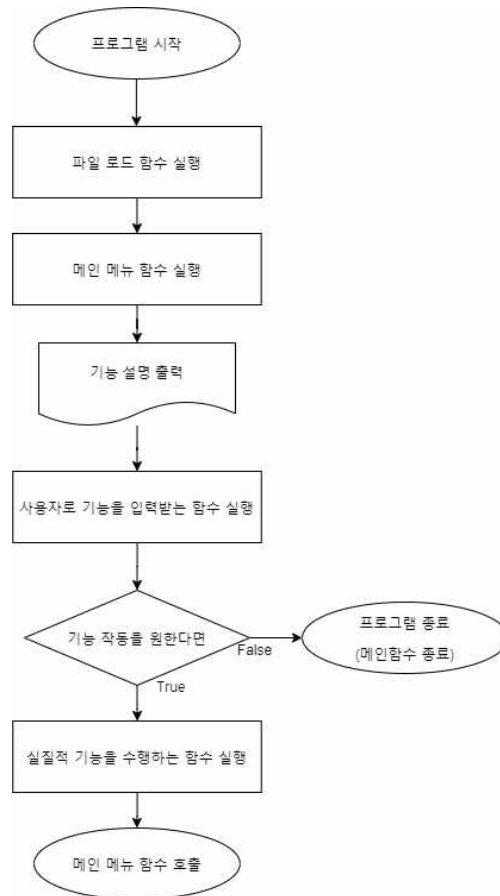
(10) 데이터베이스를 저장하는 메커니즘 (자동 저장)

프로그램 실행할 때 자동으로 데이터베이스를 불러오는 것이 요구 점이기 때문에, 관리자는 데이터베이스에 큰 신경을 쓰지 않을 것이다.

그래서 매 기능 함수가 완료될 때 저장 함수를 삽입해서, 프로그램 사용자는 비디오 관리에만 신경 쓰게 할 것이다. 각 List가 비어있지 않다면 List를 파일 출력 스트림에 보내서 텍스트 파일 형태로 저장할 것이다.

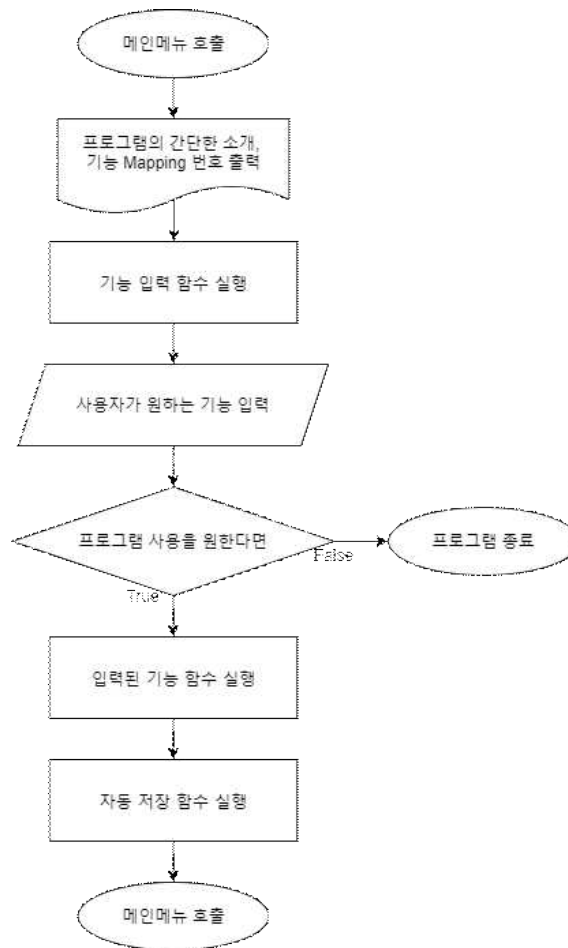
2. 논리 구성도

<프로그램의 상세한 개요 및 시작 메커니즘의 설명>



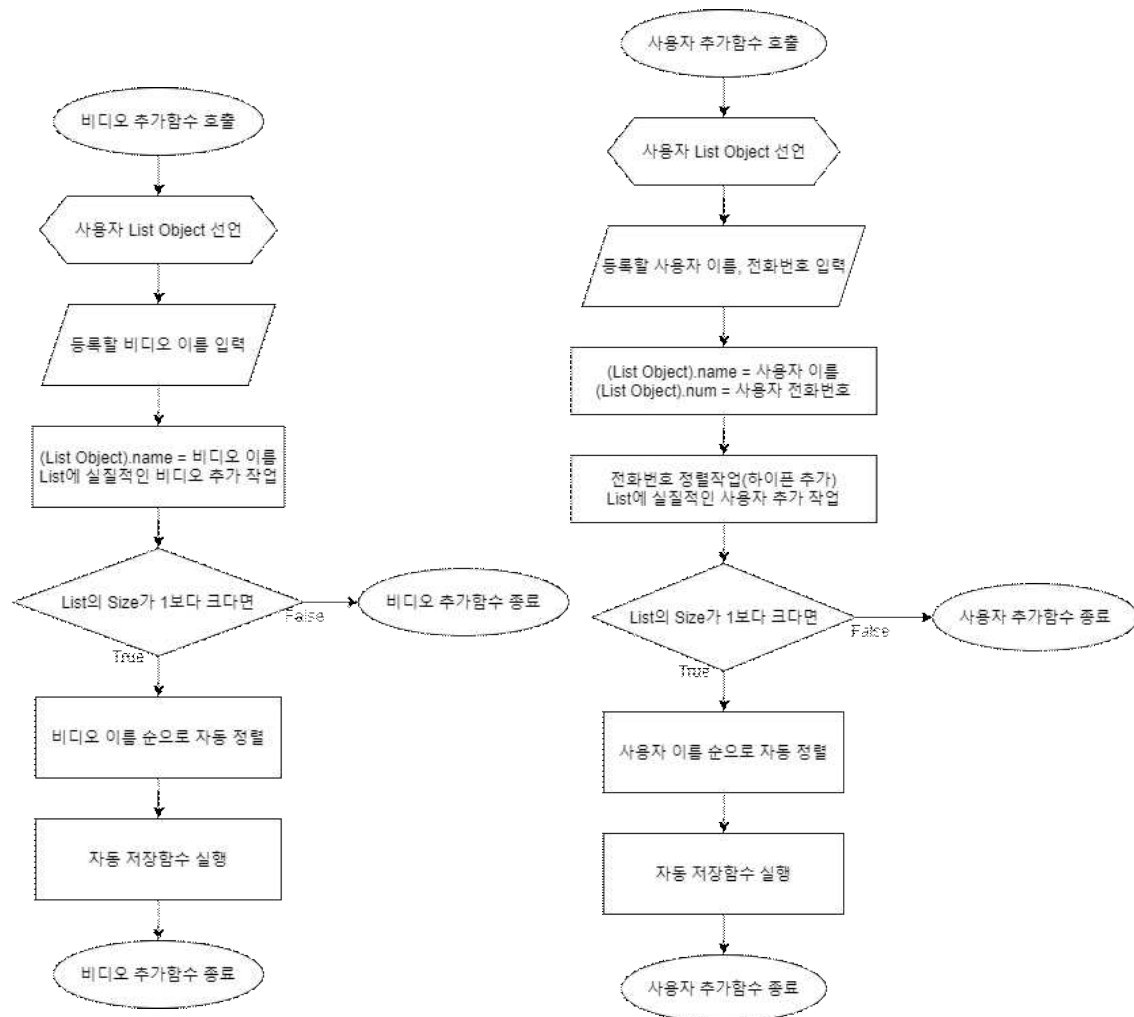
이번 비디오 대여 프로그램에는 요구 점이 2가지가 있다. 첫 번째는 고객의 정보를 제한 없이 저장할 수 있는 것, 프로그램의 종료 후 재실행 시 항상 최신 DB에서 작업하는 것이다. 이 두 가지를 지키기 위해, 본래는 순차적으로 저장될 필요가 없는 것 같아 HashMap의 구조를 가진 List를 이용해 사용자, 비디오, 대여 정보를 저장하려고 했으나, 추후 정렬을 하고자 하면 출력하면서 정렬연산을 해야 하며, 데이터 Access 시 정해진 Key로만 접근할 수 있다는 것이 장점이자 큰 단점으로 다가와서 결국 동적배열인 Vector를 이용해 List를 구성하도록 했으며, 전화번호부 프로그램에서는 이름, 전화번호만 저장해서, List 1개로 모두 구현이 가능했지만, 이번에는 사용자, 비디오, 대여 List 3개가 상호 작용하면서 각자의 기능을 보장하는 방식으로 프로그램을 작성하였다. 그리고 프로그램이 시작되면, 프로그램이 같은 비휘발성 메모리의 디렉토리에 저장된 파일 3개를 읽어오면서, 기존에 저장된 DB를 프로그램의 시작과 동시에 모두 불러오게 되며, 사용자는 프로그램의 사용에만 집중하도록 구현하였다. 이후 프로그램이 가진 기능함수가 실행되며, 사용하고자 하는 기능을 선택해서 사용만 하면 된다. 실질적인 기능을 담당하는 함수들이 끝나면, 자동 저장 함수를 실행시켜 데이터의 무결성을 보장하였으며, 프로그램의 사용을 종료하고자 한다면 원할 때 프로그램의 사용을 중단할 수 있다.

