전화번호부 프로그램 계획서

과 목	소프트웨어공학
담 당 교 수	권 세 진
학 번	201720970
학 과	소프트웨어·미디어·산업공학부
이 름	권 대 한

목 차

1. 전화번호부 프로그램이란?

프로그램의 수요층은 결국 사용자가 전화번호를 기억하지 않고, 전화번호부 프로그램에 번호를 등록해서, 사용자의 시간을 절약한다는 것이 목적일 것이다.
결과적으로 프로그램이 정돈되어 있고, 사용하기 편하다면 많은 사람이 사용할 것이라는 생각이 들었기 때문에, 기본적인 기능에 충실하고, 최대한 프로그램이 사용자가 편하게 사용하는 프로그램을 만들기 위해 노력할 것이다.

- 전체적인 프로그램 구성 계획

전화번호부 프로그램의 기능을 가지기 위해서는 전화번호 추가, 제거, 검색, 그리고 현재까지 저장된 전화번호부의 내용을 볼 수 있어야 한다.
그러나 여기까지의 과정은 단지 콘솔 창에서 입력하는 전화번호가 휘발성 메모리인 램에 임시 저장되는 것이며, 매번 전화번호를 입력해서 출력하는 것은 프로그램의 요구 점인 전화번호를 대신 기억한다는 것에 어긋나기 때문에, 프로그램에서 이미 List에 저장된 전화번호를 텍스트 파일로 비휘발성 메모리에 저장하고, 사용자가 원할 때 저장된 텍스트 파일로부터 전화번호를 로드 할 수 있는 기능을 가짐으로써, 전화번호부를 대체하는 프로그램을 구현할 것이다.

- 기능 구현에 대한 계획

(1) 프로그램 초기 실행 시

프로그램이 가지는 기능들에 전화번호 추가, 제거, 검색, 출력에 번호가 부여된 메뉴 함수를 종료를 원하기 전까지 생존시키고, 작동시켜서 사용자에게 받은 숫자 입력으로 원하는 기능을 사용하도록 유도할 것이다.

(2) 사용자가 전화번호를 등록하려고 한다면

사용자가 가나다순으로 입력할 것이라는 보장이 없고, 실제로 가나다순으로 입력하게 프로그램을 제작하면, 사용자는 전화번호 정렬에 매달리게 된다.
만약 정렬이 완료된 전화번호부에 전화번호를 추가해야 하는 상황이 온다면, 이는 전화번호를 등록하는 사용자의 관점에서 최악의 상황에 해당하고, 전화번호 관리에 무수한 시간을 허비하게 된다. 프로그래머의 관점에서, 사용자의 관점에서 편리한 stdlib의 List를 사용할 것이다. 사용자로부터 입력 개수 제한이 없어지게 되고, 굳이 사용자의 입력을 순서 가나다순으로 입력할 필요가 없게 된다. 정렬법은 정렬함수 Sort(header : algorithm)를 사용해서 이름, 전화번호가 저장된 List의 순서를 정렬할 것이다. 그런데 Sort 함수를 이용해 객체의 내용을 비교하고 정렬하기 위해서는 정렬함수에게 지금 사용자가 원하는 정렬 방식이 무엇인지에 따라 Boolean 타입의 이름, 전화번호 비교식을 제시함으로 전화번호부를 등록할 때 사용자가 원하는 방법으로 정렬해서 전화번호부를 저장할 것이다.
그런데 만약 사용자가 전화번호 정렬을 원한다면 문제가 발생하는데, 숫자뿐인 전화번호가 입력되면 사용자가 원하는 것은 하이픈의 입력 유/무에 상관없이 정렬되는 것을 원할 것이다. 그러나 실제로 정렬하고자 한다면, 하이픈을 가지고 있는 전화번호가 우선순위를 가지고 우선 정렬될 것이다. 결국 제대로 된 정렬이 되지 않으리라고 예상되기 때문에, 만약 사용자가 숫자뿐인 전화번호를 등록하고자 할 때, String 클래스에 내장되어 있는 .length()를 통해 휴대폰 번호인지, 특수목적 전화번호인지 구분하고, .insert()를 이용해 번호 중간에 하이픈을 자동 삽입시켜주는 전화번호 등록함수를 작성할 것이다.

(3) 사용자가 전화번호를 검색하려고 한다면

이미 stdlib에 검색하는 기능이 구현되어있으므로, 기본적인 검색기능 자체는 stdlib의 find 함수를 기반으로 작동시키면 되겠다.

사용자가 원하는 전화번호를 찾기 위해서 List 전체를 색인해야 하는데, 이는 List의 시작 인덱스와 마지막 인덱스를 find 함수를 통해 색인하면 원하는 데이터를 찾을 수 있다. 그러나 함수가 구현된 방식은 일반적인 변수를 검색하게 되어 있어서 객체화된 데이터 색인을 위해서 컴파일러에 객체 중 어떤 변수를 기준으로 색인해야 하는지 알려줘야 한다. 그래서 class 내부 비교연산자인 ==의 재정의해서 이름 비교식을 만들어줌으로 객체 데이터의 이름 색인이 가능하다.

(4) 사용자가 전화번호를 제거하려고 한다면

위의 전화번호 등록함수 작성계획에서 말했듯이, stdlib의 List를 사용해서 전화번호부를 저장할 것이다. 프로그램 내부에서 순차적인 인덱스 값을 배정받아 차곡차곡 저장될 것이다. 그런데 전화번호만 등록할 줄 아는 사용자가 인덱스값을 찾아 제거할 수 없으리라 생각되어, 사용자가 제거 기능을 호출하고, 제거하고자 하는 사람의 이름을 입력했을 때, find 함수에 전달시킬 것이며, 사용자에게서 철저히 감춰진 프로세스 이후 제거할 것이다.

그리고 제거과정에서 생각되는 한 가지 문제점이 있다. List 중에서도 Vector를 사용해서 전화번호부를 저장할 것인데, 배열형식으로 데이터가 이루어져 있다는 게 문제다.

배열의 크기, 정적배열이 가지는 문제가 있는 것은 아니다. 이미 Vector에서 그 문제가 해결된 상태로 상용화가 되었기 때문에 문제가 없다.

실질적인 문제는 데이터가 중간에 추가되거나 삭제되면 변경지점 뒷부분부터 현재 저장된 인덱스까지 계속 데이터 계산을 해야 하는 문제가 있다. 이는 List 자체의 문제이고, 사용자가 전화번호를 시스템 리소스에 영향이 갈 정도로 추가, 제거하지 않기 때문에 아주 큰 문제는 아니다.

(5) 사용자가 파일에 전화번호를 저장하려고 한다면

사용자에게 저장하려는 파일이름을 받아 파일 출력 스트림을 통해 파일 저장을 선언한다.

먼저 List의 객체화된 데이터를 파일에 저장하기 위해서는 List의 모든 데이터를 반대 순서로 색인하면서, 이름, 전화번호 순으로 파일 스트림에 복사하고 텍스트 파일로 저장한다.

