

전화번호부 PhoneBook

K42 팀

- 201890110 허솔지
- 201911210 안효진
- 201920501 안희연
- 201930823 서혜린
- 201950508 박정화

이 문서는 2022년 전공기초프로젝트2 과목 수강생들을 위해 기획서의 형식에 대한 개략적인 예시를 보일 목적으로 작성된 문서입니다. 이 문서에 **지금 이 색상**으로 쓰여있는 모든 사항들은 기획서의 **‘내용’의 일부가 아니라** 이 문서를 참고하는 학생들에게 기획서 작성에 관한 **‘안내 및 부연 설명’을 하려는 추가 메모**로서, 실제 기획서에는 쓰여있지 않아야 할 사항들입니다. 또한, 이 문서에서 다루는 주제는 어디까지나 예시용 주제로서, 실제 교과목에서 프로젝트 주제로 선택하여 한 학기 동안 수행하기에는 특점에 불리할 수 있을 만큼 극도로 단순하다는 점에 유의하기 바랍니다.

기획서의 내용은 반드시 ‘무엇^{what}을 만들려는지’여야만 하지만, 그 내용을 작성하는 방식(‘전체 흐름 구성’이나 ‘표기 형식’, ‘설명 방식’)에는 절대적인 규범이 없고 대상(내용)이나 개발팀 성향 등에 따라 상당히 달라집니다. 이 문서는 통상적인 소프트웨어 개발 기획서 작성 방식들 중 ‘그나마 공통적·보편적·관계적인 요소들’을 대체로 따르고 있지만, 여전히 많은 부분은 예시용 주제의 특성에 알맞게 선택·조정되어 있습니다. 따라서, 기획서 개념 설명 영상에서 강조했던 일부 요소들(표제 번호 형식, 목차 형식¹⁾ 등)은 가능한 한 그 설명과 이 예시 문서대로 따라하기를 권장하지만, 그 외의 나머지 요소들은 각 팀의 주제에 알맞게 스스로 약간 변형시켜도 되고, 그렇게 하기를 오히려 권장합니다.

목 차

1	개요	2
2	용어	3
3	기본 사항	8
3.1	작동 환경	8
3.2	프로그램 구성	8
3.3	프로그램 설치 및 실행	9
4	프로그램 사용 흐름도	10
5	데이터 요소	11
5.1	이름	11
5.2	전화번호	12
5.2.1	전화번호 목록	13
5.3	메모	13

1) ‘형식’을 강조해둔 이유는, 목차나 표제의 ‘내용’은 억지로 무조건 따라해선 안 되기 때문입니다. 예를 들어, “아항! 5번 표제의 제목은 반드시 ‘데이터 요소’여야 하는구나”라고 생각해선 안 됩니다.

6 데이터 파일	14
6.1 문법 규칙 (형식)	14
6.1.1 설명과 예시	15
6.2 의미 규칙 (추가 조건)	17
6.3 부가 확인 항목	17
6.3.1 불필요한 중복 전화번호 (표준화)	17
6.3.2 동명이인	18
6.4 무결성 확인 및 처리	18
7 주 프롬프트	19
7.1 도움말 명령어군	21
7.2 종료 명령어군	22
7.3 무결성 확인/처리 명령어군	22
7.4 검색 명령어군	23
7.5 추가 명령어군	24
7.5.1 부 프롬프트 1: 이름 입력	24
7.5.2 부 프롬프트 2: 메모 입력	25
7.5.3 부 프롬프트 3: 저장 확인	26
7.6 삭제 명령어군	26
7.6.1 부 프롬프트: 삭제 확인	27
8 기타 처리	28

1 개요

지인이나 업체들의 전화번호들을 저장하고 검색하는 프로그램.

명령어 및 그 인자들을 키 입력하는 방식으로 사용하며, 사용자가 직접 편집할 수도 있는 텍스트 파일에 데이터를 저장하고, 한 사람/업체 당 여러개의 전화번호들을 저장할 수 있으며, 동명이인 구별을 위한 메모 필드가 있고, 부분문자열 검색이 가능하며, 여러 검색어들을 논리곱(∧)으로 묶어서 모든 검색어를 전부 포함하는 교집합을 검색할 수 있습니다.

이 문서는 개요에 하나의 절(1절)을 할애해서 그 내용으로 개요를 작성했는데, 실제로 흔히 사용되는 구성이기도 하고, 목차를 조금이라도 더 알뜰 표시하려는 의도도 있습니다.

반면에, 때로는 **절을 아예 할애하지 않고** 첫번째 절보다 앞에 (물론, 문서 제목/작성자 등은 모두 표시한 후에) 개요를 작성하기도 합니다. 사실 내용상으로는 개요야말로 문서 제목을 직접적·포괄적으로 설명하는 핵심 본문이고, 각 ‘최상위 절’들(1절, 2절, ...)은 그저 그 본문(개요)에 딸려있는 ‘하위의 세부 사항’들이므로, 이 형식이 개념적으로 좀 더 합당할 수도 있습니다. 이 경우, 목차는 주로 개요 직후 1절 직전에 넣는데, (목차의) 바로 위에 쓰여있는 개요의 포괄적인 설명이 지금부터 (목차 이후부터) 세부적으로 어떻게 갈라지는지를 미리 전체적으로 표시해주고 시작하는 맥락입니다.

여러분이 이 과목의 기획서를 작성할 때에는 두 방식 중 어떤 방식을 써도 상관 없습니다.

2 용어

이 문서에서 사용할 용어들의 의미를 약속하고, 아울러 일부 용어들과 관련하여 이 문서에서 사용할 표기법도 함께 약속해둡니다. 여기서 약속하는 용어들은 다음과 같이 분류됩니다:

- 이미 세간에서 지배적으로 통용되는 의미대로지만, 소개나 설명이 필요한 경우 (용어에 **기존** 표식)
- 세간에서 지배적으로 통용되는 (포괄적인) 의미에서 벗어나지는 않고 여전히 그에 포함되지만, 보다 특정한 범위나 대상으로 좁혀서 사용하려는 경우 (용어에 **특정** 표식)
- 세간에서 둘 이상의 의미들로 혼란스럽게 통용되고 있어서, 그 중 어떤 의미로 사용할지 명확히 정하려는 경우 (용어에 **명확** 표식)
- 세간에서 널리 통용되는 의미(들)에서 다소 벗어난 다른 의미로 바꿔 사용하려는 경우 (용어에 **변경** 표식)
- 세간에서는 거의 사용되고 있지 않은 용어를 이 문서에서만 쓰려고 새로 정의하는 경우 (용어에 **신규** 표식)

[주의] 아래 용어들 중 **변경** 이나 **신규** 표식이 붙은 용어들은 어디까지나 이 예시 문서 안에서만 이런 의미로 이해하고, 이 예시 문서 밖에서는 이런 의미로 사용하지 말아야 합니다. 특히, 수강생들과는 달리 이 예시 문서를 구경해본 일조차 없는 ‘이 과목 밖의 세상’에서는, **신규** 용어들을 마치 원래 통용되는 용어인 것처럼 사용하거나 **변경** 용어들을 여기에 약속된 의미로 사용해선 안됩니다.

만일 학생들도 본인들의 기획서에 이런 용어들을 이런 의미로 사용하고 싶다면, 본인들의 기획서 초반에 해당 용어의 의미를 **만드시 이렇게 다시 명시**해두고 사용해야만 합니다. 즉, 심지어 ‘이 과목 안의 세상’에서조차도, 이 용어들은 명시적인 약속 없이 (이런 의미로) 사용할 수 있는 용어들이 아니라는 뜻입니다.

이 절에 나열된 용어들은 내용상 비슷한 계열의 관련 용어끼리 가까이 묶되, 묶음 간의 순서는 본문에 등장하는 순서를 가능한 한 따르고, 묶음 속의 순서는 만일 A 용어를 이용해서 B 용어를 정의해야 할 경우 A를 B 보다 먼저 정의하고 있습니다.

용어 정의 순서에 정답은 없습니다만, 용어 정의glossary와 색인index은 그 특성이 다르니 혼동하지 말고 잘 구별하기 바랍니다: 규모가 큰 (‘책’ 정도의) 문서에는 각 단어별로 그 단어가 처음 등장하는 페이지 번호를 대응시킨 ‘색인’이 (주로 맨 뒤에) 붙어있는데, 이런 색인들은 거의 100% 자모 (알파벳) 순입니다. 이는 색인의 본질적 특성상 최소 백여 ~ 수백 개의 단어들을 나열하고 있는 데다가, 순서대로 전부 읽는 걸 전제하는 게 아니라 필요할 때 필요한 단어만 잠깐 찾으러 와서 그 페이지 번호만 열른 확인한 후 즉시 색인을 떠나는 걸 전제하므로, ‘엄청 많은 단어들 중 한 단어만 찾을 때 최대한 빠르도록’ 구성하는 게 가장 중요한 미덕이기 때문입니다.

반면에 용어 정의는 (색인 만큼 용어 갯수가 많은 경우도 아주 가끔 있지만) 대체로 색인과는 비교가 안 되게 확연히 갯수가 적고, (특히 처음 읽을 때에는) 용어 정의들 전체를 모두 순서대로 읽어보는 경우도 전제하는 데다가, 용어마다 그에 관한 설명을 꼼꼼히 읽어야 하며 그렇게 읽다보면 관련된 다른 용어의 정의를 또 찾아야하는 일이 자주 벌어집니다. 그래서, 용어 정의는 (비록 한 용어만 찾을 때는 약간 느려지는 걸 감수하더라도) ‘**위에서부터 순서대로 읽어도 이해하기 쉽고, 관련 용어를 연쇄적으로 찾기 쉽게**’ 구성하는 게 더 중요합니다. 따라서, 다시 강조하건대 용어 정의 순서에 정답은 없지만, 용어 정의를 **굳이 색인처럼 자모순으로 정렬할 필요가 없다**는 점은 기억하기 바랍니다.

홈 경로^{기존} 운영체제가 각 사용자(계정)에게 제공하는 고유의 디렉토리(폴더)의 경로. 이 문서에서 경로명 전체 혹은 시작 부분에 “{HOME}”이라고 표기된 부분은 항상 홈 경로입니다. 예를 들어, 만일 사용자 계정명이 “reeseo”라면 홈 경로는 (운영체제 관리자의 설정에 따라 달라질 수 있지만, 기본적으로):

- MS Windows Vista, 7, 8, 10의 경우 “C:\Users\reeseo”입니다.
- Linux (중 대부분의 배포판들) 등 **FHS**를 준용하는 운영체제의 경우 “/home/reeseo”입니다.
- Apple OS X, macOS의 경우 “/Users/reeseo”입니다.

전화번호부 프로그램 (혹은 프로그램)^{특정} 이 문서를 통해 기획·명세하고있는 대상 프로그램인 《전화번호부 PhoneBook》 프로그램

주 실행 파일 (혹은 실행 파일)^{특정+변경} 전화번호부 프로그램을 실행시키기 위해 꼭 필요한 본체에 해당하는 파일. 엄밀히는 이 파일은 스스로 (혹은 운영체제의 도움만 받아서) 실행 가능한^{self-executable} 파일은 아니고 Python 인터프리터에 의해 읽히고 해석될 내용—Python 인터프리터 입장에서는 넓은 의미의 ‘데이터’인 소스 코드—을 담고있는 텍스트 형식의 스크립트 파일이지만, 이 문서에서는 혼동될만한 다른 진짜 ‘실행’ 파일을 언급할 일이 없기 때문에 편의 상 그냥 이 파일을 실행 파일이라고 부릅니다.

스크립트라고 제대로 부르지 않고 실행 파일이라는 ‘잘못된’ 표현을 굳이 사용하는 데에는 수강생들의 혼란을 줄이려는 이유도 있습니다. 예시 문서에서 프로그램의 본체에 해당하는 파일을 “스크립트”라고 부르는 것을 보고 C나 C++, Java 언어 사용자들까지 본인들 프로그램의 (기계어로 컴파일된) 진짜 ‘실행’ 파일이나 (바이트코드로 컴파일된) 이진 파일을 스크립트라고 잘못 부를 가능성을 줄이려는 것입니다.