<메인 메뉴 함수의 구현>



파일로부터 리스트의 데이터를 모두 불러오면, 메인 메뉴 함수가 호출된다. 메인 메뉴 함수는 사용자에게 입력을 받아 실질적인 기능을 실행시켜주는 중계 함수라고도 볼 수 있다. 호출 시 사용자에게 프로그램에 구현된 기능에 관해서 설명하고, 사용자에게 기능을 입력받는 함수를 실행시켜, 사용자에게 입력된 번호 기반으로 해당 기능을 실행시키며, 해당 기능이 모두 할 일을 마치고 종료된 후 바로 저장 함수가 실행되면서, 프로그램이 가지고 있는 사용자 정보, 비디오 정보, 대여 정보가 3개를 자동 저장한다. 이로 인해 사용자가 저장을 하지 않고 프로그램을 종료시키거나, 혹은 데이터 저장에 대한 개념이 없는 사용자에게 이 기능은 생산성을 매우 높여준다. 그리고 실제로 콘솔로 보이는 것보다 컴퓨터의 처리 성능은 매우 뛰어나기 때문에 매번 각 리스트 저장함수를 매번 호출하여도 순식간에 데이터 저장이 완료된다. 그리고 사용자는 항상 프로그램을 사용할 때마다 무결성이 보장된 데이터에 접근하고, 수정할 수 있다. 저장이 완료되고 사용자에게 프로그램에 대한 기능을 설명하며, 이때 사용자가 프로그램의 사용을 중단하고 싶어 한다면, 종료 기능을 입력한다면, 메인 메뉴 함수가 종료되고, 프로그램의 종료로 이어진다.

<비디오, 사용자를 추가하려고 한다면?>

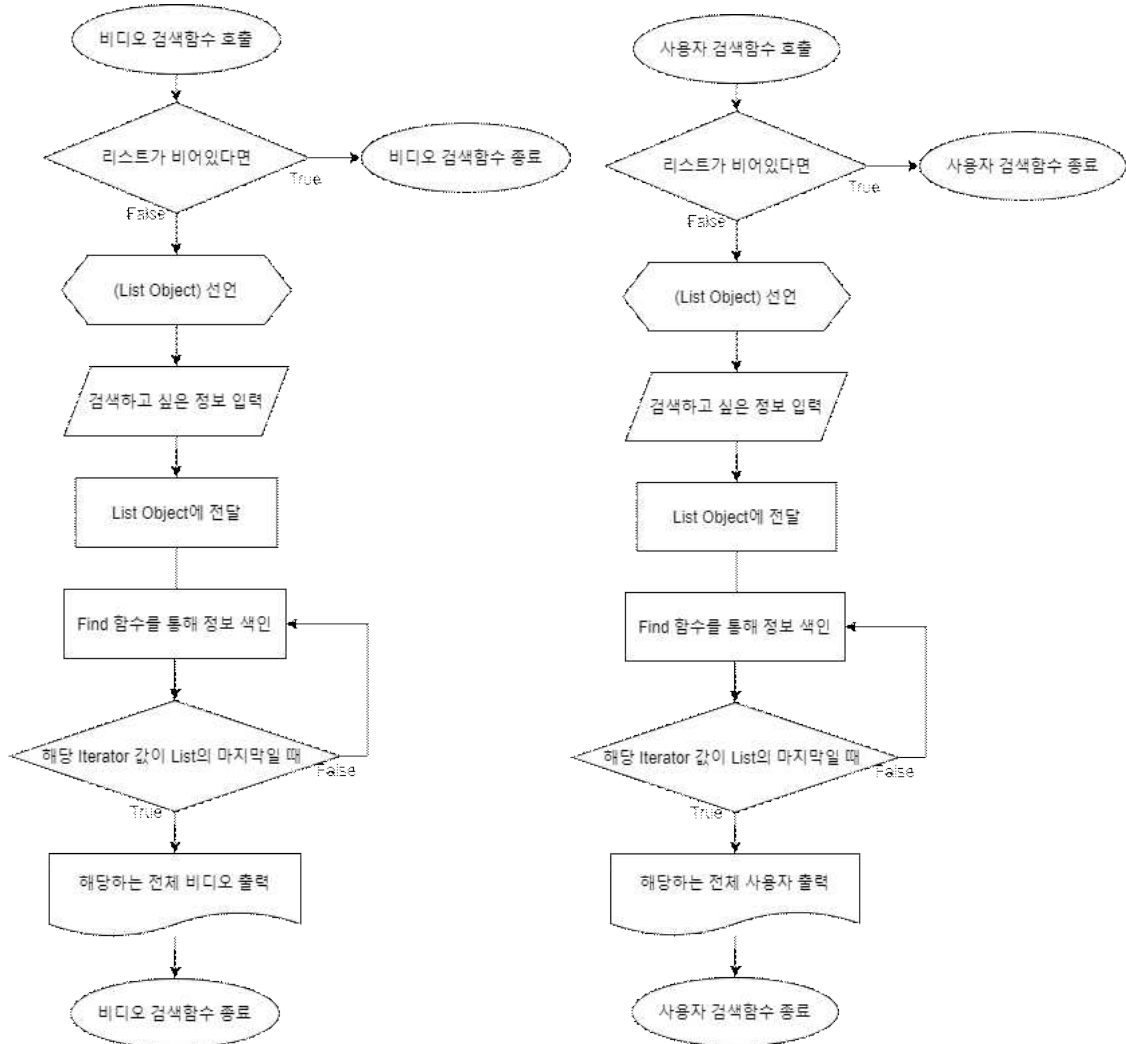


비디오 추가함수가 호출될 때는 등록할 비디오의 이름만 입력받으며, 추가함수 내부의 임시객체를 선언하면서 고유 인덱스, 대역상태가 자동 추가되며, 이 객체에 비디오 이름을 입력받은 후 비디오 List에 넣고 종료된다.

만약 데이터가 처음 추가되었다면, 그대로 추가기능을 마치며, 데이터가 두 번째로 추가된다면, Bool Type의 비교식을 기반으로 비디오를 이름 내림차순으로 정렬한다. 추가함수가 종료되면 자동 저장 함수가 실행되고 메인 메뉴 함수가 호출된다.