(6) 사용자가 파일에서 전화번호를 로드하려고 한다면

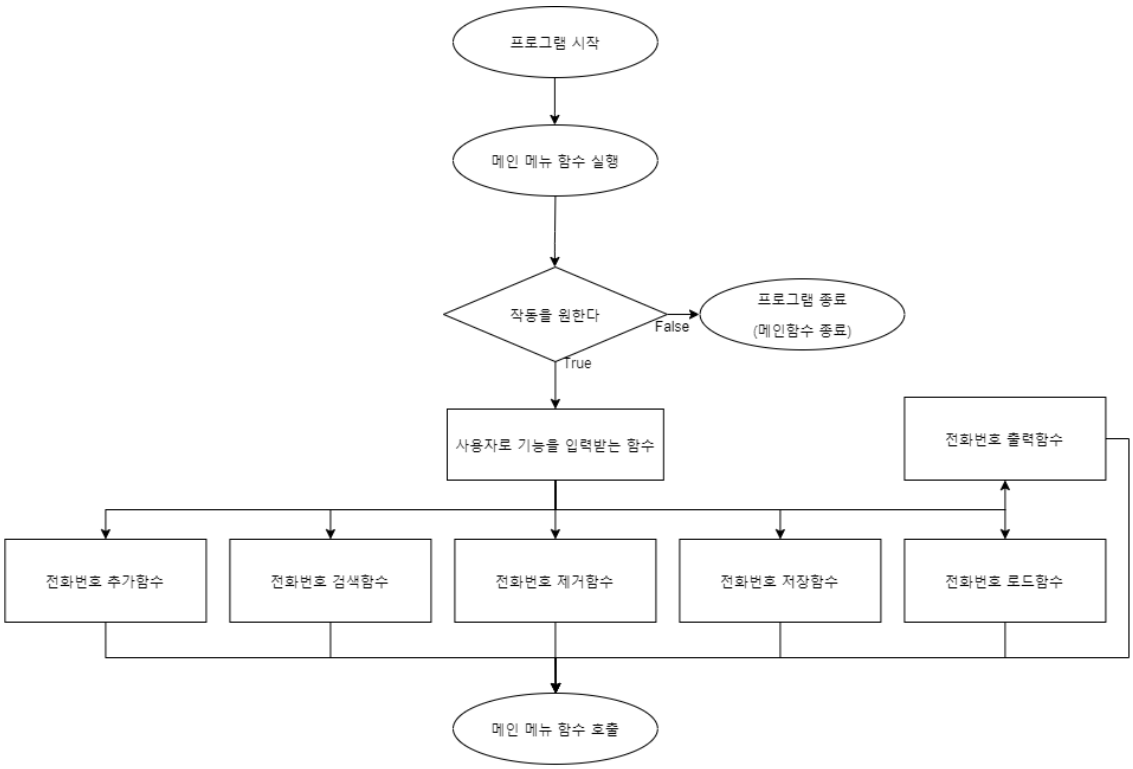
사용자에게 파일이름을 입력받아 파일 입력 스트림으로 로드하려는 파일을 선언한다. 파일 입력 스트림을 통해 파일의 내용을 토큰 단위로 불러오면서 새로 선언한 객체에 이름 전화번호 순으로 복사하고, List에 추가한다. 이후 파일의 끝(EOF)에 도달하면, 로드 함수의 작동을 중단한다.

(7) 사용자가 저장된 전화번호를 출력시키려고 한다면

출력함수가 호출되었을 때, List 타입의 데이터를 출력함수의 인자로 받아와서 데이터를 출력한다. 각 인덱스를 Iterator로 색인하는데, 이름, 전화번호를 value_type의 Iterator 인자를 불러와서 해당 인덱스의 데이터를 바로 출력한다.