데이터 파일 (혹은 파일)^{특정} 전화번호부 프로그램이 연락처를 저장하고 읽어들이는 텍스트 형식의 파일. 이 프로그램은 오직 한 개의 데이터 파일만 다루므로 달리 수식어나 번호, 이름 등을 덧붙여 부르지 않습니다.

숫자^{특정} (서)아라비아 숫자들 중 표준 키보드로 직접 입력할 수 있는 10 개(U+0030 ‘0’ ~ U+0039 ‘9’)만을 뜻합니다. 즉, 이 문서에서 말하는 “숫자”에는 아라비아 숫자가 아닌 로마 숫자나 각 언어별 고유 숫자 기호들은 포함되지 않으며, 아라비아 숫자들 중에서도 전각 숫자, 원·괄호로 둘러싸인 숫자, 위·아래 첨자용 숫자, 수식용 글꼴별 숫자 등은 역시 모두 제외됩니다.

얼핏 당연해보이는 사항을 굳이 명시하는 이유는, 각종 입력기^{IME}나 복사·붙여넣기 등의 방법을 동원하면 유니코드 표에 잔뜩 등재된 다양한 숫자들을 실제로 컴퓨터에 입력할 수 있고, 처리 방법에 따라서는 컴퓨터가 이들 중 일부를 정말로 ‘숫자’로 간주하여 처리하도록 (의도적으로든 실수로든) 만들 수도 있기 때문입니다. (예: Python에서 `"0".isdigit()`의 리턴값은 True입니다.) 또한, 숫자^{digit} 외에 수^{number}도 동시에 다루는 팀들은 이 두 용어를 서로 잘 구별하고 각각 적절히 특정해서 사용하기 바랍니다.

개행^{특정} 텍스트 형식의 파일을 편집할 때 표준 키보드의 **Enter** 키로 입력할 수 있는 두 문자들 (U+000A Line Feed, U+000D Carriage Return) 중 하나 혹은 이들의 조합. 텍스트 편집기에서 **Enter** 키를 누를 때 둘 중 어떤 문자나 조합이 입력될지는 운영체제/편집기마다 다르지만, 무조건 사용자가 사용하는 운영체제/편집기에서 입력되는 문자(조합)를 개행이라고 부르겠습니다.

참고로, MS Windows에서는 CR/LF 조합이 개행으로 사용되고, 유닉스/리눅스는 LF 단독, macOS 계열들은 CR 단독으로 개행을 의미합니다. 그리고 Python에서는 이들을 모두 자동감지하여 적절한 문자나 조합을 개행으로 취급해줍니다.

사실은, ‘넓은 의미의 개행’에는 이들 둘(의 조합) 외에도 세로 탭(U+000B Vertical Tab)이나 용지 먹임(U+000C Form Feed) 등 몇 가지 문자들이 더 포함되지만, 이 기획서에서는 이런 문자들을 혹시 언급할 일이 있으면 반드시 (“개행”이 아니라) 각각의 구체적인 이름으로 부르겠습니다.

문서나 화면에서 개행은 대체로 2차원적으로 표현되므로 시각적으로 혼동할 여지가 적지만, 간혹 문서나 화면 폭 한계에 의한 자동 줄바꿈^{line wrap}과의 구분을 명확히 해야할 때에는 개행 직전 줄의 맨 끝에 ‘↵’ 기호를 덧붙여 표시합니다.

탭^{특정} (혹은 가로 탭^{기준}) 텍스트 형식의 파일을 편집할 때나 IDLE, 일부 터미널 등에서 표준 키보드의 **Tab** 키로 입력할 수 있는 문자(U+0009 Character Tabulation). 사실은 세로 탭(U+000B)도 (넓은 의미의 개행에 포함되면서, 동시에) 탭의 일종이긴 하지만, 만일 언급할 일이 있으면 반드시 “세로 탭”이라고만 부르겠습니다. 이 기획서에서 탭을 시각적으로 명확히 표시해야할 때에는 ‘→’ 기호로²⁾ 표시합니다. 이

2) 이 기호는 탭을 시각적으로 드러내는 의미로 널리 통용되는 기호지만, 이 문서 상에는 문자가 아닌 그림으로 그려넣어져 있습니다. 따라서, 이 문서의 예시를 곁어서 실제 프로그램의 입력 프롬프트나 데이터 파일에 붙여넣으면 (때로는 행 레이아웃이 깨지고, 깨지지

기호는 위치에 따라 화살대의 길이가 다를 수 있지만, 화살대의 길이가 아무리 길더라도 화살촉(과 그 끝의 세로 선)이 한 개면 무조건 한 개의 탭 문자를 나타냅니다.

표준공백^{신규} (혹은 **공백^{space}명확**) 표준 키보드의 **Space bar**로 직접 입력할 수 있는 문자(U+0020 ‘ ’ Space).

이 문서에서 표준공백을 시각적으로 명확히 구분해야 할 때에는 그 자리에 ‘³⁾’ 기호를 써서 표시합니다.

공백류^{신규} (혹은 **whitespace^{기존}**) 앞서 언급한 표준공백, 탭, 개행 뿐만 아니라, 세로 탭, form feed, 다양한 폭의 넓은 공백들과 좁은 공백들, 다양한 폭의 짙방 공백(non-breaking space) 들 등을 모두 총칭해서 흔히 “공백류(whitespace)”라고 부릅니다. 다만 이 프로그램에서는, Python 문자열의 .isspace() 메서드가 참을 반환하는 모든 문자들을 (그리고 그런 문자들만) 공백류라고 부르는 것으로 엄밀하게 정합니다.

기획서 개념 안내 때 “기획서는 사용자 관점”, “사용자가 모를 이야기는 배제”, “안쪽의 기술적인 내막은 기획서가 아니라 설계 문서에” 라고 계속 강조했었습니다. 또한 이에 맞게, 위에서 숫자를 정의할 때에는 Python 문자열의 .isdigit() 메서드를 ‘부연 메모’에서만 언급했을 뿐 기획서 본문에서는 언급하지 않았었습니다.

반면에, 지금 여기서는 공백류를 정의하면서 Python 문자열의 .isspace() 메서드를 본문에서 아예 대놓고 언급하고 있습니다. 이 차이는 다음과 같은 이유 때문입니다:

- 숫자의 경우에는 그 용어에 정확히 무엇 무엇이 포함되는지를 ‘표준 키보드’라는 누구나 알만한 소재를 통해 명확히 짚어서 말할 수 있었고, 따라서 사용자가 .isdigit() 같은 기술적인 이야기까지는 굳이 모르더라도 이미 ‘올바른 사용법 (숫자 입력 규칙)’이 확정되어 더이상 혼동할 여지가 없었던 반면,
- 공백류의 경우에는 그 용어가 포괄하는 경계를 사용자에게 제대로 짚어서 알려주려면 부득이 “.isspace() 메서드가 참을 반환하는 모든 문자”라고 기술적인 언급을 할 수 밖에 없었고, 만일 그러지 않으면 사용자 입장에서는 ‘올바른 사용법 (공백류 입력 규칙)’을 제대로 확신할 수 없게 됩니다.

즉, 공백류를 정의하면서 .isspace()를 언급한 것은 여전히 ‘사용자 관점’이 맞고, 여전히 ‘사용자가 알아야 할 이야기’도 맞으며, 여전히 (안쪽의 기술적인 내막이 아니라) ‘겉으로 드러난 사용법’에 해당합니다.

꼭 유념해야 할 요점은 이렇습니다: 기획서에 들어갈 내용과 설계 문서에 들어갈 내용을 구분하기 위해 ‘사용자 관점 vs. 개발자 관점’이나 ‘사용자가 알 만한가 vs. 모를 만한가’를 기준으로 삼을 때, 이 기준의 진짜 의미는 **사용자의 기술적 눈높이(즉, 지식 수준)를 의미하는 게 결코 아니라** 사용자가 (충분히 똑똑하다는 전제 하에) “**알아야만 온전히 제대로 사용할 수 있는가, 몰라도 온전히 제대로 사용할 수 있는가**”를 의미한다는 점입니다.

또한, 이 기획서는 주제 자체가 유동적인 의미규칙이 거의 없고 오직 입력/데이터 문법 규칙과 아주 약간의 고정적인 의미규칙에만 의존하는 주제입니다. 그래서, 문법 규칙을 최대한 자세히 정하기 위해서 문법의 자유도를 높이고자 (일단 다 열어두고, 막을 것만 골라서 막는) 소위 ‘블랙 리스트 방식’으로 문법을 정하고있어서 이렇게 복잡한 정의가 필요해진 것입니다. 즉, 표준공백과 탭, 개행 이외의 다른 다양한 공백들을 모두 ‘처리 대상으로서 고려’하려고 이렇게 정하고 있는 것입니다.

반면에, 여러분의 주제가 만일 유동적인 의미규칙이나 사용 흐름 등 ‘문법 이외의 다른 요소’를 충분히 갖고있어서 문법 자체는 그냥 간단히 정하고싶다면, (일단 다 막아두고, 열 것만 골라서 여는) 소위 ‘화이트 리스트 방식’으로 문법을 정하면 됩니다. 예를 들어, 그냥 초장부터 “키입력/데이터에는 표준 키보드와 표준 입력기^{IME}로 입력 가능한 로마자 대소문자, 숫자, (자모 조합이 완성된) 한글, 공백(스페이스 바), 개행(엔터 키), 탭, 그리고 아래에 나열하는 특수기호들만 사용 가능하며, 그 외의 나머지 모든 문자들은 무조건 전부 오류로 처리합니다.” 라고 써두면 됩니다.

횡공백류^{신규} 공백류 문자들 중에서 (이 문서에서 정의한, 좁은 의미의) ‘개행’ 문자만 제외한 나머지 문자들.

즉, 비록 이름은 ‘가로’를 의미하는 횡(橫) 공백이지만, 여기에는 세로 탭(VT)이나 용지 먹임(FF) 문자도 포함됩니다.

않더라도) 이 기호 자리에는 탭은 커녕 아무것도 들어있지 않게 되니 주의하기 바랍니다.

³⁾ 이 기호는 표준공백을 시각적으로 드러내는 의미로 (특히 컴퓨터 분야에서는) 널리 통용되는 기호지만, 이 기호 ‘문자’ 자체는 표준공백 문자가 아니라 별개의 다른 문자(U+2423 ‘³⁾’ Open Box)입니다. 따라서, 실제 프로그램의 입력 프롬프트나 데이터 파일에 표준공백 대신 이 기호를 입력하면 진짜 표준공백을 입력하는 경우와는 다르게 동작하니, 이 문서의 예시를 굵어서 프로그램이나 데이터 파일에 붙여넣을 때 주의하기 바랍니다.

(문자열의) 길이^{명확} 이 문서에서 ‘문자’란 언제나 유니코드 문자만을 의미하고, ‘문자열’이란 언제나 유니코드 문자들의 나열을 의미합니다. 따라서, 문자열의 길이는 그 속에 들어있는 유니코드 문자들의 실제 갯수로만 세며, 각 글자들의 메모리 상의 용량(즉, 바이트 수)이나 화면 상의 시각적인 폭 정보(W, FW, N, HW, A, Neutral)와는 전혀 무관합니다. 예를 들어, “안녕하세요”와 “Hello”의 길이는 똑같이 5입니다.