사용자 추가 함수도 같다. 비디오 List와 같은 Class로 구체화했으며, 들어가는 데이터만 다르므로, 전체적인 추가 메커니즘은 같지만, 추가되면서 이름과 휴대폰 번호를 입력받아 사용자 List에 객체를 저장하며, 만약 휴대폰 번호 추가 시 하이픈이 없다면, 하이픈을 추가해서 데이터의 정렬을 보장하며, 비디오 List 추가와 똑같이 프로그램 종료 전 두 번 이상 데이터 추가가 되었을 때, 데이터 정렬을 하고 사용자 추가 함수를 마친다.

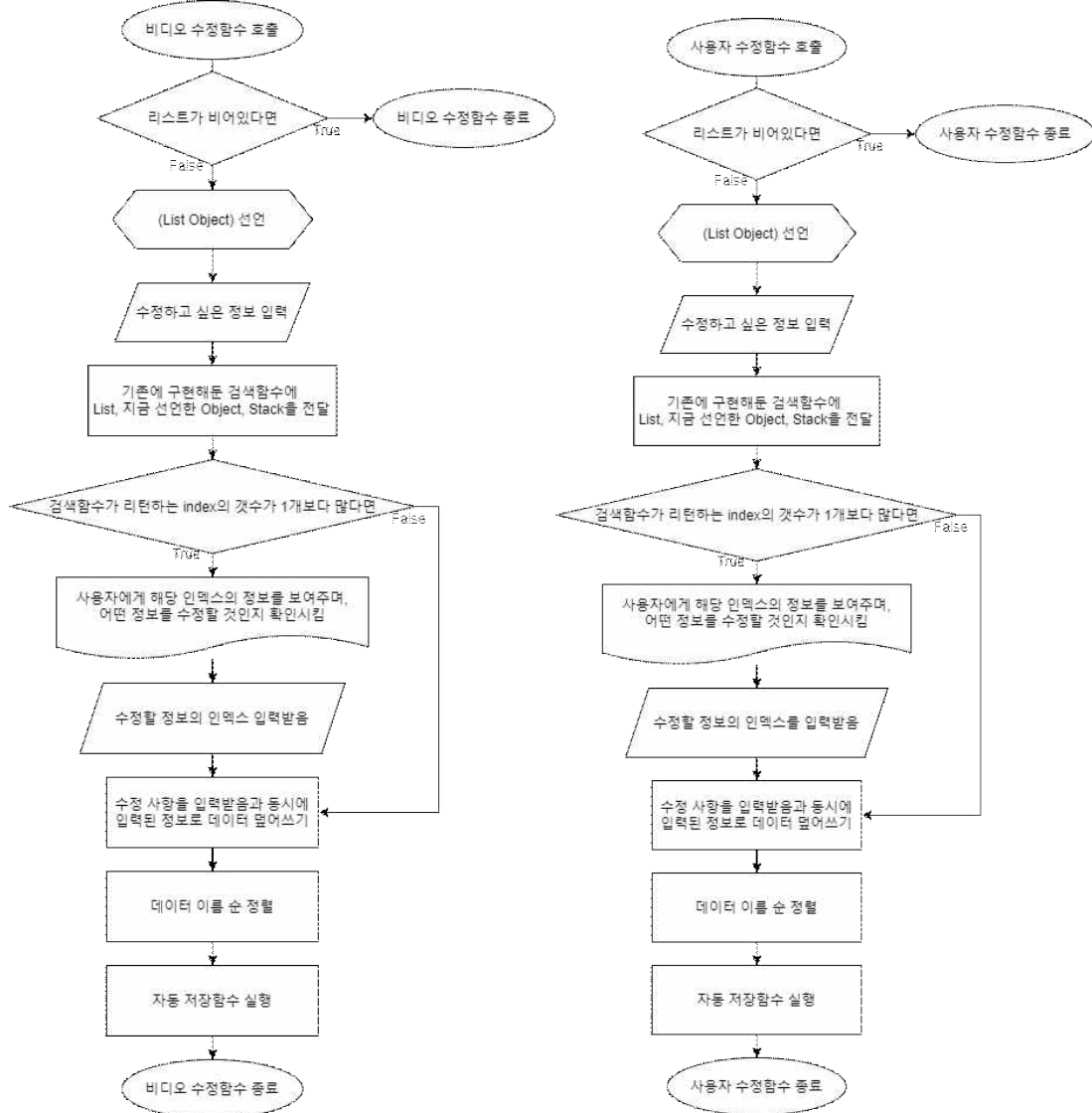
<비디오, 사용자 정보를 검색하려고 한다면>



데이터 검색에 들어가기 전에, 먼저 해당 List에 데이터가 들어있는지 확인한다. 데이터가 있다면 객체 검색 함수를 통해 해당 인덱스를 찾아내기 위해 데이터를 입력받을 임시 객체, 인덱스를 저장할 Stack을 선언하고, 사용자가 검색하고자 하는 데이터를 선택하게 하며 (이름, 전화번호, 대여상태), 임시 객체에 데이터를 입력받고, Class 내부에서 미리 재정의한 == 연산을 데이터가 존재함을 기반으로 모든 데이터를 비교하도록 선언해두었기 때문에, Stdlib의 Find 함수를 사용해 List와 동일하다면 출력시키고, 임시로 인덱스를 저장한다. 이후 List의 마지막이 될 때까지 색인하며, 검색함수가 종료되게 된다.

해당 인덱스를 임시 저장하는 데에는 이만한 방법이 없다고 생각했기 때문에 인자로 List, 임시 객체, Stack을 주솟값으로 전달받아 실행된다. 그러나 프로그램 작성하면서 느낀 것인데 Stack으로 Index를 받게 되면 상당히 큰 단점이 있다. 마지막 결과 분석에서 설명하겠다.