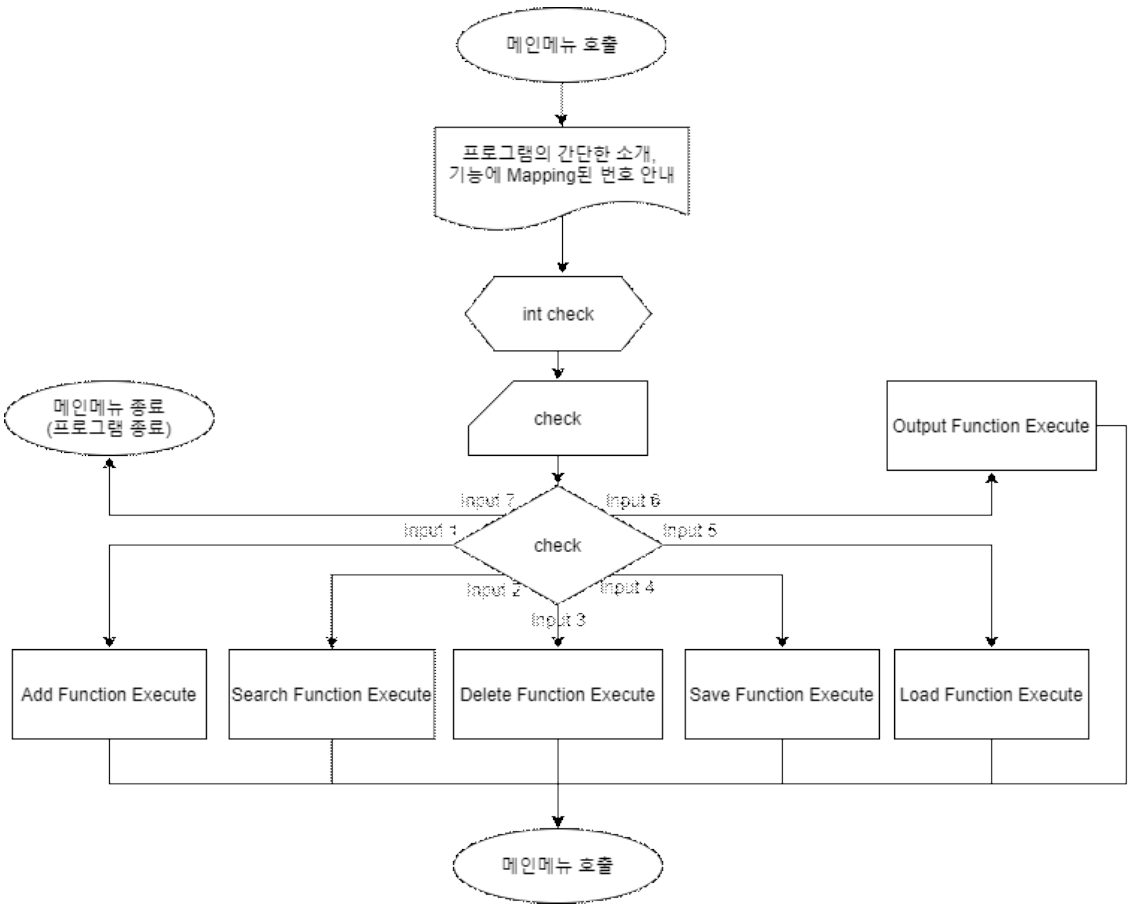
2. 논리 구성도

<전체적인 프로그램 구성도>



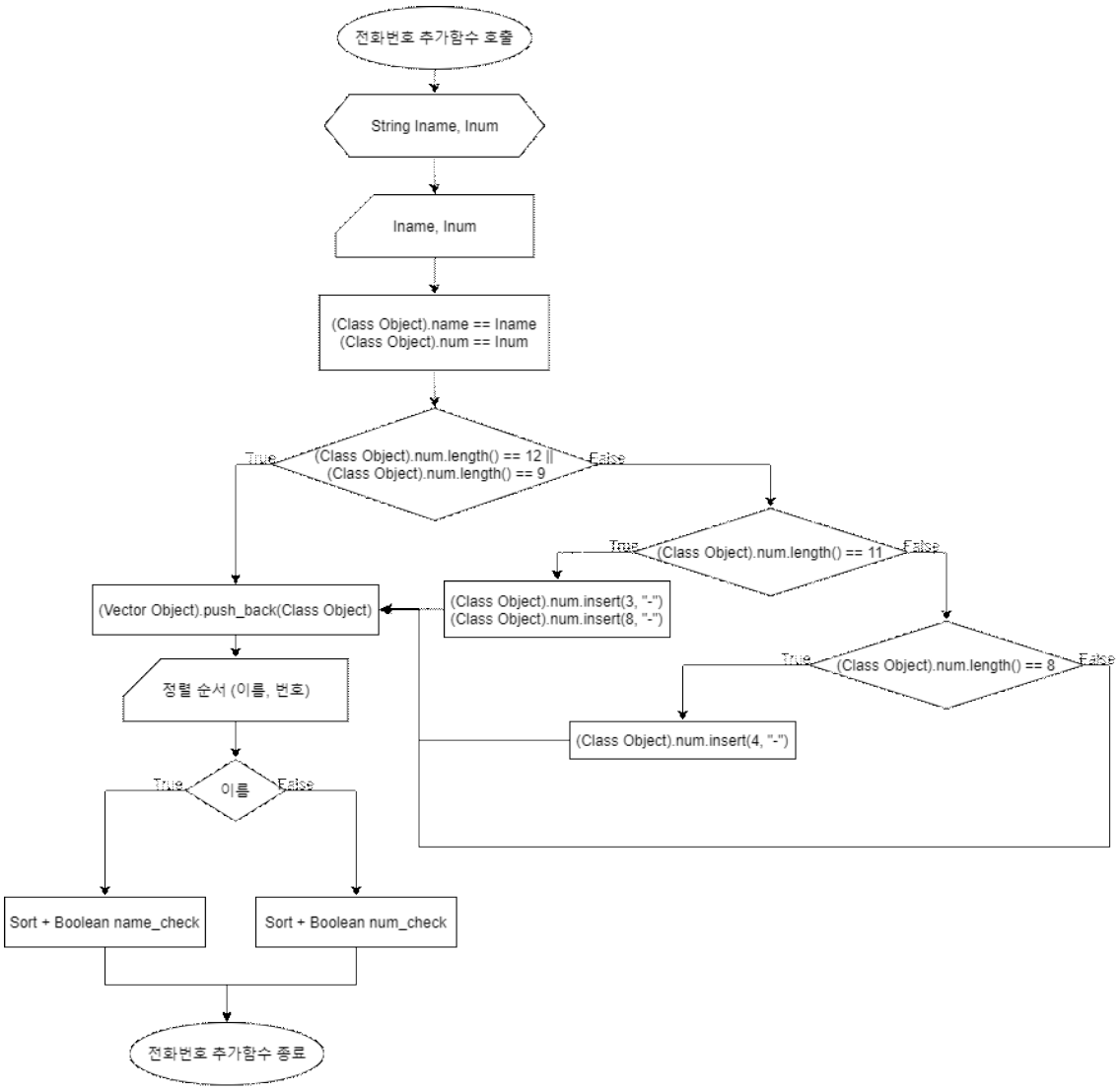
프로그램의 시작과 동시에 각 기능에 연결된 메인 메뉴 함수를 프로그램의 종료 전까지 실행시켜 사용자가 원할 때까지 프로그램을 사용할 수 있게 한다.
만약 사용자가 특정 기능을 사용하고 싶어 한다면, 화면에 표시된 맵핑 된 기능의 숫자를 입력함으로 특정 기능을 실행한다.
메인 메뉴 함수와 연결된 함수의 동작이 모두 마친다면, 다시 메인 메뉴 함수로 돌아오게 해서 일회성 기능 구현을 위한 프로그램이 아님을 강조하는 동시에 사용자가 더욱 편리하게 기능들을 사용하게 한다.

<메인 화면 구성도>



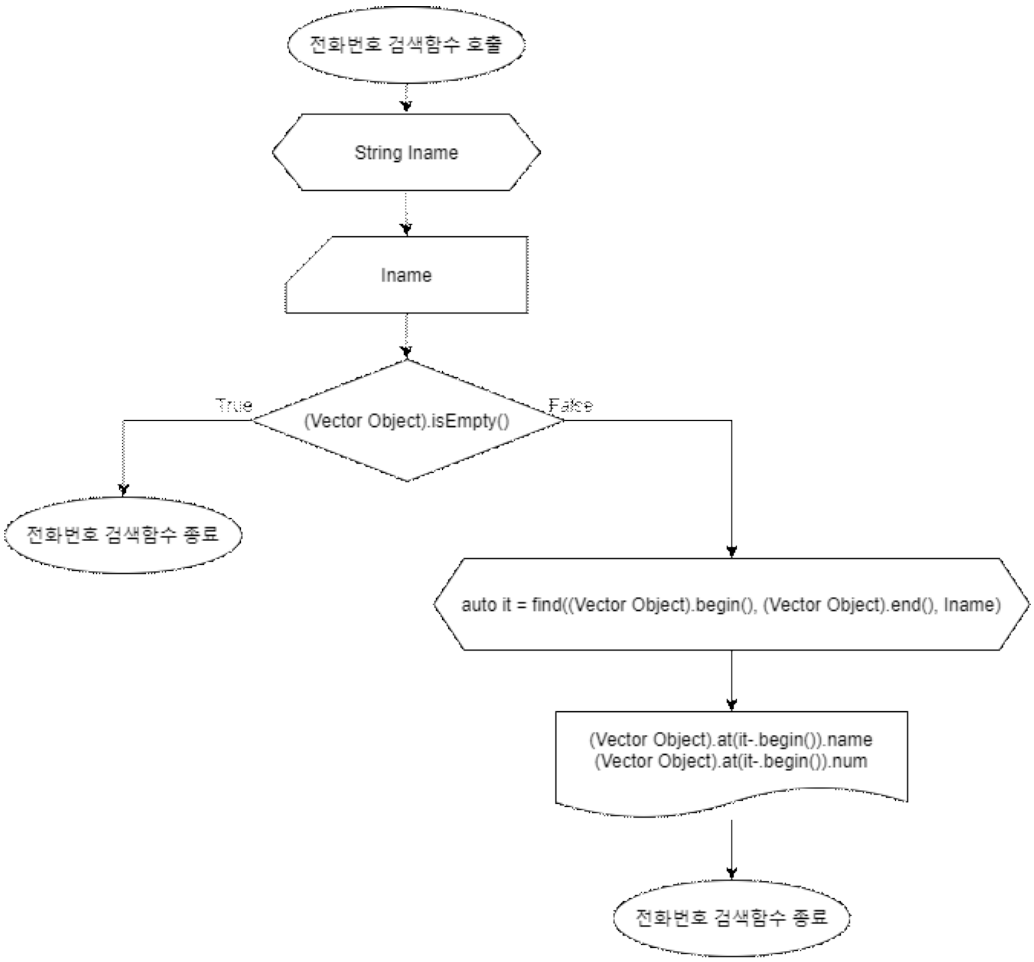
실제 프로그램 실행 시 메인 함수에서 전화번호를 저장할 List를 선언한다.
이후 메인 메뉴의 인자를 통해 메인 함수에서 선언한 List를 받아오면서, 메인 메뉴 함수와 연결된 기능함수들이 모두 같은 List를 연결하고, 처리하게 된다.
메인 메뉴 함수의 구조는 사용자가 원하는 기능을 숫자로 입력받음으로 1부터 7까지의 기능을 원하는 만큼 사용할 수 있도록 했다.
특히, 모든 기능을 마치고 다시 메인 메뉴 함수가 호출되도록 구성했기 때문에 종료를 누르기 전까지 메인 메뉴 함수는 영원히 루프를 돌면서 기능에 충실한다.