공백류열⁰ (혹은 공백류열)^{신규} 오직 공백류^{whitespace} 들로만 구성된 길이 0 이상의 문자열

공백류열¹ ^{신규} 오직 공백류들로만 구성된 길이 1 이상의 문자열

원래 수학/기호논리학/전산학에서 식의 well-formedness나 문자열의 문법, 오토마타 동작 등을 다룰 때에는, 관례적으로

- ‘0개 이상 임의의 갯수/횟수 반복’을 윗첨자 * 기호 (Kleene star) 로,
- ‘1개 이상 임의의 갯수/횟수 반복’을 윗첨자 + 기호 (Kleene plus) 로

각각 나타냅니다. 이는 매우 널리 통용되는 흔한 관례인데다, 정규표현식에도 기본 요소로 들어있기 때문에 각종 편집기의 ‘일괄 치환’ 기능을 본격적으로 활용하려면 늘상 사용하게 되니, 참고로 기억해두기 바랍니다.

다만 이 문서에서는, 혹시 이들 기호에 아직 익숙치 않거나 시각적으로 서로 잘 구별되지 않아 혼동을 일으킬까봐, 의도적으로 (서로 훨씬 잘 구별되는) ⁰ 과 ¹ 로 각각 바꿔두었습니다.

형공백류열⁰ (혹은 형공백류열)^{신규} 오직 형공백류 문자들로만 구성된 길이 0 이상의 문자열

형공백류열¹ ^{신규} 오직 형공백류 문자들로만 구성된 길이 1 이상의 문자열

형공백류열² ^{신규} 형공백류열¹ 중에서 탭 문자가 최소 1 개 이상 포함된 문자열

실상 문자^{신규} (實像文字) 공백류^{whitespace} 가 아니면서 표시가능^{printable} 한⁴⁾ 문자

단어^{변경} 오직 실상 문자들로만 구성된 길이 1 이상의 문자열. 즉, 단어 속에는 공백류가 (그리고 표시불가능한 문자들도) 전혀 들어있지 않습니다.


단어열⁰ (혹은 단어열)^{신규} 0 개 이상의 단어들이 형공백류열¹ 을 경계로 나열되어있는 문자열

단어열¹ ^{신규} 1 개 이상의 단어들이 형공백류열¹ 을 경계로 나열되어있는 문자열

행 (혹은 줄)^{변경} 텍스트 파일의 속에 등장하는, 아래 두 경우 중 하나에 해당하는 문자열:

- 파일의 맨 처음이나 어떤 개행 문자 직후부터, 이후 (무조건) 처음 만나는 개행 문자 직전까지.
- 파일의 맨 처음이나 어떤 개행 문자 직후부터, 이후 개행이 아닌 문자들을 최소 1 개 이상 먼저 만난 다음에 다다르는 파일 끝까지.

위 정의에 따르면 (특히 첫번째 경우 때문에), 개행 문자는 행을 구성하는 (마지막) 문자로 포함되지 않으며 행과 행 사이의 구분자로만 취급됩니다. 이는 행의 통상적인 정의와 다르니 이 문서를 읽는 동안 주의하기 바랍니다.⁵⁾

두번째 경우는 파일의 마지막 행에 개행이 아닌 문자가 하나 이상 있는데 그 끝에서 개행을 하지 않고 (즉, **Enter**  를 누르지 않고) 저장한 경우입니다.⁶⁾ 위 첫번째 경우에는 ‘빈 줄’도 가능했던 반면, 지금

4) 공백류가 아닌데도 여전히 표시불가능한 문자들 중에는 U+0007 Alert (BEL) 같은 문자들이 있습니다.

5) 통상적으로는 텍스트 파일에 입력된 개행 문자들을 그 개행 문자 직전 행의 마지막 문자로 포함시켜서 행의 일부로 취급합니다. 따라서, 소위 ‘빈 행’도 개행 문자 한 개로 구성되며, 행과 행은 사이에 구분자 없이 곧바로 인접해있는 것으로 간주합니다.

하지만, 이런 통상적인 정의를 따르면 끝에 개행 문자가 붙은 행과 안 붙은 행(파일 끝)이 모두 가능해서 문법을 일관되고 간결하게 표현하기 불편해지므로, 이 기획서에서는 모든 행의 문법적 통일성을 위해 이와 같이 변형된 정의를 사용하겠습니다.

6) 텍스트 파일을 다루는 프로그램들 중 대부분은 이런 파일도 아무 이상 없이 처리하지만, 간혹 오래된 프로그램들 중에는 이렇게 일반 문자 뒤에 개행 없이 곧바로 파일 끝에 다다른 경우 마지막 줄 전체를 누락시키는 등의 문제를 일으키는 프로그램들도 있습니다. 따라서, 이 문서에서는 전화번호부 프로그램이 이런 파일을 어떻게 처리하는지 확실히 밝히기 위해 두번째 경우를 명시해둡니다.

이 두번째 경우에는 ‘빈 줄’이 불가능합니다. 즉, 파일이 처음부터 곧바로 끝나거나 어떤 개행 문자 직후에 곧바로 파일이 끝나면 그 사이 영역은 행으로 간주하지 않습니다. 이는 행의 통상적인 정의와 같으며, 더 정확히는 이 프로그램을 구현할 때 사용될 Python 언어의 특성에 맞춰서 기획한 것입니다.

표 1은 위의 두 경우를 시각적으로 잘 드러나도록 네 경우로 나눠서 다시 정리한 것입니다.

시작 위치	사이 (여기에 개행 문자는 불허)	끝 위치
파일 처음	여기에 문자가 있든 없든 행 맞춤	개행 문자 직전
개행 문자 직후	여기에 문자가 있든 없든 행 맞춤	개행 문자 직전
파일 처음	여기에 문자가 있으면 행 맞고, 없으면 행 아님	파일 끝
개행 문자 직후	여기에 문자가 있으면 행 맞고, 없으면 행 아님	파일 끝

표 1: ‘행’인 것과 아닌 것

명령어 (혹은 **명령**)^{특정} 사용자가 7 절의 주 프롬프트에 입력한 문자열 속에서 **단어** 형태로 특정한 위치에 (주로 첫번째, 간혹 두번째 단어로) 등장하면, **프로그램**이 이를 특정한 의미로 간주하도록 예약된 총 59 가지 단어들. 구체적인 목록은 7 절의 표 2에 나옵니다.

어차피 자세한 사항들은 7 절에 가서야 제대로 설명하게 될 용어를 굳이 미리 2절에서 간단히 개념 소개만 하듯이 약속하는 이유는, 이 용어와 아래의 관련 용어들은 그 전에도 여기저기에서 (물론, 자세한 사항은 계속 7 절로 미뤄가며) 자주 쓰이기 때문입니다. 여러분이 기획서를 작성할 때에도, 문서 중후반에 자세히 제대로 설명할 용어라고 하더라도 그 용어가 (자세한 사항은 미룬 채로) 그 앞에서 여러번 사용되면, 그 용어를 미리 초반에 간단히라도 약속해두는 것이 좋습니다.

(명령어의) 동의어^{변경} 어떤 명령어와 서로 완전히 같은 의미로 취급되는 다른 명령어.

동의어군 (혹은 **명령어군**)^{신규} 동의어들끼리만 빠짐없이 모두 모은 묶음. 프로그램에는 총 6 종류의 동의어군이 있으며⁷⁾ 구체적인 분류는 역시 7 절의 표 2에 나옵니다.

표준 명령어 (혹은 **대표 명령어**)^{신규} 다른 명령어의 앞쪽 진부분문자열도⁸⁾ 아니고 한글 자음으로만 구성되어 있지도 않은 명령어들. 각 명령어군마다 3 개씩 (한글 단어 1 개 + 로마자 단어 1 개 + 특수문자 1 개) 총 18 개가 있으며, 구체적인 목록은 7 절의 표 2의 붉은색 명령어들 혹은 표 3의 맨 왼쪽 열에 나오는 명령어들입니다.

단축 명령어^{신규} 다른 명령어의 앞쪽 진부분문자열이거나 한글 자음으로만 구성된 명령어들. 다시 말해서, 표준 명령어가 아닌 명령어들. 만일 어떤 명령어가 다른 어떤 명령어의 앞쪽 진부분문자열이면 이 둘은 반드시 서로 동의어고 (‘도움’ = ‘도움말’), 만일 한글 자음으로만 구성된 어떤 명령어가 한글로만 구성된 다른 어떤 명령어의 초성체면 이 둘은 반드시 서로 동의어입니다 (‘ㄷㅇ’ = ‘도움’). 7 절 표 2의 59 개 명령어들 중 표 3의 18 개를 뺀 나머지가 단축 명령어들입니다.

인자^{특정} 사용자가 7 절의 주 프롬프트에 입력한 문자열 중 문법 형식 두번째 항목의 <단어열¹> 속에 들어있는 각 단어들. 다시 말해서, 첫번째 명령어 뒤에 황공백류열¹ 을 경계로 나열한 단어들이며, 프로그램은 첫번째 명령어가 같더라도 그 뒤의 인자들이 무엇인지에 따라 다르게 동작할 수 있습니다.

7) 결국 서로 다른 의미로 간주되는 명령어는 총 6 종류라는 뜻입니다.

8) ‘앞쪽 부분문자열’이라고 쓰지 않고 ‘앞쪽 **진**부분문자열’이라고 명시했음에 주의하기 바랍니다. 예를 들어, “find”는 비록 “find”의 앞쪽 진부분문자열은 아니지만, 여전히 “find”의 앞쪽 부분문자열이기는 합니다. (모든 문자열은 자기 자신의 부분문자열입니다.)

3 기본 사항

이 절의 내용은 대부분의 소프트웨어 기획서에 대략 비슷한 방식으로 작성되는 내용이고, 이 과목에서도 대부분의 팀들이 간단하라도 작성하기를 권장하는 절입니다. 적어도 ‘어떤 플랫폼에서 작동하는지’, ‘배포된 프로그램에 어떤 파일(들)이 있어야 정상인지’, ‘어떻게 설치하고 실행시키는지’ 정도는 기획서 초반에 써두는 것이 좋습니다.

3.1 작동 환경

기본적으로 Python 3 표준 인터프리터가 설치된 MS Windows 10의 cmd 창, PowerShell 창 및 Python IDLE 창에서 확실히 정상 작동합니다. 그 외에도, 아래 세 가지 요건들을 모두 만족시키면 정상적으로 작동할 수 있습니다:

- Python 3 표준 인터프리터(CPython)가 필요합니다.

Python 3 비표준 인터프리터(PyPy3 등)로도 어쩌면 작동할 수도 있지만 보장되지는 않으며, Python 2 인터프리터로는 분명히 작동하지 않습니다.

이 과목의 (학생들의) 기획서에는 당연히, 다양한 인터프리터/가상머신 버전이나 (아래 항목의) 각 운영체제 버전별, 터미널 종류별로 작동 가능 여부를 일일이 검사하는 것이 현실적으로 불가능하므로, 검사하도록 요구하지도 않습니다. 본인들이 현재 사용하고있는 운영체제와 인터프리터/가상머신 버전에서만 제대로 동작함을 확인하여 명시·보장하고, “그 외의 버전에서는 보장되지 않는다”고 작성하면 됩니다. (“보장되지 않는다”를 남발하지 말라고 안내했는데, 바로 이런 경우가 사용해도 되는 대표적인 경우입니다.)

- 사용자 계정별 홈 경로를 제공하고 환경변수를 통해 이를 특정할 수 있는 운영체제가 필요합니다.

Linux와 MS Windows 10은 분명히 이 요건을 충족시키고, MS Windows Vista, 7, 8, 11과 Apple OS X, macOS도 어쩌면 이 요건을 충족시킬 수도 있지만 보장되지는 않습니다.

- 유니코드 문자집합 입출력이 가능한 문자 기반 터미널이 필요합니다. 단, 터미널 인코딩이 꼭 UTF-8 이어야만 할 필요는 없으며, 어떤 인코딩으로 변환하든 간에 유니코드 문자로 인식할 수만 있으면 됩니다.
 - PuTTY, ConEmu, Gnome Terminal, Konsole 등 대부분의 최신 터미널 에뮬레이터들은 이 요건을 충족시킵니다.
 - gVim 창 내부에서 ‘:terminal’로 띄운 cmd 셸이나 Zsh 프롬프트에서도 (tenc 설정이 아예 부적절하지만 않다면) 전화번호부 프로그램을 정상적으로 작동시킬 수 있습니다. 어쩌면 Emacs GUI 창 내부에서 ‘M-x term’으로 띄운 셸 프롬프트에서도 작동될 수도 있지만, 보장되지는 않습니다.