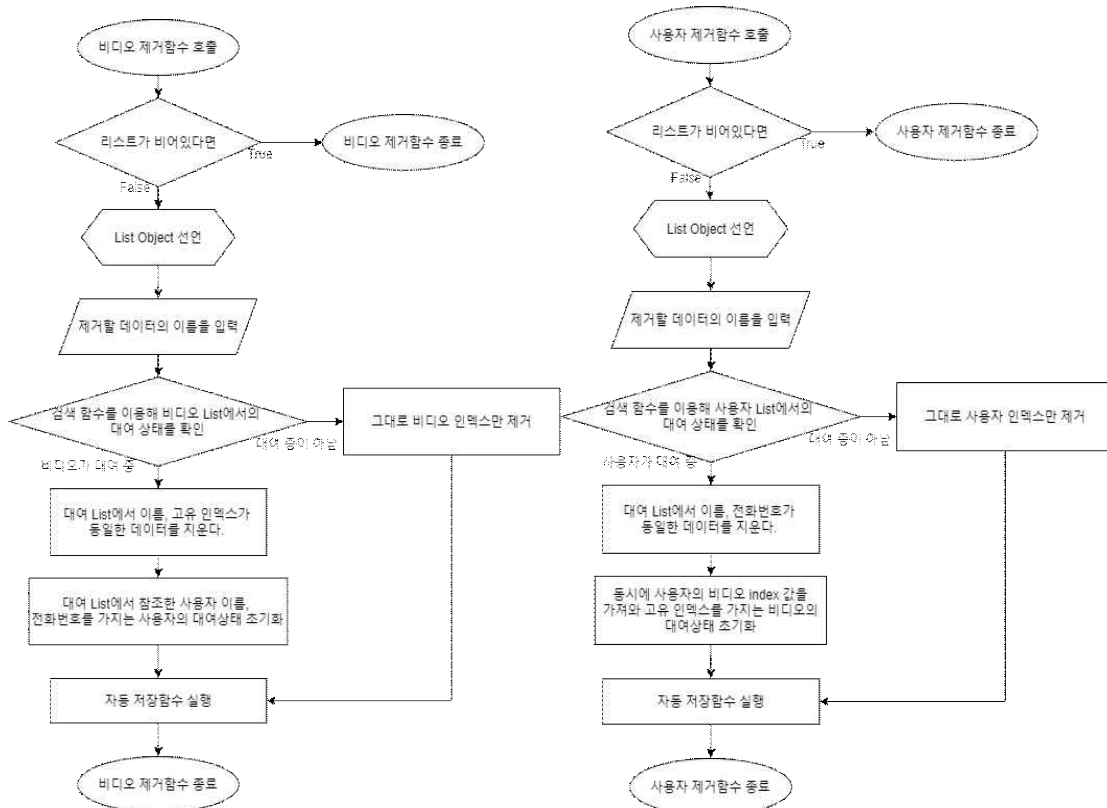
<비디오, 사용자 수정하려고 한다면?>



기본적으로 프로그램 내에서 기존의 데이터 기반 접근하는 함수가 실행된다면, 불필요한 데이터 색인을 방지하기 위해, 먼저 List에 데이터가 있는지 확인한다.

데이터가 존재한다면 수정함수가 실행되는데, 데이터의 색인을 위해 임시 객체와 인덱스의 임시 저장소 Stack을 선언하고, 사용자에게 수정하고 싶은 정보를 입력받고, 이에 따라 다른 각 다른 데이터를 임시 객체에 입력받음과 동시에, 객체 검색 함수를 이용해 List와 객체 내 데이터가 같은지 List를 색인하며, 만약 해당 데이터를 가진 인덱스(Stack.size())가 2개 이상으로 검출된다면, 사용자에게 검색 결과를 보여주며, 수정할 데이터의 인덱스, 수정할 데이터를 입력받아 List의 데이터를 수정, 정렬하고, 자동 저장하면서 수정함수는 종료된다.

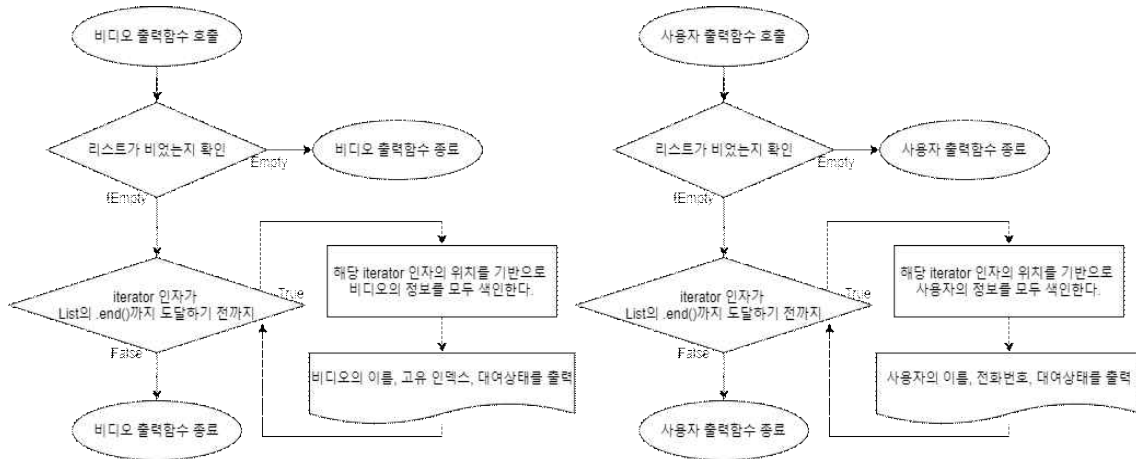
<비디오, 사용자 제거하려고 한다면?>



데이터를 삭제하려고 한다면, 결국 데이터의 해당 위치를 알아야 하므로, 객체 검색함수를 사용해 제거함수를 작성했다. 객체 검색함수를 작동시키기 위해, List와 같은 Type의 데이터의 임시 객체를 선언하고, 임시로 인덱스를 저장할 Stack을 선언했다. 사용자로부터 제거할 데이터의 이름을 입력받아 임시 객체에 저장하면서, 제거함수에서는 검색함수를 실행시킬 조건을 만들며, List와 해당 객체를 비교하고, 매칭된다면 해당 인덱스를 제거하면 되며, 만약 매칭되는 이름이 없다면 제거함수는 그대로 종료된다. 여기서 비디오 제거함수와 사용자 제거함수의 차이점이 있는데, List와 매칭되는 이름이 존재하고, 비디오가 대여 중이라면(비디오 List의 대여상태) 대여 List의 정보를 지움과 동시에 대여자 정보를 찾아 대여상태를 초기화한다. 이는 결국 대출 중인 비디오를 반납시키고 제거하는 행위와 같다. 그리고 반대로 대여 중이 아니라면, 비디오의 인덱스만 제거되며, 이후 자동 저장되면서 비디오 제거함수가 종료된다.

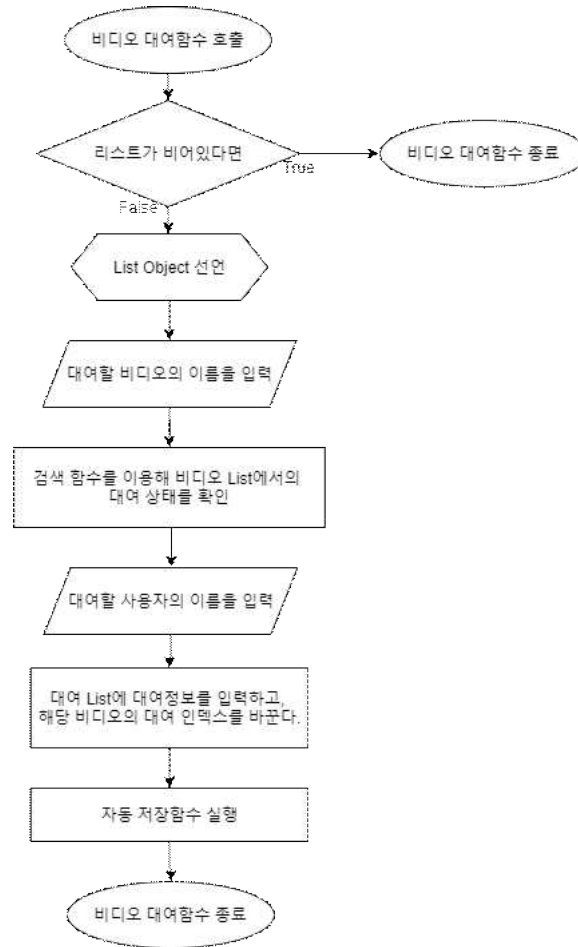
사용자 제거함수에서의 예외 처리는 어떨까? 제거하려는 사용자가 비디오 대여 중이라면(사용자 List의 대여상태) 반납처리를 위해서 대여 List에 있는 대여 정보를 지우고, 대여한 비디오가 저장된 List의 인덱스에서 대여 정보를 초기화하고 사용자를 제거 후 모든 작업이 끝났으면, 자동 저장 함수에 의해 List 저장과 함께 사용자 제거함수가 종료된다.

<사용자, 비디오, 대여List를 출력하려고 한다면?>



출력함수가 호출되었을 때도 결국 이미 있는 데이터에 접근하여 사용자에게 보여주는 것이기에, 제일 먼저 리스트가 비었는지 확인한다. 비었다면 불필요한 연산을 방지하기 위해 함수는 바로 종료되며, 이후 출력함수의 기능을 시작한다. 비디오 List와 사용자 List는 서로 같은 Type으로 구체화되어 있기에, 출력함수를 재사용해 코드의 길이를 줄였으며, 전체의 리스트를 이름, 고유 인덱스 (전화번호), 대여상태를 관리자에게 보여주고, 이후 대여 List의 인덱스 개수를 기반으로 현재 대여 중인 비디오의 개수를 파악할 수 있도록 구성했으며, 대여 List 같은 경우 역시 출력 시 현재 대여 중인 사용자와 비디오를 한 번에 파악할 수 있도록 하였다.