<전화번호 추가 구성도>



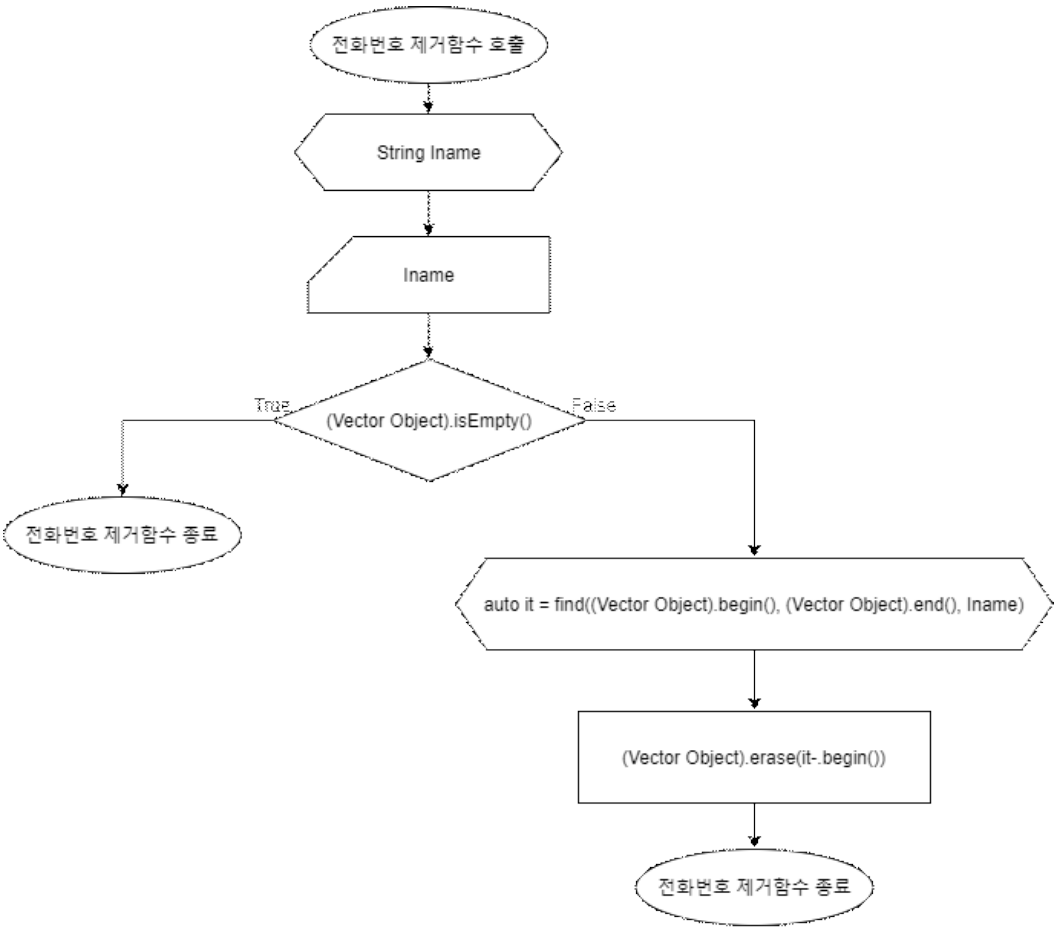
전화번호 추가함수가 호출될 때, 두 가지의 String Type 변수를 선언해서, 사용자에게서 추가하고자 하는 이름, 전화번호를 입력받는다. List에 두 개의 String Type의 변수를 한꺼번에 저장하기 위해서는 구조체와 같이 데이터를 객체화해서 저장해야 한다.
그래서 먼저 입력받은 데이터를 객체에 복사한다. 복사되었다면, 데이터의 정렬을 보장하기 위해 구분선을 맞춰 입력하지 않은 전화번호를 구분해서 데이터를 담는다.
구분선을 맞춰서 전화번호를 입력했다면, 객체 그대로 List에 저장하면 되겠고, 반대로 구분선에 맞춰 저장하지 않았을 때 전화번호 자릿수를 두 가지로 비교해서 저장한다. 입력된 번호가 휴대폰 전화번호인 11자리일 경우 두 번 구분선을 추가하고 List에 저장한다. 특수목적 전화번호 8자리일 경우엔 중간에 한 번 구분선을 추가하고, 저장을 마무리한다.
전화번호가 추가되었다면, 이름, 전화번호 순 정렬을 선택할 수 있도록 했으며, 이미 구현되어있는 Sort 정렬함수를 사용해서 정렬할 것이며, 객체 데이터를 비교하기 위해서는 어떤 데이터를 비교할 것인지 비교식이 필요하기에 비교식을 따로 만들어서 구현한다.

<전화번호 검색 구성도>



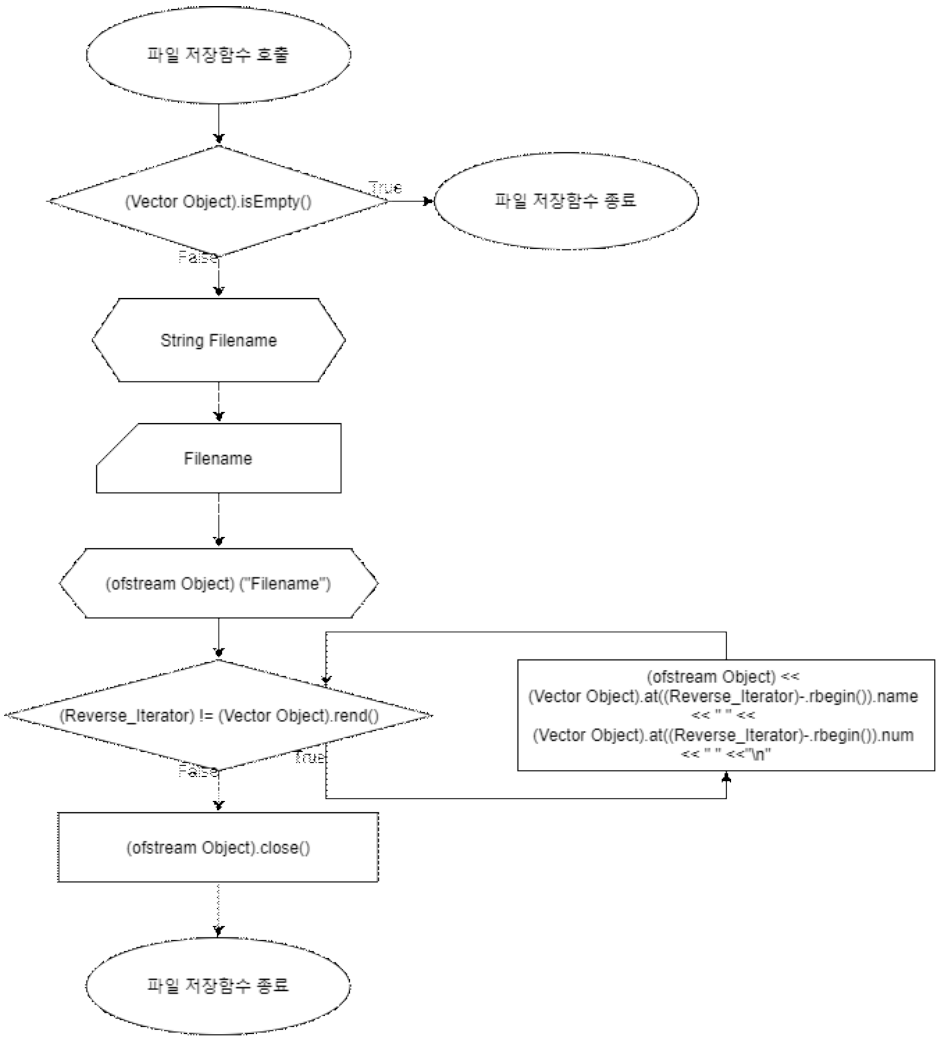
전화번호 검색함수 호출이 발생했을 때, 사용자에게 입력받을 String Type의 변수를 선언하고 사용자에게 검색하려는 사용자 이름을 입력받는다.
먼저 검색하기 전에 불필요한 데이터 색인을 피하고자, List가 비어있는지 크기를 확인한다.
실제로 List가 비어있다면, 전화번호부가 비어있다는 메시지와 함께 함수가 종료되며, 반대로 List에 전화번호가 들어있으면, 검색함수를 통해 입력받은 사용자의 전화번호부 인덱스를 찾는다. 상세 과정은 해당 Iterator 인자와 List 시작점의 거리로 도출된 인덱스(Integer Type)의 이름, 전화번호를 사용자에게 출력하고 전화번호 검색함수는 종료된다.
그러나 find 함수는 같은 타입의 데이터만 비교할 수 있으므로, Class 내부에 String Type의 변수와 객체의 비교가 가능한 비교 연산식을 재정의함으로 서로 다른 데이터 타입을 비교할 수 있게 하였다.