3.2 프로그램 구성

- 프로그램이 배포될 때에는 주 실행 파일인 phonebook.py 파일 하나만 배포됩니다.
- 프로그램이 올바르게 실행되면, 홈 경로에 “phonebook-data.txt”라는 이름의 데이터 파일이 있는지 확인하고 없으면 생성합니다. 즉, 데이터 파일의 전체 경로는 “{HOME}\phonebook-data.txt” 입니다. 이 파일은 사용자가 직접 수동으로 지우기 전까지는 (즉, 프로그램을 통해서) 지워지지 않습니다.

[Q] 앗! 데이터 파일은 반드시 이렇게 홈 경로에 저장해야 하나요?

[A] 아니요! 이렇게 홈 경로를 활용하는 방식도 존재하고 흔히 사용되는 방식이라는 점을 참고로 보여주려고 써두었을 뿐, 필수 사항은 당연히 아니고 점수에도 반영되지 않습니다. 이 과목에서 여러분이 만들 프로그램은 데이터 파일(들)을 ‘프로그램 실행 파일이 존재하는 경로’나 ‘사용자의 현재 경로’에 저장하고 읽어오면 충분합니다.

단, 데이터 파일을 어디에 저장하든 간에, 그 사실을 (어느 경로에 데이터 파일이 있는지를) **기획서에 반드시 정확히 기재**해야만 하고, 혹시 빼먹고 기재하지 않으면 당연히 감점입니다.

- 프로그램 스스로는 위 데이터 파일 외에 다른 파일을 더 생성하지 않습니다. 혹시 Python 인터프리터가 어딘가에 디렉토리(폴더)나 (주로 .py 확장자의) 파일을 생성할 수도 있지만, 그럴 가능성은 낮습니다.

3.3 프로그램 설치 및 실행

기본적으로, 운영체제와 상관 없이:

- 아무 경로에나 프로그램 주 실행 파일을 복사하여 설치합니다.
- IDLE Editor 창으로 주 실행 파일을 열고, **F5** 키를 누르거나 메뉴에서 Run → Run Module 항목을 누르면 IDLE Shell 창에서 실행됩니다.
- 혹은, 탐색기나 노틸러스 등 GUI 파일 관리창에서 (*.py 파일에 대한 연결 프로그램 설정이 적절하다면) 주 실행 파일을 더블클릭하여 실행할 수도 있습니다.

MS Windows의 경우, 추가로:

- cmd 창, PowerShell 창, 혹은 그에 준하는 터미널에서 주 실행 파일의 절대 경로나 상대 경로를 입력하여 실행할 수 있습니다. 이때 확장자(.py)는 생략할 수도 있습니다.

```
PS D:\work\SoPrj\PhoneBook> .\phonebook[.py] Enter ↵
```

바로 이런 게 mockup이고, 여기서는 ‘화면 입출력과 똑같이 생긴 문자열을 기획서에 (고정폭 글꼴로) 직접 타이핑’하는 방식을 썼습니다. 단, 이렇게 mockup을 ‘직접 타이핑’하는 경우에는 그게 mockup인지 본문 내용인지 한 눈에 헛갈리지 않고 확실히 구별할 수 있도록 표시해주기 바랍니다. (이 문서에서는 모든 mockup에 일괄적으로 회색 테두리를 치고 있습니다.)

- 만일 %PATH% 환경변수에 들어있는 경로에 주 실행 파일을 설치했다면, 그 설치 경로나 현재 경로에 상관 없이 어디서나 주 실행 파일의 파일명만 입력하여 실행할 수 있습니다.

```
PS D:\> phonebook[.py] Enter ↵
```

Linux의 경우(와 어쩌면 macOS도), 추가로:

- 대부분의 터미널에 띄운 대부분의 셸 프롬프트에서, python 명령 뒤에 주 실행 파일의 절대 경로나 상대 경로를 입력하여 실행할 수 있습니다.

```
reeseo@iserlohn:~/work/SoPrj/PhoneBook$ python phonebook.py Enter ↵
```

- 만일 \$PATH 환경변수에 들어있는 경로에 주 실행 파일을 설치했고 주 실행 파일에 자신의 실행 (x) 권한을 설정했다면, 그 설치 경로나 현재 경로에 상관 없이 어디서나 주 실행 파일의 파일명만 입력하여 실행할 수 있습니다.

```
reeseo@iserlohn:~$ phonebook.py Enter ↵
```

4 프로그램 사용 흐름도

사용 흐름도는 너무 늦지 않게 보여주는 것이 좋고, 대체로 용어 정의나 기본적인 설치/실행 방법을 설명한 직후에 (즉, 구체적인 각각의 세부 기능이나 세부 데이터 등을 설명하기 전에) 보여주는 경우가 많습니다. 각각의 세부 기능/데이터보다 전반적인 사용 흐름을 먼저 파악해두면, 이후에 각 세부 기능/데이터를 이해하기도 더 수월해지기 때문입니다.

프로그램 사용 흐름, 즉, 프로그램 사용자 관점에서 프로그램을 사용하는 단계별 경로는 10 쪽의 “**그림 1: 사용 흐름도**”와 같습니다. (데이터 파일을 외부 편집기로 직접 편집하는 방식은 허용된 사용 방식이지만, 프로그램 실행 중의 사용 흐름과는 별개이므로 흐름도에 포함시키지 않습니다.)

순서상 바로 지금 이 (보라색 설명이 있는) 자리가 그림이 들어갈 자리가 맞는데, 간혹 하필 이 자리부터 이 페이지의 아래쪽 끝까지의 남은 공간이 그림의 높이보다 짧으면 (그림은 글과 달리 중간에서 앞뒤 페이지로 잘라넣기 곤란하므로) 이 자리에 그림을 넣지 못하는 경우도 있습니다. 이런 경우에는 관례적으로, 그림을 다음 페이지로 미루고, 그렇게 빈 ‘지금 이 자리’에는 뒤쪽의 ‘글’ 들(지금 이 문서를 예로 들면, 5 절 ‘데이터 요소’ 부분)을 당겨와서 대신 채워넣습니다. 또한, 다음 페이지로 미뤄진 그림은 그 페이지의 (가능한 한) 맨 위나 혹은 (여의치 않으면) 맨 아래에 넣습니다. (아주 드물게, 오히려 그림을 원래 페이지 상단에 넣고 글을 더 뒤로 밀어버리는 경우도 있습니다.) ‘페이지가 잘리는 문서’를 만들 때의 통상적인 관례니, 참고로 기억해두기 바랍니다.

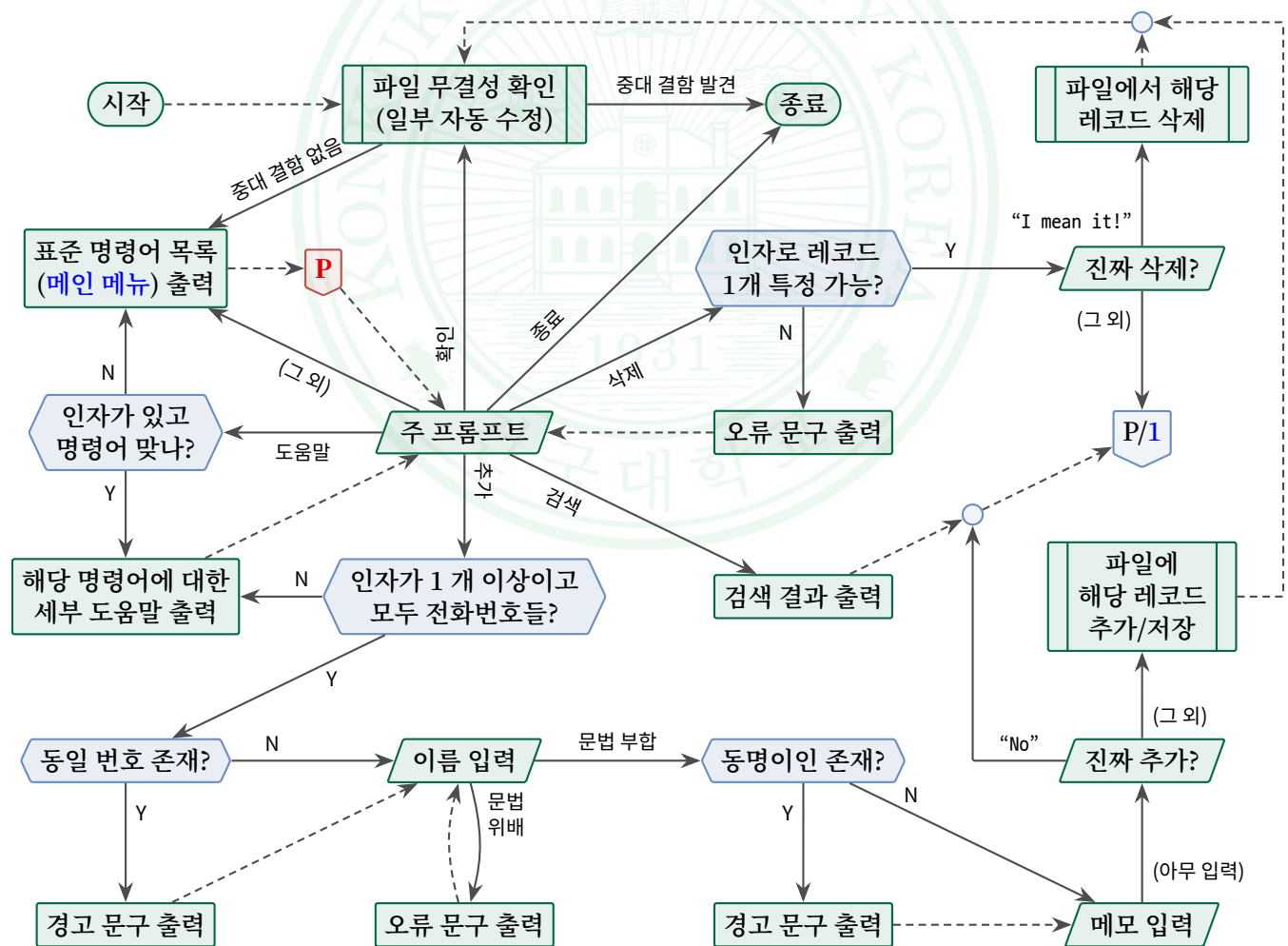


그림 1: 사용 흐름도

5 데이터 요소

데이터 요소에는 크게 이름, 전화번호, 메모가 있습니다.

각 요소마다 동치 비교 규칙도 서로 다르고 검색시의 검색어 합치⁹⁾ matching 규칙도 서로 다르므로, 아래의 각 요소별 소절에서 해당 요소의 동치 비교 규칙, 검색어 합치 규칙을 각각 명시하겠습니다. 단, 정렬시의 순서 비교 방법은 모든 데이터 요소들이 공통적으로 ‘파일에 저장된 문자열 그대로, 사전식 순서로’ 비교됩니다.

5.1 이름

이름은 지인이나 업체의 정식 명칭입니다.

문법 형식: 문법적으로 올바른 이름은 아래 세 조건을 모두 만족시키는 문자열입니다:

- 길이가 1 이상⁹⁾
- 첫 문자와 마지막 문자는 실상 문자¹⁰⁾
- 탭도 개행도 전혀 들어있지 않음

예를 들어, (따옴표 없이) “버거킹 건대역점”과 “Burger King”, “A”, 심지어 “!@#\$”도 모두 문법적으로 올바른 이름입니다.

[Q] 어떤 규칙이나 개념을 설명할 때, ‘정의’와 ‘예시’는 둘 다 필요한가요?