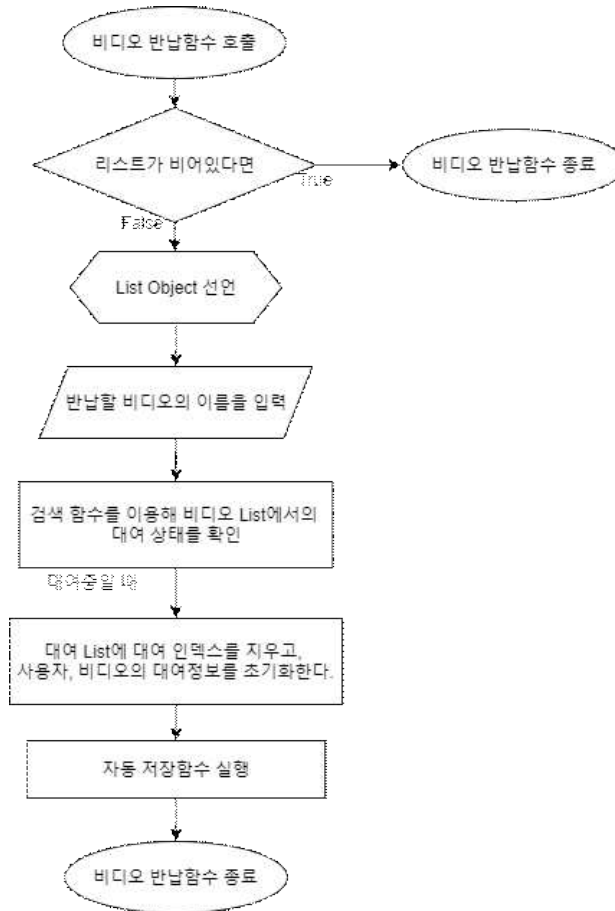
<특정 사용자가 비디오를 대여하려고 한다면?>



비디오 대여 함수가 실행될 때, 비디오 List의 Size를 확인해서 데이터가 존재하는지 확인하고, 데이터가 존재하면, 대여 함수의 기능 실행을 진행한다.

대여하려는 비디오, 사용자 정보를 결국 찾아서 대여 정보를 넣어야 완벽한 대여 처리가 가능하므로, 임시 객체를 선언하면서 객체에 대여할 비디오 이름을 입력받아 객체 검색함수를 이용해, 해당 비디오 인덱스를 찾으며, 모든 비디오가 대여 중이라면, 비디오 대여 함수가 종료된다. 만약 1개의 비디오라도 대여 중이 아니라면, 대여할 사용자의 이름을 입력받아 사용자 List의 대여정보를 대여상태로 바꿔주며, 대여 List Type 임시객체에 이름, 전화번호, 비디오 정보, 대여일, 반납일을 저장함으로 비디오 대여 작업은 끝난다.

<어떤 사용자가 비디오를 반납하려고 한다면?>



비디오 List가 비어있는지 확인함으로, 함수의 오류를 방지하며, 불필요한 연산을 줄인다. 대여 List와 비교를 위해, 사용자에게 반납하고자 하는 비디오의 입력을 받고, 대여 List와 같은 Type인 객체에 저장한다. 객체 검색함수를 활용해, 대여 List의 비디오, 사용자 정보를 Video type, User type 임시 객체에 저장하며, 객체 검색함수를 사용해 해당 비디오, 사용자 List의 어떤 index에 있는지 확인해서 대여 정보를 초기화시키며, 마지막으로 대여 List의 대여 정보를 삭제시키며 반납함수가 종료되게 된다.

3. 프로그램의 실제 작성

위 플로우 차트는 전화번호부 프로그램의 계획서와 다르게 최대한 코드답지 않으며, 간결히 작성하는 것에 집중하였다. 해당 부분에서는 비디오 가게 프로그램이 되기까지의 실질적인 기능들을 어떻게 구현하였는지 설명하도록 하겠다.

<프로그램의 실행, List 선언 부>

```
int main()
{
    vector<Info> videodata;
    vector<Info> userdata;
    vector<R_Info> rentaldata;
    auto_fileload(videodata, userdata, rentaldata);
    menu(videodata, userdata, rentaldata); //메인 가
}
```

프로그램의 요구 점에서 프로그램의 재 실행 때, 항상 최신 DB를 사용할 수 있어야 한다는 점에 따라, 비디오 관리 프로그램이 처음 실행된다면, 비디오, 사용자, 대여 정보를 저장할 List를 선언하며, 파일을 불러오는 함수를 실행해 파일 형태의 데이터를 불러오며, 함수 작동을 마쳤다면, 메인 메뉴 함수를 실행해, 사용자가 원하는 작업을 하게 한다.

<파일 자동 로드 함수의 구현 부>

```
void auto_fileload(vector<Info>& o1, vector<Info>& o2, vector<R_Info>& o3)
{
    ifstream v_ifs, u_ifs, r_ifs;
    v_ifs.open("VideoList.txt");
    u_ifs.open("UserList.txt");
    r_ifs.open("RentalList.txt");
    if (v_ifs.is_open() && u_ifs.is_open() && r_ifs.is_open())
    {
        cout << "파일을 모두 성공적으로 불러왔습니다. " << endl;
        Info T1, T2;
        R_Info T3;
        while (v_ifs >> T1.name >> T1.num >> T1.rental)
```

프로그램과 같은 디렉토리에 있는 VideoList, UserList, RentalList로 3가지의 파일 입력 스트림을 정의하여, 모든 파일이 존재하고, 파일의 끝이 아닐 때까지, 데이터를 토큰 단위로 불러와 List의 변수에 차곡차곡 저장한다. 해당 함수를 통해 파일의 위에서부터, 끝까지 읽으며, vector에 데이터를 추가할 때 뒤에 저장되기에 차곡차곡 저장된다.

<파일 자동 저장 함수의 구현 부>

```
void auto_filesave(vector<Info>& o1, vector<Info>& o2, vector<R_Info>& o3)
{
    system("cls"); //clear console

    ofstream v_ofs, u_ofs, r_ofs;
    v_ofs.open("VideoList.txt");
    u_ofs.open("UserList.txt");
    r_ofs.open("RentalList.txt");
    vector<Info>::reverse_iterator v_iter, u_iter;
    vector<R_Info>::reverse_iterator r_iter;
    if (!o1.empty())
    {
        for (v_iter = o1.rbegin(); v_iter != o1.rend(); v_iter++)
        {
            v_ofs << o1.at(v_iter - o1.rbegin()).name << " " << o1.at(v_iter - o1.rbegin()).num << " "
            << o1.at(v_iter - o1.rbegin()).rental << " " << "\n";
        }
    }
}
```

프로그램의 요구 점에 의해 재실행 때 자동으로 이전에 작업한 데이터를 불러온다. 그러나 과연 사용자는 파일 자동 저장 함수를 통해 파일 로드를 직접 하지 않기에, 저장을 제대로 하지 않을 것으로 생각이 되어 매 기능이 종료되고 List의 데이터를 바로 저장되도록 작성하였다. 저장 함수가 실행되면, 3개의 파일 출력 스트림을 선언해서 각 List를 파일 별로 저장하게 하였고, 이후 List에 데이터가 있을 때, 역순으로 저장하도록 작성하였다.