<전화번호 제거 구성도>



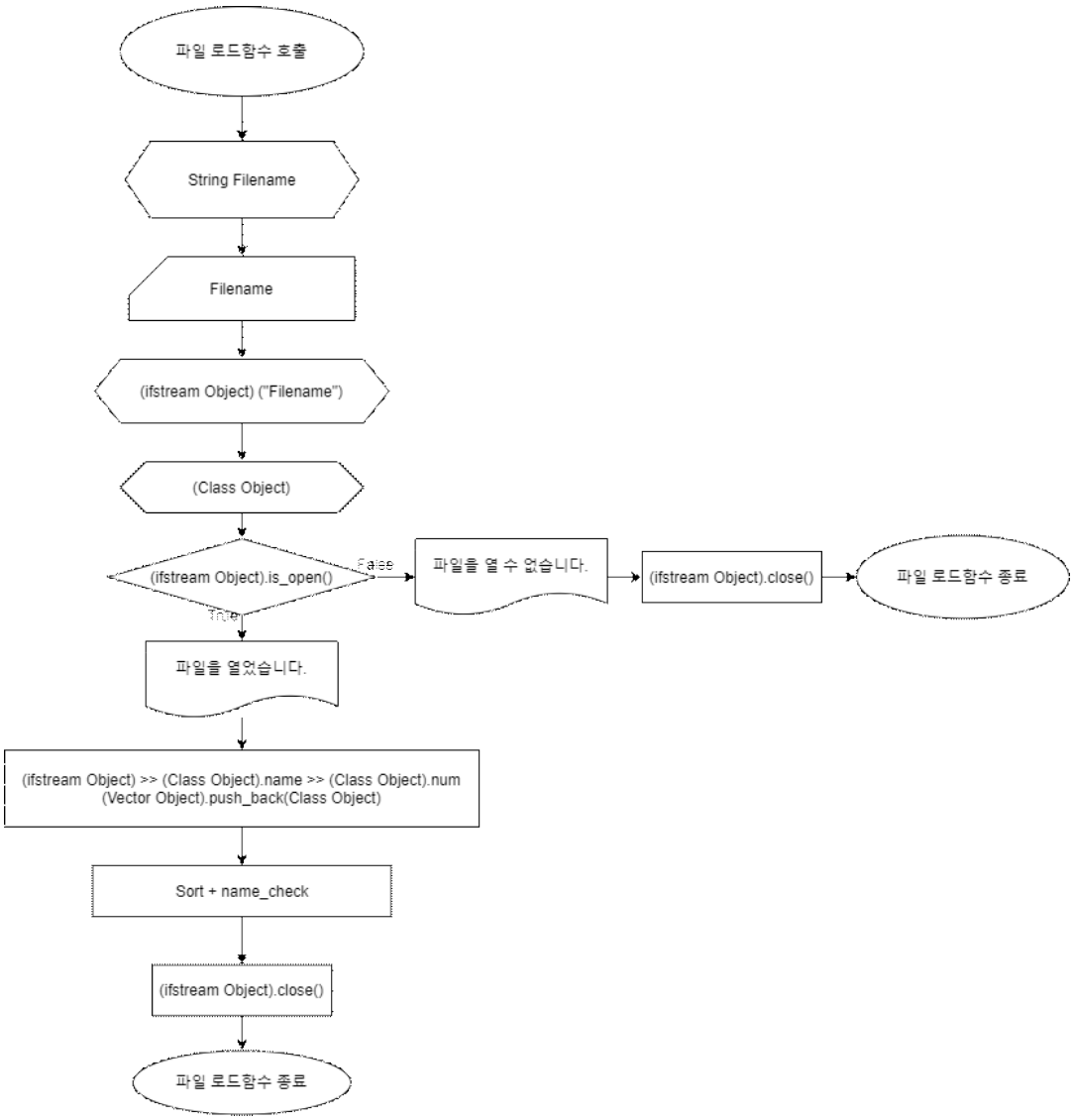
전화번호 제거함수는 전화번호 검색함수와 작동방식이 아주 유사하다고 할 수 있다. 사용자로부터 제거할 사용자의 이름을 입력받은 후에, 불필요한 데이터 색인을 방지하기 위해, 전화번호부가 비었는지 먼저 확인한다. 비었다면 전화번호 제거함수를 종료하고, 메인 메뉴 함수로 되돌아간다. 그러나 전화번호부에 전화번호가 1개라도 들어있다면, 처음에 입력받은 사용자의 이름을 find 함수와 미리 재정의한 비교식을 기반으로 List를 색인한다. 이후 리턴된 Iterator 인자와 시작점의 차이로 얻은 Integer Type의 인덱스를 지우면 전화번호 제거함수가 종료된다.