[A] ‘정의’는 반드시 필요하고, ‘예시’는 선택입니다. 즉, 예시 없이 정의만 써두는 건 별 문제 없지만, 정의 없이 예시만으로 규칙을 설명하면 안 됩니다. 또한, 정의와 예시가 둘 다 있을 경우에는 서로 모순 없이 들어맞아야 합니다.

의미 규칙: 이름에는 아무런 추가적인 의미 규칙도 없습니다.

이렇게 ‘추가적인 의미 규칙이 없을 경우’에는, 그냥 이 ‘의미 규칙’ 항목 자체를 아예 빼버려도 논리적으로 틀리진 않지만, 기왕이면 “의미 규칙은 없다”고 확실히 명시하는 게 더 좋습니다.

동치 비교: 두 이름 간의 동치 비교를 할 때에는, 두 문자열 전체가 (로마자 대소문자와 내부의 공백류들까지 포함해서) 서로 완전히 일치해야만 같은 이름으로 간주합니다.

예를 들어, “BillJoy”와 “BillJoy”, “billjoy”는 셋 모두 서로 다른 이름이고, “손승원”과 “손승원”도 서로 다른 이름입니다.

[Q] 기획서 개념 안내 때에는 “동치 비교” 규칙이나 “검색” 규칙도 필요하다는 안내는 없었는데요?

[A] 네, 바로 그래서 “각 팀의 주제/상황에 더 적합한 구성, 형식, 내용, 서술방식을 사용”하라고 계속 강조했던 것입니다. 지금 이 예시 기획서의 주제에는 그 특성상 동치 비교 규칙도 꼭 필요하기 때문에 추가했을 뿐이고, 만일 여러분 주제에는 필요 없다면 추가하지 않으면 그만입니다. (또한 여러분 주제에 필요한 다른 요소를 자유롭게 추가할 수 있다는 뜻이기도 합니다.)

검색: 검색 명령의 인자들 각각을 검색어로 삼아¹¹⁾ 파일에 저장된 이름 중에서 찾을 때에는:

- 저장된 이름 속의 공백류들을 전부 무시한 채로 (즉, 공백류 없이 딱 붙어있는 것으로 간주하고)
- 대/소문자를 구분하지 않으면서
- 검색어가 이름의 부분문자열이면 합치된 것으로 간주합니다.

9) Python 특성상 문자열 길이의 실질적인 최대 한계는 실행 환경과 상황 등에 따라 (비록 이론적인 최대치는 있지만, 그보다 짧게) 달라지므로, 이 문서에서 규정하지 않습니다. 이후 ‘전화번호’와 ‘전화번호 목록’, ‘메모’의 문법을 정할 때도 마찬가지입니다.

10) 길이가 1일 때에는 당연히 첫 문자가 곧 마지막 문자입니다.

11) 인자의 정의상 각각의 인자는 모두 한 단어이므로, 검색은 항상 단어 단위로만 이루어지며 공백류가 포함된 문자열을 검색어로 삼지 못합니다. 예를 들어, “앨런 튜링”이라는 길이 5 짜리 문자열은 단어가 아니므로 이 문자열 전체가 하나의 검색어가 될 수 없습니다.

예를 들어 검색어가 “손승완”일 때, 파일에 저장된 이름들 중 “손승완” 뿐만 아니라 “손_u승완”, 심지어 “윤손승_u완구점”도 모두 합치됩니다. 또한, 검색어가 “Illjo”일 때, 저장된 이름들 중 “Bill_uJoy”와 “Bill_uJoy”, “bill_ujoy”는 셋 모두 이 검색어에 합치됩니다.

참고로, 대/소문자를 구분하지 않는 대상은 대/소문자 개념이 존재하는 거의 대부분의 문자들입니다. 예를 들어, 아래 표의 둘째 줄에서 한 칸에 나오는 두 문자들끼리는 검색시 서로 같은 문자로 취급됩니다.

일반 영문자	움라우트	전각 문자	원 문자	로마 숫자	그리스 문자	키릴 문자	합자	발음 기호	...
A a	Ä ä	A a	Ⓐ ⓐ	III iii	Δ δ	Б б	Æ æ	Ð ð	...

엄밀히는, 만일 두 문자에 대한 Python casefold() 메서드의 리턴값이 서로 같으면 이 프로그램에서 검색할 때에도 서로 같은 문자로 간주합니다. (이 메서드는 유니코드 문자 정보에 명시된 case 변경 표식을 따릅니다.)

5.2 전화번호

이 프로그램은 “전화번호” 요소로 휴대전화 번호와 유선전화 번호를 모두 표현합니다.

문법 형식: 문법적으로 올바른 전화번호는 아래 네 조건 모두에 부합하는 문자열입니다:

- 6 개 이상의 숫자들과 0 개 이상의 ‘-’ 문자들만으로 구성되어야 함
- 마지막 문자는 숫자여야 함¹²⁾
- 처음 세 문자들이 ‘010’이면, 이후 (‘-’는 몇 개든 상관 없고) 숫자는 정확히 8 개가 있어야 함
- 처음 두 문자들이 ‘01’이고 세번째가 ‘0’이 아니면, 이후 (‘-’는 몇 개든 상관 없고) 숫자는 7 개 혹은 8 개가 있어야 함

예를 들어, “02-45099-99”나 “024509999”, “-82-2-450-9999”, “011--345-6789”, “042---99-5555”는 모두 올바른 전화번호지만, “450-99”나 “450-9999-”, “010-345-6789”는 올바른 전화번호가 아닙니다.

의미 규칙: 전화번호에는 아무런 추가적인 의미 규칙도 없습니다. (의도한 모든 규칙을 문법으로 전부 표현했습니다.)

동치 비교: 두 전화번호 간의 동치 비교를 할 때에는, 두 문자열 모두 내부의 ‘-’ 문자들을 전부 무시한 채로, 그 문자열 전체가 서로 완전히 일치해야만 같은 전화번호로 간주합니다.

예를 들어, “450--99-99”와 “4509999”는 서로 같은 전화번호지만, “450-9999”와 “02-450-9999”는 서로 다른 전화번호입니다.

검색: 검색 명령의 인자들 각각을 검색어로 삼아 파일에 저장된 전화번호 중에서 찾을 때에는, 검색어와 저장된 전화번호 모두 내부의 ‘-’ 문자들을 전부 무시한 채로, 검색어가 전화번호의 부분문자열이면 합치된 것으로 간주합니다.

예를 들어, 검색어가 “5099”일 경우, 저장된 전화번호들 중 “010-1111-5099”와 “010-5099-1111” 뿐만 아니라 “02-4509999”와 “02-450-9999”까지도 모두 합치됩니다.

¹²⁾ 첫 문자의 ‘-’는 국제전화 표시를 위해 ‘+’ 대신 활용할 수 있도록 허용했습니다.

5.2.1 전화번호 목록

데이터 요소들 중 ‘이름’, ‘전화번호’, ‘메모’는 서로 (설명은 일부 가져다 쓰더라도) 내용상 의존관계가 없는 요소들이어서 각각을 동격의 소절로 나란히 배치했지만, 지금 이 ‘전화번호 목록’은 ‘전화번호’에 내용상 의존하는 (즉, ‘전화번호’가 없으면 ‘전화번호 목록’도 존재할 수 없게 되는) 요소여서, 전화번호 소절의 하위 소소절로 구성한 것입니다.

물론, 이것 하위 절로 넣지 않고 그냥 4개의 요소들(‘이름’, ‘전화번호’, ‘전화번호 목록’, ‘메모’)을 전부 동격의 소절들로 구성했어도 큰 문제는 없습니다.

전화번호 목록은 연락처 속의 한 사람이나 한 업체에 해당하는 전화번호들의 집합입니다. (서로 다른 사람이나 업체의 전화번호들은 전화번호 목록으로 묶어 부르지 않습니다.)

문법 형식: 문법적으로 올바른 전화번호 목록은 1 개 이상의 **전화번호**들이 1 개 이상의 연이은 **표준공백**들을 경계로 나열되어있는 문자열입니다.¹³⁾ 예를 들어, 다음과 같은 문자열은 문법적으로 올바른 전화번호 목록입니다:

010-1234-5678 02-555-5555 02-450-9999

의미 규칙: 전화번호 목록에는 아무런 추가적인 의미 규칙도 없습니다.

표준형 normal form: 하나의 전화번호 목록 속에는 서로 동치인 전화번호가 중복되어 들어있을 수도 있지만 (즉, 이 경우에도 여전히 문법적으로나 의미상으로나 올바른 전화번호 목록이지만), 이런 경우에는 중복된 전화번호들 중 먼저 (왼쪽에) 등장한 전화번호만 남기고 나중에 (오른쪽에) 등장하는 동치인 전화번호들을 무시한 목록을 표준형으로 삼습니다.

동치 비교: 두 전화번호 목록 간의 동치 비교를 할 때에는 두 목록 모두 표준형으로 바뀌서 이들 표준형끼리 ‘집합으로서의 동치’를 비교합니다. 즉, 두 목록의 표준형이 서로 전화번호 갯수가 같고, 한 목록의 표준형에 들어있는 각각의 전화번호들이 다른 한 목록에도 반드시 들어있다면, 이들 두 전화번호 목록(표준형으로 바꾸기 전의 원본)들은 서로 동치로 간주합니다.

이렇게 개념/규칙의 ‘정의’ 직후에 (흔히 “즉,”으로 시작하는) ‘부연 설명’을 붙일 경우, 반드시:

- 부연 설명 없이 정의만 있더라도 내용상 (이해하기가 다소 어렵긴 해도) 틀리거나 빠진 요소는 없어야 합니다. 정의는 그 자체로 ‘틀리거나 빠진 것 없이 완성된 서술’이어야 하고, 부연 설명은 어디까지나 이해를 돕기 위해서만 존재해야 하기 때문입니다.
- 정의와 부연 설명 간에 서로 충돌하는 모순점이 있어선 안 됩니다.

검색: 검색 명령의 인자들 각각을 검색어로 삼아 파일에 저장된 전화번호 목록 중에서 찾을 때에는, 전화번호 목록 속의 전화번호들 중 어느 하나라도 해당 검색어에 합치되면 그 전화번호 목록 자체가 검색어에 합치된 것으로 간주합니다.

5.3 메모

메모는 동명이인을 구분하거나 추가 정보를 담기 위한 데이터입니다.

문법 형식: 문법적으로 올바른 메모는 아래 세 조건에 모두 부합하는 문자열입니다:

- 길이가 0 이상 (즉, 빈 문자열도 가능)

¹³⁾ 이 정의에 따라, 전화번호 목록의 맨 앞이나 맨 뒤에는 표준공백을 포함한 어떤 공백류도 나올 수 없고, 전화번호 목록은 전화번호로 시작하고 전화번호로 끝납니다. 전화번호와 전화번호 사이에는 표준공백(들)만, 반드시 1 개 이상 등장합니다.

- (길이가 1 이상일 경우에 한해) 첫 문자와 마지막 문자는 실상 문자
- 탭도 개행도 전혀 들어있지 않음

즉, 5.1 절 이름의 문법 규칙과 거의 비슷하되, 메모에는 빈 문자열도 허용된다는 게 유일한 차이입니다.

의미 규칙: 메모에는 아무런 추가적인 의미 규칙도 없습니다.

동치 비교: 두 메모 간의 동치 비교를 할 때에는, 두 문자열 전체가 (로마자 대소문자와 내부의 공백류들까지 포함해서) 서로 완전히 일치해야만 같은 메모로 간주합니다.

검색: 검색 명령의 인자들 각각을 검색어로 삼아 파일에 저장된 메모 중에서 찾을 때에는,

- 저장된 메모 속의 공백류들을 전부 무시한 채로 (즉, 공백류 없이 딱 붙어있는 것으로 간주하고)
- 대/소문자를 구분하지 않으면서
- 검색어가 메모의 부분문자열이면 합치된 것으로 간주합니다.