<메인 메뉴 함수의 구현 부>

<pre>void menu(vector<Info>& o1, vector<Info>& o2, vector<R_Info>& o3) { int check = 0; while (true) // 1 or 2일 경우 다음 함수로 진행 { system("cls"); cout << "관리하고 싶은 데이터를 고르세요 (1 : 비디오 데이터, 2 : 사용자 데이터, 3 : 렌탈 데이터)"; cout << "기능 >> "; cin >> check; if (check != 1 && check != 2 && check != 3 && check != 4) { system("cls"); cout << "다시 입력해주세요" << endl; } else if (check == 1) { v_menu(o1, o2, o3); } } }</pre>	<pre>void v_menu(vector<Info>& o1, vector<Info>& o2, vector<R_Info>& o3) { int check = 0; while (true) { system("cls"); cout << "비디오 데이터를 관리합니다." << endl; cout << "1. 추가, 2. 검색, 3. 수정, 4. 제거, 5. 출력, 6. 대어, 7. 반납, 8. 예약"; cout << "기능 >> "; cin >> check; switch (check) { case 1: v_add(o1); auto_filesave(o1, o2, o3); //auto saving function case 2: v_search(o1, o2); case 3: v_modify(o1, o2); case 4: v_remove(o1); case 5: v_output(o1); case 6: v_rental(o1, o2, o3); case 7: v_return(o1, o2, o3); case 8: v_reservation(o1, o2, o3); default: break; } } }</pre>
--	---

좌 : 메인 메뉴 함수, 우 : 비디오 관리 함수(사용자 관리 함수와 매우 유사한 작동방식)

메인 메뉴 함수는 메인 함수에서 선언한 3개의 List를 인자로 받아오는 방식으로 List에 접근하도록 작성하였으며, 사용자가 사용하고 싶은 기능을 선택받고, 해당 기능과 List를 연결해주면서 실질적으로 메인 메뉴 함수의 기능은 끝이 난다.

그리고 호출된 List 관리 함수에서 기능을 실행한다. List 정보를 파일에 저장하면서 사용자가 선택한 기능의 작동이 끝나게 된다.

<추가함수의 구현 부>

<pre>void v_add(vector<Info>& o1) { system("cls"); Info T1; cout << "비디오 정보를 추가합니다. " << endl; cout << "추가하실 비디오 이름을 입력해주세요." << endl; cin >> T1.name; o1.push_back(T1); if (o1.size() != 1) //1번째 구동이 아니라면 정렬처리 { sort(o1.begin(), o1.end(), name_check); // name } }</pre>	<pre>cout << "사용자의 전화번호를 입력해주세요." << endl; cin >> T1.num; if (T1.num.length() == 11) { T1.num.insert(3, "-"); T1.num.insert(8, "-"); } o1.push_back(T1);</pre>
--	--

좌 : 비디오 추가함수, 우 : 사용자 추가함수의 전화번호 추가 부분

List가 같은 구조로 되어있어서, 비디오와 사용자 추가함수의 내용은 매우 유사하며, 웬만하면 하나의 함수에서 비디오, 사용자 데이터를 관리하고 싶었지만, 실제로 그 데이터가 가지는 의미가 서로 다르므로, 모든 함수를 분리해서 작성하였다.

사용자 추가함수에서 전화번호를 추가하려고 한다면, 하이픈 보정을 마친 객체를 List에 추가해줌으로 함수가 종료된다.

<검색함수의 구현 부>

<pre>void v_search(vector<Info>& o1) { system("cls"); if (o1.empty()) { cout << "비디오 리스트가 비어있습니다." << endl; system("pause"); return; } else { Info object1; stack<int> t_index;</pre>	<pre>template <class T> // <T> IntelliSense에 대한 샘플 템플릿 인수를 제공함 void vu_search(vector<T>& o1, T& T1, stack<int>& T2) { auto it = find(o1.begin(), o1.end(), T1); while (true) { it = find(it, o1.end(), T1); if (it != o1.end()) { T2.push(it - o1.begin()); ++it; } } }</pre>
---	---

좌 : 비디오 검색 호출 함수 우 : 비디오, 사용자, 대여목록 객체 검색함수

사용자가 비디오를 검색하고자 하면, Info Type의 객체를 선언하고 객체에 비디오 이름을 직접 입력받아 저장하며, 추후 vu_search라는 객체 검색함수를 이용해 List 전체를 객체로 색인하고, 발견된 인덱스(Stack)들을 출력함수에 전달해서 사용자에게 검색된 내용을 보여준다.

오른쪽의 객체 검색함수는 비디오와 사용자 데이터만 다룬다면 단일 데이터 타입으로 작성해도 괜찮았겠지만, 대여 List를 색인하고자 하여도 사용할 수 있게 하려고 template Type으로 작성하였다.

<수정함수의 구현 부>

```
stack<int> t_index, t_index1;
cout << "수정하실 비디오 이름을 입력해주세요." << endl;
cin.ignore();
getline(cin, object1.name);
object1.num = ""; // random index clear
vu_search(o1, object1, t_index); //saving index
t_index1 = t_index; // stack issue.. stack을 사용하면 안 된다.
```

```
else
{
    index = t_index.top();
}
//Video List Index를 기반으로 R_Info의 리스트를
cout << "수정하실 이름을 입력해주세요 " << endl;
cin.ignore();
getline(cin, o1.at(index).name);
cout << "이름 수정이 완료되었습니다." << endl;
sort(o1.begin(), o1.end(), name_check);
```

특정 데이터를 수정하려면, 해당 지점을 알아야 수정할 수 있기에, 위에서 구현한 객체 검색함수를 재사용해서 사용자가 수정하고자 하는 데이터의 인덱스를 찾아낸다.

List Type의 객체를 선언해 비디오 이름을 입력받으며, 이후 Object 생성 시 Random한 Index 값을 부여하도록 했기 때문에, 오류 없는 데이터 색인을 위해, num의 Index 값을 초기화하고, 데이터 색인에 들어간다.

색인을 마치고 stack에 객체와 일치하는 해당 index들이 저장되는데, stack이 몇 번째 존재하는 데이터를 직접 접근이 불가능해서 stack을 복사해서 pop을 하며 저장된 인덱스의 데이터를 표시해주며, 사용자에게 입력받은 인덱스의 데이터에 접근하기 위해 pop을 한다. 이는 stack이 무조건 FIFO만 된다는 사실을 간과해서 발생한 문제이다. 뒤에서 작성한 함수들도 작성한 객체 검색함수의 표준에 따라야 하기에, 같은 문제가 계속 반복된다.

데이터 색인을 마치고, 해당 인덱스의 데이터를 수정하고 정렬하면서 수정함수는 종료된다.

<제거함수의 구현 부>

```
stack<int> t_index, t_index1;
cout << "제거하실 비디오 이름을 입력해주세요." << endl;
cin.ignore();
getline(cin, object1.name);
object1.num = ""; // random index clear
vu_search(o1, object1, t_index); //saving index
t_index1 = t_index; // stack issue.. stack을 사용하면
```

```
if (o1.at(index).rental)
{
    cout << "사용자가 대여 중입니다. 반납 후 시도 해 주세요." << endl;
    system("pause");
    return;
}
cout << "사용자 제거가 완료되었습니다." << endl;
o1.erase(o1.begin() + index);
sort(o1.begin(), o1.end(), name_check);
```

List에 저장된 정보를 제거하려고 한다면, 사용자에게 제거하려고 하는 인덱스의 이름을 입력받아, 객체 검색함수를 통해 해당 리스트에 있는 데이터의 인덱스를 받는다.

여기서 stack이 여러 개의 인덱스를 포함하고 있다면, 사용자에게 물어보고 데이터를 지울 준비를 한다. 여기서 검색된 인덱스의 데이터가 이미 대여 중이라면(index > 0 : 대여 중) 제거 작업을 중지하고 사용자에게 반납을 요구하며, 특별한 이상이 없다면 사용자, 비디오를 제거한다.