<전화번호 텍스트 파일 저장 구성도>



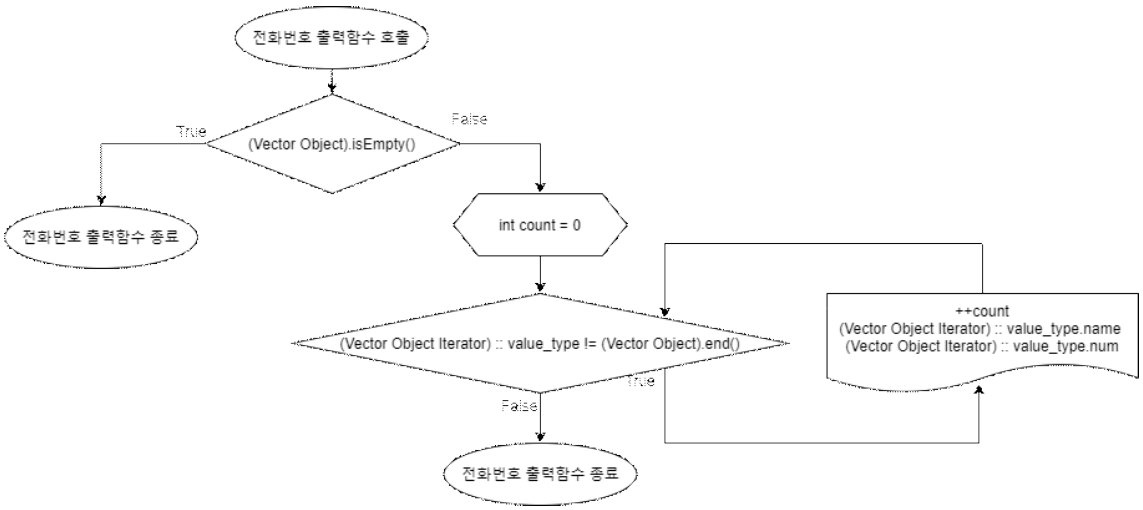
전화번호 저장함수가 호출되면, 불필요한 데이터 저장을 방지하기 위해서 전화번호부에 전화번호가 등록되어있는지 확인한다. 등록된 전화번호가 없다면, 저장함수는 종료되며, 메인 메뉴 함수로 되돌아가게 된다.
반대로 전화번호가 등록되어있다면, 사용자에게서 저장할 파일이름을 입력받고, 저장작업을 시작한다. 먼저 파일 저장하기 위해서 파일 출력 스트림에 저장할 파일을 선언한다. 그리고 List로부터 파일에 저장할 객체의 구체적인 데이터를 복사해주면 파일의 저장은 끝이다.
여기까지가 핵심 과정이라고 생각되어 더 이상의 문제가 없을 것으로 생각될 수도 있다.
그러나 한가지 간과된 것이 List는 이미 가나다, 번호가 빠른 순으로 정렬이 되어있는 상태다. 이것을 낮은 인덱스부터 순차적으로 파일 출력 스트림에 복사한다면, List에 저장된 인덱스의 반대로 텍스트 파일에 저장될 것이다. 본래는 Value_type의 Iterator를 사용해 바로 파일 출력 스트림에 복사했지만, 이를 해결 하기 위해서 Reverse_Iterator를 사용해서 List의 반대부터 순차적으로 역 인덱스의 마지막에 도달할 때까지 파일 출력 스트림에 복사함으로, List에 저장된 순서대로 파일에 저장하고, 전화번호 저장함수가 종료된다.

<전화번호 텍스트 파일 로드 구성도>



전화번호 로드함수가 호출되면, 사용자로부터 불러올 파일의 이름을 입력받는다.
그것을 기반으로 파일 입력 스트림인 ifstream 객체를 통해 불러올 파일을 선언한다. 파일을 찾을 수 없다면, 해당 파일이 존재하지 않는다는 메시지와 함께 로드함수가 종료된다.
파일 입력 스트림에서 파일을 찾으면 파일에서 데이터를 불러올 준비를 한다. 파일로 불러올 텍스트는 토큰 단위, 즉, String Type의 단일 변수로 불러오기 때문에, 객체 타입으로 구체화한 List에 데이터를 추가하기 위해서는 텍스트의 내용을 토큰 단위로 받아와서 이름과 전화번호를 Class Type의 객체에 1차 저장하고, 이후 List에 객체를 추가하면서 전화번호를 등록하며, 파일의 끝이 도달한다면 불러오는 과정이 자동으로 끝이 나게 된다. 텍스트 파일을 불러오면서 임의로 파일에 전화번호를 추가했을 가능성이 있기에, 전화번호를 정렬하고, 파일 스트림을 닫으면서 로드함수가 종료되게 된다.

<전화번호 출력 구성도>



전화번호 출력함수가 호출되면, 전화번호부가 비어있는지 먼저 확인한다.
전화번호부가 비어있다면 전화번호 출력함수를 종료한다. 전화번호가 등록되어있다면,
전화번호의 현재 인덱스를 알아내기 위해, Integer Type의 count 변수와 객체화된 데이터로 접근 가능한 Iterator 인자를 선언한다. 이후 반복문이 돌아가면서 count 변수의 전위 증가 연산을 시작으로 현재 Iterator 인자의 이름과 전화번호 출력을 시작한다.
List의 마지막에 도달하였다면 반복문을 중단하고, 전화번호 출력함수를 중단하면서, 메인 메뉴 함수로 되돌아가게 된다.

3. 프로그램 구현

위의 플로우차트에서 설명한 내용은 최대한 배제하고 구현 방법을 설명하였다.
코드 적 구현 방법, 왜 이 방법을 사용하였는지 설명하겠다.

<Class + Operator Overloading>

```
class Info
{
public:
  string name, num;
  Info(string Iname = "", string Inum = "")
    :name(Iname), num(Inum)
  {

  }
  bool operator== (const string c1)
  {
    return name == c1;
  }
};

bool name_check(Info c1, Info c2)
{
  return c1.name < c2.name;
}

bool num_check(Info c1, Info c2)
{
  return c1.num < c2.num;
}
```

데이터를 담을 구조를 어떻게 할 것인가 고민을 많이 하였는데, 클래스 사용을 선호하기도 하고, 추후 접근 지정자를 선택할 수 있어서 클래스로 구현하기로 하였다.
클래스 내부에 Bool 타입의 연산자 재정의 식이 있는데, 이는 전화번호 검색, 제거함수에서 객체와 String 타입의 비교를 위해 비교식을 작성해둔 것이다. 이때 데이터 무결성을 위해 const string Type으로 사용자 입력값을 복사해온다.
클래스 외부의 Bool 타입의 함수는 전화번호의 정렬을 위해 전화번호 추가함수에서 사용하는 것이며, 비교식이 호출되면서 2번째 변수가 큰 것일 때 True를 리턴시켜 데이터가 오름차순으로 정렬이 된다.

<메인 함수에서 메인 메뉴 함수의 시작 부분>

```
int main()
{
  vector<Info> List;
  cout << "전화번호부 프로그램을 시작합니다." << endl;
  menu(List);
}
```

프로그램 실행 시 처음으로 시작되는 메인 함수이다.
실행 시에 List를 선언하고, 메인 메뉴 함수에 List를 전달함으로, 모든 함수에서 같은 데이터를 다룰 수 있는 편리성, 코드가 간결해진다는 점을 높게 생각해 이렇게 구현하였다.

<메인 메뉴 함수의 인자 부분>

```
template<class T>
void menu(vector<T>& o1)
{
    int check = 0;
    while (check != 7)
    {
        cin >> check;
        switch (check)
        {
        case 1:
            Add_func(o1);
```

메인 메뉴 함수의 인자 값을 template type, List의 객체의 레퍼런스 값을 가져옴으로, 다른 객체 타입의 리스트가 불러와도 큰 수정 없이 사용할 수 있도록 하였으며, 종료 값인 7번이 입력되면 메인 메뉴 함수의 종료 후 프로그램 종료가 되도록 작성하였다.