6 데이터 파일

기본적으로, 이 프로그램은 실행 중에 사용자로부터 연락처 정보를 키 입력으로 받아들이고 이 정보들을 프로그램 스스로 내부 기능을 통해 데이터 파일에 저장합니다. 하지만 그와는 별개로, 사용자가 메모장 등 별도의 텍스트 편집기를 이용해서 데이터 파일을 (전화번호부 프로그램을 통하지 않고) 직접 생성/편집/저장하는 방식도 지원합니다. 단:

- 프로그램이 실행되고 있는 중에 데이터 파일 직접 편집할 경우, 정상 작동을 보장하지 않습니다.
- 데이터 파일은 (BOM 없는) UTF-8 인코딩으로 저장되어야 합니다.
- 운영체제마다 개행 문자들을 다루는 기본 방식이 서로 다르지만 (LF 단독 vs. CR/LF 조합 vs. CR 단독), 이 프로그램은 모든 개행 방식을 똑같이 다룹니다. 즉, 데이터 파일을 어떤 운영체제에서 어떤 개행 방식으로 작성/저장해도 상관 없습니다.
- 이 절에서 정하는 문법과 의미 규칙을 준수하며 편집해야만 프로그램이 정상적으로 동작합니다.

위 마지막 항목에 의거하여, 사용자가 데이터 파일을 올바르게 편집하고 어떤 상황에서 어떤 결과가 나올지 정확히 예상할 수 있도록 데이터 파일의 문법(형식)과 의미 규칙(추가 조건)들을 명시합니다.

6.1 문법 규칙 (형식)

문법 규칙을 정확히 설명하기 위해, 먼저 한 가지 용어부터 정의하겠습니다:

레코드: 다음 두 형식 중 하나에 부합하는 문자열 (단, 화살괄호 <, >는 실제 문자가 아니라 문법 설명용 표식):

- <이름> <링공백류열^㉔> <전화번호 목록>
- <이름> <링공백류열^㉔> <전화번호 목록> <링공백류열^㉔> <메모>

이 용어를 2 절에서 정하지 않고 이제와서 정하는 이유는, 이 용어를 정의하려면 5.1 절의 ‘이름’, 5.2.1 절의 ‘전화번호 목록’, 5.3 절의 ‘메모’가 먼저 확실히 정해져있어야 했기 때문입니다.

이제, **문법적으로 올바른 데이터 파일**이란 그 파일 속의 모든¹⁴⁾ 행들이 다음 두 형식 중 하나에 부합하는 텍스트 파일이라고 정의합니다:

- <항공백류열⁰>
- <항공백류열⁰> <레코드> <항공백류열⁰>

앞서 5.1 절에서 문자열 길이의 최대 한계에 관해 주석으로 언급했듯이, 마찬가지로 파일의 실질적인 최대 행 수도 (운영체제와 파일시스템이 정하는 이론적인 한계는 있지만) 실질적으로는 실행 환경과 실행 시점의 상황들에 따라 달라지며, 이 문서에서 규정하지 않습니다.

사실은 데이터 파일의 문법 정의는 이것으로 끝나고, 확실히 충분합니다. 즉, 아래 6.1.1 절의 설명은 어디까지나 지금까지의 용어 정의와 데이터 요소 정의, 그리고 지금 이 절의 파일 문법 규칙에 대한 종합적인 해설일 뿐이자 이들로부터 모두 유추해낼 수 있는 사항들일 뿐이며, 따라서 **아래 6.1.1 절이 없더라도 문법 규칙은 전혀 달라지지 않는다**는 뜻입니다. 결국 ‘간결함’ vs. ‘친절함’ 간의 trade-off인데, 정답은 없지만 그냥 대체적으로

- 만일 (특히 전산 이론이나 언어론, 혹은 수학) 논문이었다면 같은 내용을 다시 풀어서 해설하기보다는 간결명료한 정의 위주로,
- 만일 (요구자 등) 비전문가에게 보여줄 기획서라면 아래와 같이 좀 더 친절하게 설명을 붙여서

라고 생각하면 됩니다. 이 과목에서는 기본적으로 후자(요구자)를 염두에 두고 훈련하고 있지만, 전자(논문) 스타일의 훈련도 언젠가는 해두면 좋은 훈련이니 지금 전자의 방식을 시도해보는 것도 나쁘지 않습니다.

6.1.1 설명과 예시

5.2 절 ‘전화번호’의 문법 규칙과 5.2.1 절 ‘전화번호 목록’의 문법 규칙상 전화번호 목록에는 탭이 포함될 수 없습니다. 그리고 이름과 메모에도 탭이 포함될 수 없도록 규칙을 정했었습니다. 따라서, 레코드를 구성할 수 있는 세 필드 (필수적^{mandatory}으로 이름과 전화번호 목록, 선택적^{optional}으로 메모) 모두 탭을 포함하지 않습니다. 따라서, 한 레코드 속의 두 (혹은 세) 필드들을 서로 경계지어주는 구분자(혹은 구획 문자)^{delimiter}로서 탭을 사용하기에 적절한 상황입니다.

다만, 실제로는 필드 구분자로 **항공백류열⁰**을 사용함으로써 한 덩어리의 구분자 속에:

- 탭이 한 개만 있어도 되지만 두 개 이상의 탭도 허용하고,
- 탭(들)의 전·후·사이에 다른 (개행만 아니면) 공백류들도 자유롭게 허용합니다.

게다가, 최종적으로 파일의 각 행에는 (레코드가 들어있는 행일 경우) 레코드 앞이나 뒤, 혹은 양쪽에 (개행만 아니라면) 탭을 포함한 임의의 공백류들이 자유롭게 더 붙을 수 있습니다. (필드 사이에는 탭이 최소 1 개 이상 꼭 있어야 하고, 레코드 앞뒤에는 탭이 있어도 그만 없어도 그만이라는 차이만 있습니다.) 이 방식은 세간에 널리 통용되는 TSV^{tab-separated values} (혹은 CSV^{comma-separated values}) 파일에서 필드 구분자로 딱 한 개의 탭 (혹은 쉼표)만 허용하고 레코드 전체의 맨 앞이나 맨 뒤에는 추가 구분자를 허용하지 않는 방식과 다릅니다.

먼저, 이렇게 탭과 (개행만 아닌) 공백류들이 자유롭게 등장하도록 허용한 이유는 사용자가 메모장 등으로 파일을 직접 열어서 편집할 때 각 행마다 자유롭게 각 필드 사이의 간격을 조절하여 파일 내용을 시각적으로 표처럼 보이게 만들^{TABulate} 수 있도록 하기 위해서입니다. 예시로서, 아래 파일은 앞서 정의한 ‘문법적으로 올바른’ 데이터 파일입니다:

¹⁴⁾ ‘모든 행’들이 두 형식 중 하나에 부합하면 문법적으로 올바른 데이터 파일이라고 했으니, 행 자체가 아예 없는 (즉, 빈) 파일은 두 형식 중 하나에 부합하든 말든 상관 없이 무조건 문법적으로 올바른 데이터 파일입니다. (6.1.1 절의 설명 메모 참조)

```

손승완→010-1981-0323→한빛 스타즈
파파존스 피자→02--555-5555→건대점
선우정아→010-1985-0511
정은비→010-1997-0530→짜냥이
강슬기→010-1994-02100002-2014-0801→곰, 깡슬
노을→1989-0510→똥깨멍
서문탁→010-1978-03030342--000-0001→이수진, 기억해줘
김예원→010-1998-0819→오무지
김예원→010-1989-1205→언니, 저...
김유빈→010-1987-1211→김예원, 볼륨을 높여요

```

이렇게 탭을 자유롭게 허용해도 필드 수를 세거나 몇번째 필드인지 짚는 데에 문제가 없는 이유는, 어떤 내용을 담은 어떤 필드들 몇 개가 사용될지 미리 알 수 없는 ‘범용’ 형식인 TSV와는 달리, 이 프로그램의 데이터 파일은 필드 갯수와 그 종류, 그리고 그 값에 포함될 문자까지 우리가 미리 확실히 정해두었고 이를 기준으로 논리적 충돌이 없기 때문입니다. 먼저 행의 처음과 마지막에 관해서는, 우리는 이미

- 이름은 반드시 **실상 문자**로 시작한다
- 전화번호 목록은 반드시 실상 문자로 끝난다
- 메모는 (빈 문자열일 수도 있지만, 빈 문자열이 아니라면) 반드시 실상 문자로 끝난다

는 사실을 알고있고, 이 사실들과 상기 레코드의 정의로부터 결국 “모든 레코드의 유의미한 (비어있지 않은) 필드 값들은 반드시 실상 문자로 시작하고 실상 문자로 끝난다”는 사실을 간단히 유추할 수 있습니다. (메모가 빈 문자열일 경우 메모는 유의미한 값이 아니므로, 마지막 유의미한 값인 전화번호 목록은 실상문자로 끝나는 게 맞습니다.) 따라서, 각 행마다 첫번째 실상 문자보다 앞에 있거나 마지막 실상 문자보다 뒤에 있는 공백류들은 (그 공백류들 중에 심지어 ‘탭’이 포함되어있건 말건) 그냥 가뿐히 무시해버리면 그만입니다.

레코드 속에서 각 필드 사이에 탭과 공백류가 자유로울 수 있는 건, 이름과 전화번호 목록은 빈 문자열일 수 없도록 규정했었고, 유일하게 빈 문자열일 수 있는 메모 필드는 레코드의 가장 마지막에 ‘없어도 되는’ 필드로서 놓여있기 때문에, 탭에 인접한 모든 공백류들과 그 공백류들을 시멘트 삼아 벽돌처럼 연결된 다른 탭들까지 모두 한 덩어리로 취급해버리면 그만이기 때문입니다.

[‘해석’에 관한 부연] 2주차 《주제 선정》 슬라이드에서는 각 입력 문자열에 대해 주로 ‘문법 규칙 → (문법 검사를 통과한 문자열을) 해석(하는 규칙) → 의미 규칙’이라는 흐름으로 진행했던 반면, 이 문서에는 문법 규칙과 의미 규칙 사이에 ‘해석’이 들어있지 않은 경우가 많습니다. (지금 이 6.1.1 절도 어디까지나 문법 규칙에 대한 ‘부연 설명’이지, 문법 규칙을 통과한 파일 내용을 어떻게 ‘해석’할지를 이제와서 새로 정하고 있지 않습니다.) 이유는 크게 두 가지입니다:

- 주제 선정 슬라이드에서 문법 규칙을 정할 때에는 주로 큰 덩어리(예: 문자열 “2021-03-10”)부터 먼저 시작했기에, 의미 규칙을 말하려면 이 큰 덩어리를 작은 토막들로 쪼개서 각 토막을 무엇인가로 간주하는 ‘해석’ 과정이 필요했었습니다. (날짜 표기 문법 규칙 → 연/월/일로 토막내서 해석하는 규칙 → 각 토막별/토막간 의미 규칙)
반면에, 이 문서에서는 애초에 작은 토막부터 먼저 시작하고 이를 이용해서 점점 큰 덩어리로 진행했기에 (문자 및 문자열에 관한 용어 → 데이터 요소 → (데이터 요소들로 이루어진) 목록 → 레코드 → 행 → 파일), 이제와서 굳이 다시 토막쳐서 해석하는 규칙을 정할 필요 없이 이미 자동적으로 해석이 모두 되어있습니다.
- 주제 선정 슬라이드에서는 문법 규칙과 의미 규칙을 비교 설명하기 위해서 가능한 한 의미 규칙이 꼭 필요한 예시들을 보였던 반면에, 이 문서에는 (필요한 조건들이 이미 문법으로 모두 커버되어) 의미 규칙이 굳이 필요 없거나 필요하더라도 잘게 토막쳐서 말할 필요 없는 의미 규칙 뿐인 경우가 많습니다.

따라서, 학생들은 주제 선정 슬라이드의 방식이나 이 문서의 방식을 맹목적으로 준용하기보다는, 각 팀의 실제 주제와 구체적인 상황, 접근 방식, 문서 서술 순서 등에 따라 적절한 방식을 택하기를 권장합니다.