<출력함수의 구현 부>

<pre>void av_output(vector<Info>& o1, vector<R_Info>& o3) { system("cls"); //clear console if (!o1.empty()) { cout << "전체 비디오 목록을 출력합니다." << endl; int count = 0; v_frame(); for (const auto& iter : o1) //value_type iterator {</pre>	<pre>void v_output(vector<Info>& o1, stack<int>& T1) { int count = 0; while (!T1.empty()) { cout.width(2); cout << ++count; cout.width(12); cout << o1.at(T1.top()).name; cout.width(12); cout << o1.at(T1.top()).num; cout.width(11); if (o1.at(T1.top()).rental == 0)</pre>
---	---

리스트 관리함수에 의해 해당 List의 모든 데이터를 출력하는 요청이 들어왔다면, 함수로부터 입력된 List의 끝에 도달할 때까지 value_type iterator가 for문을 반복하면서, 모든 데이터를 출력한다.

그리고 stack에 저장된 특정 index를 출력하려고 할 때 오른쪽과 같은 함수를 사용하는데, List와 Reference를 받아 stack이 끝날 때까지 List의 데이터를 모두 표시해주는 함수이다.

<대여 함수의 구현 부>

```
cout << "대여하실 비디오 이름을 입력해주세요." << endl;
cin.ignore();
getline(cin, object1.name);

vu_search(o1, object1, t_index); //saving index
t_index1 = t_index;
if (t_index1.size() == 0)
```

```
index1 = t_index2.top();
o1.at(index).rental = 1; //video Only one at same time
o2.at(index1).rental++; //user rental status
object3 = o2.at(index1);
object3 = o1.at(index); //copy of user, video data
o3.push_back(object3);
```

객체를 검색하는 함수를 재사용하기 위해, 대여하고자 하는 것의 이름을 객체로 받았으며, 그 객체와 stack을 객체 검색함수에 전달했다. 그 결과로 출력되는 인덱스 개수가 여러 개라면 사용자에게 어떤 비디오를 빌릴 것인지, 어떤 사용자가 대여하는지 찾아내며, 해당 인덱스가 정해지면, 비디오, 사용자 정보에 대여 중이라는 기록을 하며, 대여 List Type 객체에 해당 정보를 전달하고 되어 List에 정보를 추가하며 대여 함수는 종료된다.

<반납함수의 구현 부>

```
cin.ignore();  
getline(cin, object1.v_name);  
  
r_search(o3, object1, t_index); //saving index  
t_index1 = t_index;
```

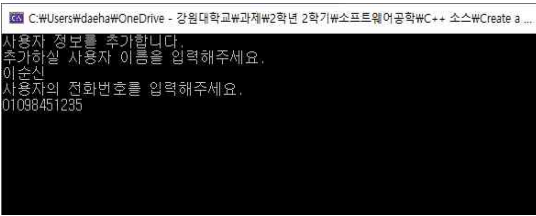
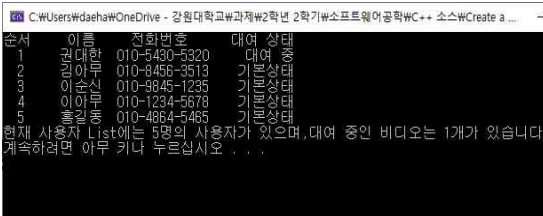
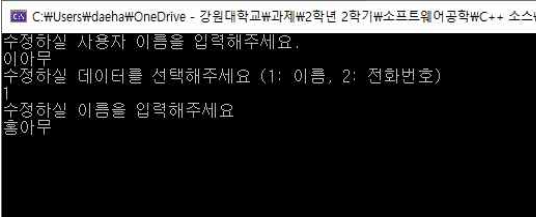
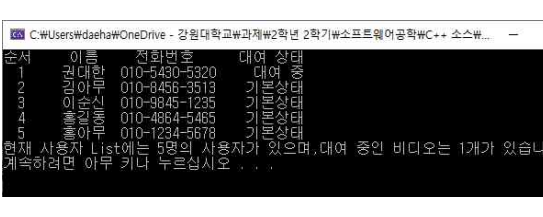
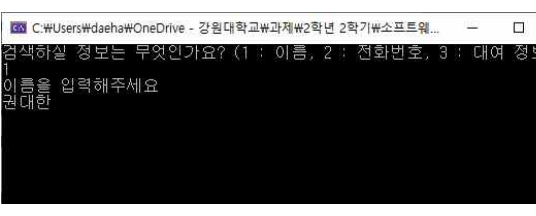
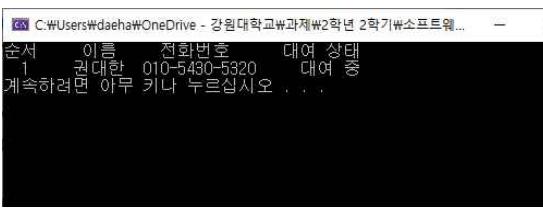
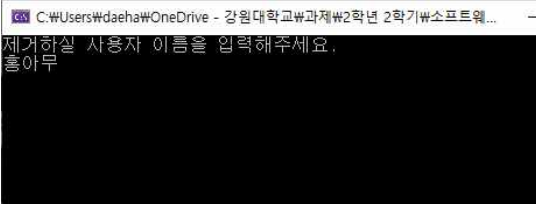
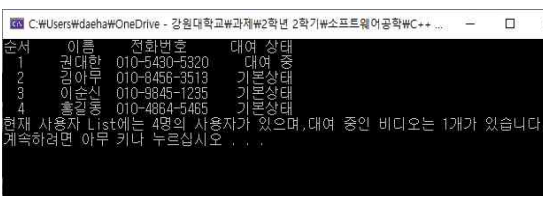
```
//object2 == video, object3 == user  
//video rental info reset  
object2 = o3.at(index); //copy of vid, user info  
object3 = o3.at(index);  
o3.erase(o3.begin() + index); //rental list erase  
  
vu_search(o1, object2, t_index2); //index saving to t_index2  
v_index = t_index2.top();  
vu_search(o2, object3, t_index4); //index saving to t_index4  
u_index = t_index4.top();  
o1.at(v_index).rental = 0; //video Only one at same time  
o2.at(u_index).rental--; //user rental status
```

객체 검색함수를 재사용하기 위해, 반납하고자 하는 비디오 이름을 객체로 받았고, 그 객체와 stack을 객체 검색 함수에 전달했다. stack에 저장된 인덱스 개수가 여러 개라면 사용자에게 어떤 비디오를 반납할 것인지를 알아내고, 대여 List를 참조해서 어떤 사용자가 대여했는지 찾아내고, 비디오, 사용자 정보의 대여상태를 초기화한다. 이후 대여 List의 정보를 지우며 반납함수가 종료된다.

4. 실제 프로그램 실행

<테스트 결과>

	
프로그램 초기 실행 시, 메인 메뉴 함수	1번을 입력해 비디오 관리함수 호출
	
2번을 입력해 사용자 관리함수 호출	비디오 관리함수에서 추가를 호출
	
기존의 DB, 추가가 정상적으로 됐음을 확인	비디오 관리함수에서 수정을 호출
	
수정이 제대로 됐음을 확인	방금 수정한 비디오 제거
	
비디오가 제거됨을 확인	비디오 기능함수에서 대여를 호출
	
비디오 반납함수를 호출, 대여 여부 확인	반납이 제대로 됐음을 확인

	
<p>사용자 관리함수에서 사용자 추가</p>	<p>사용자 List 출력</p>
	
<p>List에 존재하는 Data 수정</p>	<p>Data 변경을 확인 가능</p>
	
<p>사용자 검색함수 호출</p>	<p>사용자 검색 결과</p>
	
<p>사용자 제거함수 호출</p>	<p>사용자 List 출력</p>

<프로그램 개발 후 결과 분석>

이번 프로그램을 개발하면서, 프로그램 내에 많은 것을 담으려고 했기 때문에, 간결한 프로그래밍을 하지 못했다는 생각이 든다. 그러나 역시 내가 아는 수준에서 프로그램을 작성했기 때문에, 상당히 영성하고 오류가 있는 부분이 있다. 지금부터 내가 구현하지 못했거나 문제가 있는 기능들에 관해 설명하겠다.