<전화번호 추가 함수 중 정렬 부분>

```
template<class T>
void Add_func(vector<T>& o1)
{
    if (o1.size() != 1)
    {
        cout << "이름 순 정렬을 원하시면 1번을 입력하시고, 전화번호 순 정렬을 원하시면 2번을 입력하세요." << endl;
        cin >> sort_check;

        while (true)
        {
            if (sort_check == 1)
            {
                sort(o1.begin(), o1.end(), name_check);
                cout << "이름 순으로 정렬되었습니다. " << endl;
                break;
            }
            else if (sort_check == 2)
            {
                sort(o1.begin(), o1.end(), num_check);
                cout << "번호 순으로 정렬되었습니다. " << endl;
                break;
            }
```

```
if (Object1.num.length() == 11) //전화번호 정렬을
{
    cout << "휴대폰 번호입니다." << endl;
    Object1.num.insert(3, "-");
    Object1.num.insert(8, "-");
}
```

메인 함수의 List에 사용자에게 입력받은 이름, 전화번호를 객체화해서 전화번호를 저장한다. 사용자에게 원하는 정렬 조건을 제시하고, sort_check에 따라 이름의 오름차순 정렬, 번호의 오름차순 정렬을 실행한다.

<전화번호 검색 함수 중 검색부분>

```
if (o1.empty())
{
    cout << "전화번호부가 비어있습니다." << endl;
    return;
}
else
{
    auto it = find(o1.begin(), o1.end(), lname); //iterator type
    cout << o1.at(it - o1.begin()).name << "의 전화번호는 " << o1.at(it - o1.begin()).num << "입니다." << endl;

}
```

사용자가 검색할 이름을 String Type의 변수에 입력받고, Class에서 선언된 연산자 재정의 비교함수와 find 함수를 통해 o1 List의 시작부터 끝까지 색인을 한다. 결과는 it에 Iterator 인자 값을 전달하게 되며, o1의 시작과 현 Iterator 인자의 위치 차이를 구해서 색인이 완료된 List의 결과를 이름, 전화번호에 따라 출력한다.

<전화번호 제거 함수 중 제거 부분>

```
if (o1.empty())
{
    cout << "전화번호부가 비어있습니다." << endl;
    return;
}
else
{
    auto it = find(o1.begin(), o1.end(), Iname);
    cout << o1.at(it - o1.begin()).name << "의 전화번호가 제거되었습니다." << endl;
    o1.erase(it);
}
```

제거하려는 전화번호부의 이름을 String Type의 Iname에 입력을 받았다면, 미리 구현되어있는 find 함수를 통해 List의 처음부터 마지막까지 재정의된 연산자를 통해 색인을 진행하고, 해당 Iterator 값을 참조해서 List의 해당 인덱스를 제거한다.

<전화번호 저장 함수 중 List의 역 색인 부분>

```
if (o1.empty())
{
    cout << "전화번호부가 비어있습니다. 추가하고 시도해주세요!" << endl;
    return;
}
else
{
    string Filename;
    cout << "저장하려는 파일이름을 입력해주세요! ex)Daehan's P.B.txt" << endl;
    cin >> Filename;
    ofstream ofs;
    ofs.open(Filename);
    vector<Info>::reverse_iterator iter;
    for (iter = o1.rbegin(); iter!=o1.rend(); iter++)
    {
        ofs << o1.at(iter - o1.rbegin()).name << " " << o1.at(iter - o1.rbegin()).num << " " << "₩n";
    }
```

사용자에게 입력받은 String Type의 파일이름을 저장하겠다고 파일 출력 스트림으로 선언하였다. 일반적으로 순차적인 인덱스를 통해 도출된 데이터 순서대로 파일 스트림에 전달한다면, 스트림 입장에서 처음에 들어온 데이터를 파일에 저장하게 되면서, List에 저장된 반대 순서로 파일에 저장되는 문제가 있을 것이다. 그래서 평범한 Iterator 인자 대신 Reverse_Iterator를 사용해서 List의 반대, 역순서로 색인을 지정하고, 파일 출력 스트림에 연결해서 문제를 해결하였다. 처음 설계 단계에서는 value_type의 Iterator 인자로 파일 스트림에 데이터를 전달했지만, 실제로 실행해보니 반대로 저장되는 문제가 있어서 수정하였다.

<전화번호 로드 함수 중 List에 저장되는 부분>

```
if (!o1.empty())
{
    cout << "이제까지 입력된 전화번호부를 비우고 전화번호를 불러오시겠습니까? (입력 : Y/N)";
    cin >> check;
}
if (check == "y" || check =="Y")
{
    cout << "전화번호부를 비우고 파일로부터 전화번호를 불러옵니다." << endl;
    o1.clear();
}
else if (check == "n" || check == "N")
{
    cout << "이제까지 입력된 전화번호와 같이 불러옵니다." << endl;
}
else
{
    cout << "전화번호를 불러옵니다." << endl;
}
while (ifs >> Temp.name >> Temp.num)
{
    o1.push_back(Temp);
}
cout << "이름 기반으로 자동정렬됩니다." << endl;
sort(o1.begin(), o1.end(), name_check);
```

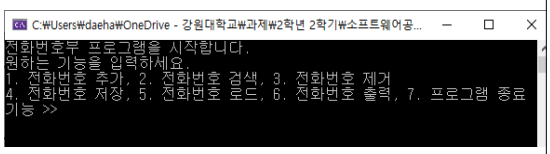
사용자에게 입력받은 파일이름을 기반으로 파일 입력 스트림이 선언된 상태이다. ifstream이 파일의 끝 EOF에 도달할 때까지 Token 단위의 문자열을 임시 저장 객체에 1차로 저장하고, 이후 List에 실제로 저장하는 방법을 사용했다. 띄어쓰기, \n을 기반으로 문자를 받아들이기 때문에, 간단하게 구현 가능했다.

<전화번호 출력 함수 중 value_type Iterator로 객체에 접근하는 부분>

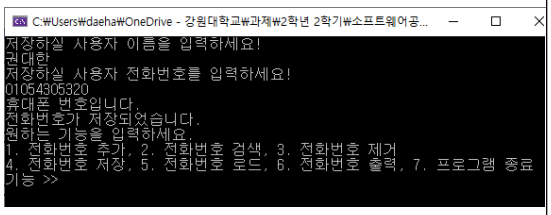
```
template<class T>
void Output_func(vector<T>& o1)
{
    system("cls"); //clear console
    int count = 0;
    if (!o1.empty())
    {
        for (const auto& iter : o1) //value_type iterator
        {
            cout.width(3);
            cout << ++count;
            cout.width(9);
            cout << iter.name;

            if (iter.num.length() == 9)
            {
                cout.width(13);
                cout << iter.num << endl;
            }
```

for문 내에 iter라는 자동 레퍼런스 변수를 선언함으로, List 객체인 o1의 마지막에 도달할 때까지 Iterator :: value_type으로 값을 받아서 동작하도록 작성했기에, 깔끔해 보이며, 번호에 따라 적당한 공백을 주고 바로 객체에 데이터를 출력할 수 있게 작성하였다.