["모든"이라는 표현에 내포된 의미] 앞서 6.1 절의 주석에서 "모든"이라는 표현에 관해 언급했었습니다. "모든"이라는 표현은 항상 "만일 뭔가 있다면, 그들 중 모든"이라는 뜻이며, "만일 아무것도 없다면 뒷말은 들어볼 필요도 없이 자동적으로 참"이라는 뜻을 내포하고 있습니다. 기획서에 "모든"이라는 표현을 쓸 때에는 항상 이 사실을 유념하기 바랍니다.

가장 가까운 예는 '부분집합'의 정의입니다. 부분집합 개념을 정의하는 흔한 방법 중 하나가 "만일 어떤 집합 A의 모든 원소가 B의 원소라면, A를 B의 부분집합이라고 한다."인데, 다들 알다시피 공집합은 임의의 집합의 부분집합입니다.

논리적으로는 보편 양화사 universal quantifier \forall 에 해당하는데, 양화문 $\forall a \in A, B$ 에서 A가 공집합일 때에는 B의 진위에 상관 없이 전체 양화문 $\forall a \in A, B$ 를 참으로 간주하는 것과 같습니다. 또한, 이는 함의문 $(A \rightarrow B)$ 의 가정(A)이 거짓이면 결론(B)의 진위에 상관 없이 무조건 전체 함의문 $(A \rightarrow B)$ 을 참으로 간주하는 vacuous truth에 해당합니다. 예를 들어, "10보다 작고 동시에 100보다 큰 모든 정수들은 3의 배수다" (혹은 같은 의미로, "만일 정수 x 가 10보다 작고 동시에 100보다 크다면, 그런 x 는 항상 3의 배수다")라는 문장은 그런 x 가 존재하지 않기 때문에 무조건 참입니다.

프로그래밍 언어들도 "모든"의 개념을 다룰 때 공통적으로 이렇게 다룹니다. 예를 들어, Python에서 표현식 `all([])`의 평가치는 True입니다. 혹은, 아래 예시도 참고하기 바랍니다:

```
>>> all( isinstance(e, int) for e in [12, 34, "abc", 56] )
False
>>> all( isinstance(e, int) for e in [12, 34, 56] )
True
>>> all( isinstance(e, int) for e in [] )
True
```

6.2 의미 규칙 (추가 조건)

문법적으로 올바른 데이터 파일은 무조건 올바른 데이터 파일입니다. 즉, 올바른 데이터 파일로 간주되기 위해 추가적으로 준수해야하는 조건은 더이상 없습니다.

프로그램을 어떤 내용으로 기획하는가에 따라서는 여기에 다양한 의미 규칙을 추가할 수도 있습니다. 예를 들어, 유선전화는 (집 전화나 사무실 전화 등의 경우처럼) 여러 사람이 공유할 수도 있지만 휴대전화는 (대포폰이 아닌 이상, 실명 인증까지 해야하니) 둘 이상이 공유할 수 없다는 점을 반영하도록 만든다면, "데이터 파일 속의 서로 다른 두 레코드들에는 속에는 서로 같은 '휴대전화' 번호가 존재할 수 없다"는 규칙을 만들 수 있습니다.

한 가지 유념할 사항은, 무엇인가가 올바른 A에 해당하기 위해 갖춰야할 의미 규칙이 정해져있는 상황에서 만일 어떤 대상이 이 의미 규칙에 위배될 경우, 그 대상은 '여전히 A이긴 한데 문제가 좀 있는 것'이 아니라 '아예 A 자체가 아닌 것'이 된다는 점입니다. 이는 마치 문자열이 올바른 A에 해당하기 위해 갖춰야할 문법 규칙이 정해져있는 상황에서 만일 어떤 문자열이 이 문법 규칙에 위배될 경우, 그 문자열은 그냥 'A가 아닌 문자열'로 간주되는 것과 마찬가지입니다.

그래서, 다음 절에서 설명할 '부가 확인 항목'들을 이 의미 규칙에 넣지 않고 굳이 별도로 ('부가 확인 항목'이라는 다른 이름으로) 서술하고 있는 것입니다. 아래 부가 확인 항목들은 혹시 위반하더라도 (자동 수정이나 경고 출력 대상일 뿐) 여전히 올바른 데이터 파일이긴 하기 때문입니다.

6.3 부가 확인 항목

6.3.1 불필요한 중복 전화번호 (표준화)

한 레코드의 전화번호 목록 속에 서로 동치인 전화번호가 두 번 이상 등장하더라도 이 데이터 파일은 여전히 '올바른 데이터 파일'이지만, 이런 중복은 표준화 normalization 대상이며 만일 프로그램이 6.4 절의 '무결성 검사 및 처리' 과정 중에 이를 발견하면 자동적으로 5.2.1 절에서 설명한 '표준형'으로 만듭니다.

6.3.2 동명이인

동일한 이름이 둘 이상의 레코드에 각각 등장하더라도 이 데이터 파일은 여전히 ‘올바른 데이터 파일’이지만, 이런 동명이인은 경고 대상이며

- 새 연락처를 추가하면서 사용자가 입력한 이름이 기존 이름과 같을 경우, 동명이인들 전원의 정보를 출력하고 “메모를 이들과 다르게 작성하세요”라는 경고(안내) 메시지를 출력합니다.
- 6.4 절의 ‘무결성 검사 및 처리’ 과정 중에 동명이인을 발견하면
 - 메모까지 서로 같은 사람이 있을 경우, 동명이인들 전원의 정보를 출력하고 “동명이인들 중 메모까지 동일한 사람들이 있습니다”라는 경고(안내) 메시지를 출력합니다.
 - 메모까지 서로 같은 사람은 없을 경우, 아무 경고 없이 지나갑니다.

6.4 무결성 확인 및 처리

프로그램이 실행될 때, 혹은 주 프롬프트에 확인 명령어를 입력할 때, 혹은 연락처를 추가하거나 삭제한 직후에, 다음과 같이 데이터 무결성을 확인하고 자동 수정 가능한 사항들을 처리하거나 치명적 오류를 알리고 프로그램이 종료됩니다:

1. 사용자의 홈 경로를 파악하려고 시도해서, 파악되면 아무 출력 없이 아래 단계로 넘어가고, 파악되지 않으면 오류 메시지를 출력한 후 프로그램이 즉시 종료됩니다.

```
PS D:\> phonebook Enter
!!! 오류: 홈 경로를 파악할 수 없습니다! 프로그램을 종료합니다.
PS D:\>
```

2. 홈 경로에 데이터 파일이 있는지 확인해서:

- 없으면 경고 문구를 출력고, 홈 경로에 빈 데이터 파일 생성을 시도합니다. 만일 파일 생성에 성공하면 성공 메시지를 출력한 후 다음 단계로 넘어가고

```
PS D:\> phonebook Enter
...! 경고: 홈 경로 C:\Users\reeseo\ 에 데이터 파일이 없습니다.
... 홈 경로에 빈 데이터 파일을 새로 생성했습니다:
C:\Users\reeseo\phonebook-data.txt
```

만일 파일 생성에 실패하면 오류 메시지를 출력한 후 프로그램이 즉시 종료됩니다.

```
PS D:\> phonebook Enter
...! 경고: 홈 경로 C:\Users\reeseo\ 에 데이터 파일이 없습니다.
!!! 오류: 홈 경로에 데이터 파일을 생성하지 못했습니다! 프로그램을 종료합니다.
PS D:\>
```

- 있으면 파일에 대한 입출력 권한이 있는지 확인합니다. 만일 권한이 있으면 아무 출력 없이 아래 단계로 넘어가고, 없으면 오류 메시지를 출력한 후 프로그램이 즉시 종료됩니다.

```
PS D:\> phonebook Enter
```

```
!!! 오류: 데이터 파일
C:\Users\reeseo\phonebook-data.txt
에 대한 입출력 권한이 없습니다! 프로그램을 종료합니다.
PS D:\>
```

3. 데이터 파일을 처음부터 끝까지 전부 읽어서,

- 문법 규칙에 위배되는 행이 한 개 이상 발견되면, 오류 문구와 함께 해당되는 행들을 모두 출력한 후 프로그램을 즉시 종료합니다.
- 한 행 속에서 불필요한 중복 전화번호가 발견되면 6.3.1 절과 5.2.1 절에서 명시한 대로 표준화합니다.
- 동명이인들이 발견되면, 각 조합에 대해 6.3.2 절에서 명시한 대로 (필요에 따라) 경고 메시지를 출력합니다.

4. (경고는 있었더라도) 오류 없이 이 단계까지 오면 무결성 확인/처리가 완료된 것이고, 7 절의 표준 명령어 목록(메인 메뉴)과 주 프롬프트를 각각 출력하고 대기합니다.

7 주 프롬프트

주 프롬프트는 화면에 다음과 같이 출력되고, 사용자의 키 입력을 기다립니다:

```
PhoneBook >
```

문법 형식: 주 프롬프트에 키 입력하는 올바른 문법은 다음 두 형식 중 하나입니다:

- <황공백류열⁰> <명령어> <황공백류열⁰>
- <황공백류열⁰> <명령어> <황공백류열¹> <단어열¹> <황공백류열⁰>

단, 위의 <명령어> 자리에는 반드시 표 2의 59 개 단어들 중 하나를 써야 하며, 이들만이 문법적으로 올바른 ‘명령어’입니다. 표 2에서 같은 칸 속에 들어있는 명령어들끼리는 모두 서로 동의어고, 각 칸은 명령어군(동의어군)을 나타내며, 붉은색 명령어들은 다른 명령어의 축약형이 아닌 표준 명령어들이자 각 명령어군을 나타내는 대표 명령어들이기도 합니다.

help	도움말	quit	종료	verify	확인	find	검색	add	추가	delete	삭제
hel	도움	qui	종	verif	확	fin	검	ad	추	delet	삭
he	도	qu	스	veri	ㅎ	fi	ㄱ	a	ㅎ	dele	스
h	ㅏ	q	스	ver	ㅎ	f	ㄱ		ㅎ	del	스
	ㅏ			ve						de	
?	ㅏ	.		v	!	/		+		d	-

표 2: 올바른 모든 명령어들 (명령어군별)

본문의 두 문법 형식은 인자가 하나 이상 있는 경우와 하나도 없는 경우를 각각 따로 표현한 것입니다. 그래서, 어쩌면 얼핏 보기에는 단어열¹ 대신 단어열⁰을 활용해서 본문의 ‘두 문법 형식 중 하나’를 아래처럼 한 줄로 합칠 수 있겠다는 생각이 들지도 모르는데:

〈윙공백류열⁰〉 〈명령어〉 〈윙공백류열¹〉 〈단어열⁰〉 〈윙공백류열⁰〉

실은 본문의 두 줄짜리 문법과 이 메모의 한 줄짜리 문법은 의미하는 바가 서로 다릅니다. 약간 더 좁혀서 말하자면, 본문의 두 줄짜리 문법으로는 올바르다고 판정될 어떤 입력 문자열이 메모의 한 줄짜리 문법으로는 틀린 입력으로 판정됩니다. 정확히 어떤 종류의 입력이 본문 문법으로는 허용되는데 메모 문법으로는 금지될지 각자 생각해보기 바랍니다. (힌트: 문법을 메모의 한 줄짜리로 정하면, 일부 특수한 경우에만 불편해지는 게 아니라 매우 흔하게 자주 불편해집니다.)

해석: 명령어는 이미 문법 형식에 구분되어 쓰여있으므로 해석 규칙이 필요 없습니다. 〈단어열¹〉은 (만일 있다면) 그 속의 **단어**들 각각을 인자들로 간주합니다.