문제 1. 콘솔로 입력되는 프로그램이다.

전에 전화번호부를 설계하면서도 작성하였지만, 이는 콘솔 기반으로 입출력이 이루어지는 프로그램이기 때문에, 사용자가 입력하기 힘들 뿐만 아니라, 가독성도 떨어져 프로그램의 수요가 적을 것이라는 문제가 존재할 것이다. 그리고 콘솔로 구현한다는 점에서 프로그래머 입장은 문자 처리가 상당히 까다롭다는 점이다. 저번 전화번호부를 구현하고, 일부 문자가 입력되어도 검색이 되도록 작성하려고 했으나, 작동이 되지 않아 삭제하였고, 위에 코드 구현 부분에서 `getline`으로 입력받아, 띄어쓰기(Token)기반 문자 구분이 아닌, 개행 문자(`\n`) 기반으로 문자를 입력받으려고 했으나, 특정 상황에서 데이터가 온전히 전달되지 않는 문제가 있어 이 또한 삭제하여, 입력값으로 띄어쓰기가 들어온다면 프로그램은 정상 작동하지 않는다.

문제 2. stack의 사용

검색함수의 재사용을 위해 Index를 보관하는 Stack을 선언하여 사용하도록 했으나, Stack의 Index를 통한 데이터 접근할 수 없다는 것을 간과하여, 출력된 Stack을 출력해줄 때마다 새로 복사해서 사용한다는 점에서 메모리 낭비가 심한 프로그램임을 나타내고 있다.

문제 3. 파일이 입력되는 조건

이것은 의도한 것이지만, List를 저장한 텍스트파일이 모두 존재하지 않는다면 List에 데이터를 불러오지 않는다. 결국 하나의 파일이라도 없으면 List의 데이터 무결성이 떨어지기 때문이다. 그러나 이 점은 큰 단점이 된다. 결국 사용자는 List 파일 하나 삭제시킨 것으로 전체 List의 내용을 다시 작성해야 하기 때문이다.

문제 4. `stdlib` List의 과도한 사용

학부생 수준에서는 어쩔 수 없긴 하지만, 현재 Vector 3종, 함수마다 Stack을 2배씩 사용함으로 프로그램이 무거워지며, 현재는 데이터를 넣는데 매우 빠른 속도로 처리되지만, 추후 가게 규모로 데이터가 들어온다면, 프로그램의 연산이 느려진다는 단점이 있다.

문제 5. 데이터 입력에 대한 예외 처리

이 프로그램은 List가 비어있는지 확인해서, 불필요한 연산을 줄이는 것을 중심으로 작성되어있다. 이로 인해서 만약 비상식적인 길이를 가진 데이터가 입력된다면, 그대로 받거나 Stream이나 운영체제가 전달하는 최대의 크기로 저장되며, 이후 토큰으로 넘어가서 다른 부분의 데이터까지 무결성에 영향을 받을 것이다.

결국 설계자의 수준에 따라 프로그램의 품질이 달라지기 때문에, 내가 알지 못하는 문제들이 많겠지만 학부생 수준인 나는 이 정도의 문제가 있다고 생각한다.

4. 실제 프로그램 실행

<예외 처리문>

```
cout << "사용자의 전화번호를 입력해주세요." << endl;
cin >> T1.num;
if (T1.num.length() == 11)
{
    T1.num.insert(3, "-");
    T1.num.insert(8, "-");
}
o1.push_back(T1);
```

사용자에게 전화번호가 입력되었을 때, 하이픈 보정

```
Info(const string lname = "", const string lnum = "", const int lrental = 0)
: name(lname), num(lnum), rental(lrental)
{
    if (lnum == "")
    {
        srand((unsigned)time(NULL));
        int temp = rand() % 1000000; //gen serial number
        num = to_string(temp);
    }
}
```

이름으로 구분할 수 없는 비디오에 고유 인덱스 부여해 비디오 구분 + 대여상태 부여

```
if (o1.at(index).rental)
{
    cout << "사용자가 대여 중입니다. 반납 후 시도 해 주세요." << endl;
    system("pause");
    return;
}
```

관리자가 대여 중인 비디오/사용자 정보를 수정하려고 할 때

```
ifstream v_ifs, u_ifs, r_ifs;
v_ifs.open("VideoList.txt");
u_ifs.open("UserList.txt");
r_ifs.open("RentalList.txt");
if (v_ifs.is_open() && u_ifs.is_open() && r_ifs.is_open())
```

List를 저장한 파일이 1개라도 없을 때, 불러오지 않는 것 (데이터 무결성 보장)

```
vector<Info>::reverse_iterator v_iter, u_iter;
vector<R_Info>::reverse_iterator r_iter;
if (!o1.empty())
{
    for (v_iter = o1.rbegin(); v_iter != o1.rend(); v_iter++)
    {
        v_ofs << o1.at(v_iter - o1.rbegin()).name << " " << o1.at(v_iter - o1.rbegin()).num << " " << o1.at(v_iter - o1.rbegin()).rental << " " << "\n";
    }
    cout << "VideoList를 저장중입니다..." << endl;
}
```

저장 함수 호출 시, 역순으로 저장하며, 각 List의 데이터 유/무를 확인하고 저장한다.

<pre> auto it = find(o1.begin(), o1.end(), T1); while (true) { it = find(it, o1.end(), T1); if (it != o1.end()) { T2.push(it - o1.begin()); ++it; } } </pre>	
<pre> template <class T> void operator =(const T& o1) { if (num == "") // User Info { name = o1.u_name; num = o1.u_num; } else { name = o1.v_name; num = o1.v_num; } } </pre>	<p>데이터 색인 시, 해당하는 데이터가 있더라도 리스트의 끝까지 인덱싱하는 것 같은 비디오를 여러 개 가지는 비디오 가게 특성을 생각했다.</p>
<pre> void v_search(vector<Info>& o1) { system("cls"); if (o1.empty()) { cout << "비디오 리스트가 비어있습니다." << endl; system("pause"); } return; } </pre>	<p>해당 연산자는 비디오, 사용자 List 내에 존재하는 연산으로, 비디오 반납 시에 대여 List 객체를 그대로 비디오, 사용자가 저장된 List 객체로 가져오게 하였음</p>
<pre> cout << "수정하실 데이터를 선택해주세요 (1: 이름, 2: 전화번호)" << endl; cin >> u_check; if (u_check == 1) { //User List Index를 기반으로 R_Info의 리스트를 색인(operator 재정의), 렌탈 리스트의 사용자 이름 수정 cout << "수정하실 이름을 입력해주세요 " << endl; cin >> o1.at(index).name; cout << "이름 수정이 완료되었습니다." << endl; } </pre>	<p>각 리스트가 비어있는데, 데이터에 접근하는 함수를 실행시켰을 때 종료시킴</p>
<pre> if (t_index.size() > 1) { v_frame(); v_output(o1, t_index1); cout << "발견된 비디오가 2개 이상입니다. 제거하실 비디오를 선택해주세요." << endl; cin >> check; } </pre>	<p>사용자 정보 수정함수가 호출되었을 때, 사용자가 선택해서 정보수정 하도록 하였음</p>
<p>비디오/사용자를 검색, 제거, 대여, 반납하려고 할 때, 수정하려는 데이터를 선택하게 함</p>	