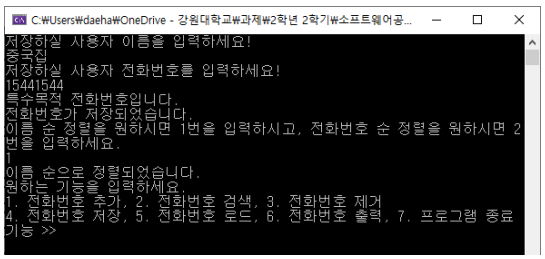
4. 실제 프로그램 실행



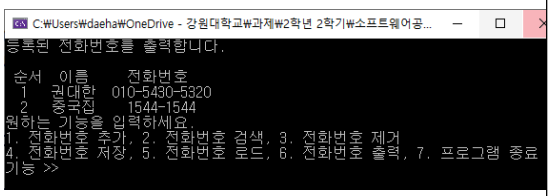
<초기 프로그램 실행 시>



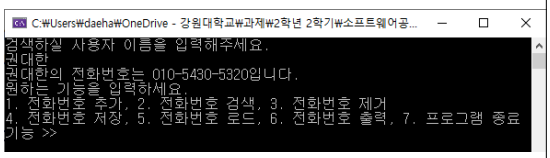
<초기 전화번호 저장 시>



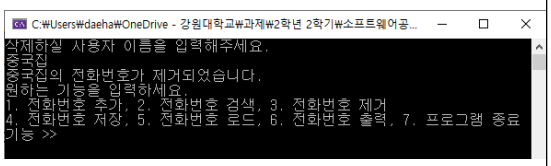
<한 개 이상의 전화번호가 들어있을 때>



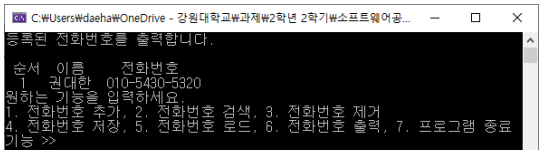
<현 상태의 전화번호부 출력 시>



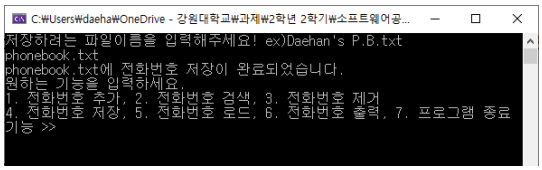
<전화번호 검색 기능을 실행했을때>



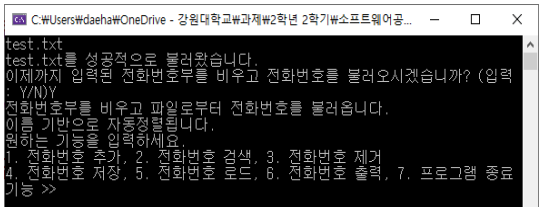
<전화번호 제거 기능을 실행했을 때>



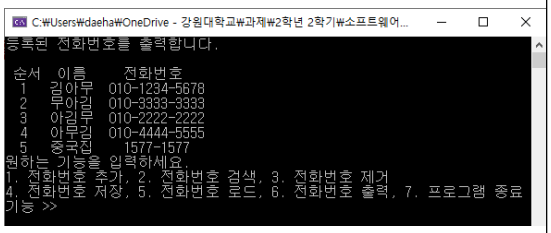
<제거 기능이 동작한 후 전화번호부>



<전화번호부를 텍스트 파일로 저장 시>



<이미 전화번호가 들어있는 상태에서
텍스트 파일을 로드하려고 할 때>



<로드 후의 전화번호부 출력>

<프로그램 개발 후 결과분석>

사용자가 친숙하게 사용할 수 있고, 크게 신경을 쓰지 않아도 되는 프로그램을 만들려고 했으며, 현재 요구분석, 설계, 구현까지 완료하였다. 그런데 지금 프로그램 결과를 봤을 때 단지 C++과 low-level Stream을 이용해 프로그램을 작성했기 때문에, 사용하기 불편한 프로그램이 만들어졌다고 생각한다. 그리고 이 문제의 가장 큰 요인은 내가 아는 만큼의 수준에서 프로그램을 설계했기 때문에 상당히 엉성한 프로그램이 만들어졌음을 느낀다.

내가 느낀 이 프로그램의 문제점을 설명하고자 한다.

문제 1. 단지 C++을 이용해 콘솔 창에서 모든 입출력이 처리되기 때문에 문자의 조합으로 이루어지는 한국어의 입력이 상당히 불편하다. 이는 콘솔로 작성된 프로그램의 한계로 GUI 기반의 프로그램으로 작성 후 TextForm으로 입력을 받거나, Mac OS의 Xcode로 실행하면 이와 같은 문제는 없어진다.

문제 2. 콘솔 프로그램이기 때문에, 2020년에 사는 사람들이 프로그램을 사용하려고 하지 않는다. 요즘은 GUI 엔진이 잘 되어있으며, 운영체제의 Low-level까지 GUI로 조절이 가능한 시대에서 콘솔 프로그램과 같이 CLI 기반으로 작동해서 가독성이 떨어지는 프로그램을 사용하려고 하지 않을 것이다. 이는 WinAPI를 제외하더라도 QT와 같은 라이브러리로 작성한다면 보기도, 사용하기도 편한 프로그램을 작성할 수 있을 것이다.

문제 3. 메인 메뉴 함수에서 1 ~ 7 범위의 수와 문자를 동시에 입력한다면, 이름값이 들어가지는 문제가 있다. 이는 Integer Type의 변수와 String Type 변수의 입력이 동시에 처리되어서 생기는 문제인데, 기능함수들이 작동되었을 때, 입력된 변수의 길이가 짧거나, 입력된 이름값을 사용자에게 재확인함으로써 문제 해결이 가능하다.

문제 4. 현재 프로그램 내에서의 검색, 제거함수에서 전화번호 기반 검색, 제거 기능이 없다. 이는 Class의 비교연산자 재정의 함수에서 이름뿐만 아니라, 전화번호가 같더라도 True를 리턴하도록 한다면 구현할 수 있다.

문제 5. 검색, 제거함수에서 이름의 일부 문자로 검색, 제거할 수 없다.

이는 비교연산자 재정의 함수 내부에서 객체의 name의 일부를 찾아내는 함수를 사용해서 name.find()로 일부 포함된 문자를 찾아내려고 하였지만, 가끔 포함되지 않은 사용자까지 출력하는 문제가 있어 기능을 제외하였다. 왜 오작동 하는지 구체적인 이유를 모르겠다.

문제 6. 요즘 전화번호부에는 생일, 집 주소, 그룹과 같은 많은 자세한 기능을 가지지만, 이 프로그램은 이름, 전화번호만 입력할 수 있다. 이는 굳이 기능을 추가하고 싶다면 추가할 수는 있지만, C++만을 이용해서 큰 객체의 데이터를 추가한다는 것은 상당히 비효율적이고, 개발하는 시간적으로도 많이 소요되기 때문에 추가하지 않았다.

문제 7. 전화번호 로드 시 프로그램 내에서 생성된 텍스트 파일을 로드하면 문제없지만, 띄어쓰기 없이 마음대로 텍스트 파일을 새로 저장하고, 프로그램에서 로드한다면 이름 전화번호 구분 없이 토큰 단위로 로드한다. 이는 파일 입력 스트림에서 입력되는 문자열의 길이에 조건식을 추가해서 해결할 수 있다. 이름보다 비상식적으로 긴 문자열이 입력된다면, 처음으로 숫자에 도달하는 문자열의 인덱스부터 전화번호로 입력받는다면 해결할 수 있다고 생각한다.

이것 말고도 타인이 내 프로그램을 본다면, 더 많은 문제가 보일 것이지만, 내가 생각하는 프로그램의 문제점은 이 정도이다.