[[기획서용] 해석 ‘규칙’ vs. (설계 문서용) 해석 ‘절차/계획’] 위 해석 규칙은 (‘무엇’이 명령어고) ‘무엇’이 인자인지 지목하는 규칙으로서, 사용자도 알아야하는 규칙이므로 기획서에 들어갈 내용이 맞습니다. 반면에, 아래에 나열하는 내용은 ‘이 규칙을 기계에게 어떻게 시킬 것인가’ 하는 구현 전략(Python 특성 상 거의 코딩에 가까워졌지만)에 해당하며, 이는 기획서가 아니라 **설계 문서에 들어갈 내용**입니다 (즉, 구별/비교를 위한 반례입니다):

1. 입력받은 문자열 `s`를 `s.split()`하여 문자열 (단어)들의 리스트 `ws`를 만들고
2. `cmd, args = ws[0], ws[1:]` 를 시도(`try:`)하여 (성공하면) 명령어 `cmd`와 인자들의 리스트 `args`를 얻되,
3. 만일 위 단계가 실패(`except: IndexError`)하면 문법 오류로 간주합니다.

의미 규칙: 〈명령어〉 자리에 어떤 명령어군에 속한 명령어가 들어오는지에 따라서, 위의 〈단어열¹〉이 허용되는지 여부와 (허용된다면) 그 속에 어떤 **인자** 몇 개가 쓰일 수 있는지도 달라집니다. 표 3은 각 명령어군별 인자 규칙과 설명으로서, 맨 왼쪽 열에는 표 공간 문제상 각 명령어군을 나타내는 대표 명령어 3 개씩만 쓰여있지만, 실제로는 해당 명령어군에 속한 어떤 동의어를 사용하는 경우라도 마찬가지입니다.

명령어군	올바른 인자들	설명
? help 도움말	없거나, 있다면 명령어 1 개	전체 혹은 명령어별 도움말을 출력합니다
. quit 종료	없음	프로그램을 종료합니다.
! verify 확인	없음	데이터 무결성 확인 및 처리를 진행합니다.
/ find 검색	자유	데이터 파일에서 인자들을 검색합니다.
+ add 추가	전화번호 1 개 이상	데이터 파일에 새 레코드를 추가합니다.
- delete 삭제	자유	데이터 파일에서 기존 레코드를 지웁니다.

표 3: 명령어군별 대표 명령어들과 인자 규칙 및 설명

문법 오류 결과: 만일 주 프롬프트에서 그냥 곧바로 **Enter** 키만 누르거나 (즉, 빈 문자열 입력), 공백류 (들)만 입력하거나, 입력 중 첫번째 단어가 명령어가 아닐 경우, 틀린 입력으로 간주하고 (“잘못된 입력입니다.” 같은 진부한 안내 없이, 조용히) 표 3에 준하는 표준 명령어 및 인자 안내 화면을 출력하고 주 프롬프트로 되돌아갑니다. 프로그램의 사용 흐름 상, 이 출력은 통상적인 ‘메뉴 번호 선택’ 방식으로 진행되는 프로그램의 ‘메인 메뉴’에 해당합니다.

PhoneBook > 뭐냐 이건? 어떻게 쓰는 거냐? **Enter**

명령어군	올바른 인자들	설명
? help 도움말	없거나, 명령어 1개	전체 혹은 명령어별 도움말 출력
. quit 종료	없음	프로그램 종료
! verify 확인	없음	데이터 무결성 확인 및 처리

/	find	검색	자유	데이터 파일에서 인자들 검색
+	add	추가	전화번호 1개 이상	데이터 파일에 새 레코드 추가
-	delete	삭제	자유	데이터 파일에서 기존 레코드 삭제

PhoneBook >

의미 오류 혹은 정상 결과: 만일 입력 중 첫번째 단어가 명령어가 맞으면, 표 3의 의미 규칙을 기준으로 의미 오류 혹은 정상으로 판정합니다. 두 경우 모두 프로그램의 구체적인 동작 방식은 이후의 각 명령어군별 소절에서 명시하겠습니다.

7.1 도움말 명령어군

인자에 따라 표준 명령어 목록이나 특정 명령어의 세부 도움말을 보여줍니다.

인자 문법 형식: 이 명령은 인자가 0 개 혹은 1 개여야 하고, 1 개일 경우 그 유일한 인자가 표 2의 59 개 명령어들 중 하나여야 합니다.

인자 의미 규칙: 위 문법 규칙을 만족시키는 한, 추가적인 의미 규칙은 없습니다.

7 절에서는 각 명령어별 인자 규칙을 문법 규칙이 아닌 의미 규칙으로 서술했었는데, 이는 ‘주 프롬프트’에 다양한 명령어들을 입력하는 넓은 관점에서 그렇게 서술했던 것이었습니다. 반면에 ‘도움말 명령의 인자’라는 관점으로 좁혀서 보면, “정해진 59 개 문자열 중 하나여야 한다”라는 내용은 (비록 단순 무식한 ‘원소 나열법’이긴 해도 어쨌든 ‘문자열의 구성 문자 규칙’을 말한 것이긴 하므로) 의미 규칙이 아닌 문법 규칙으로 보는 게 더 합당합니다.

비정상 결과: 만일 인자가 2 개 이상이거나, 1 개지만 그 인자가 표 2의 59 개 명령어들 중 하나가 아닐 경우, 상응하는 오류 메시지를 출력한 후 곧바로 7 절의 ‘문법 오류 결과’에서 출력했던 표준 명령어 및 인자 안내 화면을 출력하고 주 프롬프트로 돌아갑니다.

```
PhoneBook > ☐○☐ find add Enter ↵
.!! 오류: 인자가 너무 많습니다. 최대 1개의 명령어만 인자로 입력하세요.

: (7 절 문법 오류시 출력한 메인 메뉴와 동일)
PhoneBook > ☐○ 독심술 Enter ↵
.!! 오류: "독심술"이라는 명령어는 없습니다.

: (7 절 문법 오류시 출력한 메인 메뉴와 동일)
PhoneBook >
```

정상 결과 1: 만일 인자가 없으면 7 절의 ‘문법 오류 결과’에서 출력했던 표준 명령어 및 인자 안내 화면을 출력하고 주 프롬프트로 돌아갑니다.

```
PhoneBook > ? Enter ↵

: (7 절 문법 오류시 출력한 메인 메뉴와 동일)
PhoneBook >
```

정상 결과 2: 만일 인자가 1 개만 있고 이 인자가 명령어면 해당 명령어에 대한 약간 더 자세한 사용법을 출력하고 주 프롬프트로 돌아갑니다.

```
PhoneBook > □○□ / Enter ↵
... 데이터 파일에 저장된 연락처 정보에서 인자들을 검색하는 명령어
동의어: / 검색 검 ㅏㅓ ㅓ find fin fi f
인자: 0 개 이상 임의의 갯수의 아무 단어(들)
동작: * 인자가 없으면 저장된 모든 연락처를 출력합니다.
      * 인자가 1 개면 그 인자가 포함된 연락처들을 모두 출력합니다.
      * 인자가 2 개 이상이면 각 인자가 모두 포함된 연락처를 출력합니다.
PhoneBook >
```

7.2 종료 명령어군

프로그램을 종료합니다.

문법 형식: 이 명령은 인자를 허용하지 않으며, 반드시 명령어 단독으로만 사용해야 합니다.

의미 규칙: 이 명령은 문법 형식 외에 더 준수해야할 의미 규칙이 없습니다.

비정상 결과: 만일 문법 형식에 위배되면 (인자가 존재하면) 문법에 맞지 않는다는 오류 메세지만 출력하고 주 프롬프트를 다시 출력합니다.¹⁵⁾

```
PhoneBook > quit now Enter ↵
.!! 오류: 인자가 없어야 합니다.
PhoneBook >
```

정상 결과: 명령이 올바르게 입력되면 즉시 프로그램을 종료합니다.

```
PhoneBook > . Enter ↵
PS D:\>
```

7.3 무결성 확인/처리 명령어군

데이터 파일의 무결성을 확인하고 처리합니다.

문법 형식: 이 명령은 인자를 허용하지 않으며, 반드시 명령어 단독으로만 사용해야 합니다.

의미 규칙: 이 명령은 문법 형식 외에 더 준수해야할 의미 규칙이 없습니다.

비정상 결과: 만일 문법 형식에 위배되면 (인자가 존재하면) 문법에 맞지 않는다는 오류 메세지만 출력하고 주 프롬프트를 다시 출력합니다.¹⁶⁾

정상 결과: 문법에 맞게 명령이 입력되면 6.4 절의 데이터 무결성 검사 및 처리를 즉시 수행합니다. 수행 결과에

¹⁵⁾ 그냥 인자를 무시하고 정상 종료하는 대신 굳이 오류로 처리하는 이유는, 예를 들어 원래 의도했던 입력은 연락처 추가 명령인

```
PhoneBook > a 010-1234-5678 02-123-4567 Enter ↵
```

였는데 a를 q로 잘못 눌러서 q에 인자가 들어간 경우, 프로그램이 종료되지 않고 곧바로 주 프롬프트를 다시 제공하기 위함입니다.

¹⁶⁾ 앞의 q 명령과 마찬가지로 이유입니다.

따라 상응하는 경고나 오류 문구가 표시되며, (6.4 절의 규칙에 따라) 주 프롬프트로 다시 돌아갈 수도 있고 프로그램이 종료될 수도 있습니다.

7.4 검색 명령어군

문법 형식: 이 명령은 인자가 몇 개여도 상관 없고, (만일 있다면) 각각이 어떤 문자열이어도 상관 없습니다. 즉, 문법 규칙이 전혀 없습니다.

혹시 여기서 ‘음? 이렇게까지 말한다면, 인자 속에 이상한 공백류나 희안한 제어문자 같은 게 포함되는 특수한 상황들은 어쩌고?’라고 생각할 수도 있겠지만, 앞에서 이미 “인자는 오직 실상 문자로만 구성된다”고 명확히 정해두었고 실상 문자가 무엇인지도 명확히 하기 때문에, 일단 ‘인자’라고 해석되고 난 후부터는 이런 걱정을 할 필요가 전혀 없습니다. 바로 이렇게 후반부에 여러 서술들을 두루두루 간결 명료하게 만들기 위해서 2 절에서 그렇게 ‘빡세계’ 용어 정의 작업을 해두었던 것입니다.

의미 규칙: 이 명령은 인자가 준수해야 할 의미 규칙도 전혀 없습니다.

비정상 결과: 이 명령은 문법 규칙도 의미 규칙도 없으므로, 어떤 경우에도 비정상 결과를 내놓지 않습니다.

정상 결과: 인자의 유무와 구성에 따라 동작이 달라지지만, 모두 정상 결과입니다. 만일 인자가:

- 딱 1 개면, 이 인자를 ‘검색’했을 때 합치되는 데이터 요소가 들어있는 모든 레코드들을 찾아서 이름 순, 메모 순, 레코드 위치 (행) 순으로 정렬하여 출력하고 주 프롬프트로 돌아갑니다. 각 데이터 요소별로 ‘검색시 무엇을 합치된 것으로 간주할 것인가’가 다른데, 이는 5.1 절, 5.2 절 (및 5.2.1 절), 5.3 절의 “검색” 항목에 각각 명시되어 있습니다.

```
PhoneBook > / 김예 Enter ↵
==>   이름: 김 예원
      메모: 언니, 저...
      전화번호: 010-1989-1205
==>   이름: 김 예원
      메모: 오무지
      전화번호: 010-1998-0819
==>   이름: 김 유빈
      메모: 김예원, 볼륨을 높여요
      전화번호: 010-1987-1211
PhoneBook >
```

- 2 개 이상이면, 각각의 인자를 위 첫번째 항목의 규칙대로 검색한 결과들의 교집합만을 이름 순, 메모 순, 레코드 위치 (행) 순으로 정렬하여 출력하고 주 프롬프트로 돌아갑니다. 즉, 모든 인자에 전부 합치되는 레코드만으로 검색 결과가 좁혀집니다.

```
PhoneBook > / 김예 언니 Enter ↵
==>   이름: 김 예원
      메모: 언니, 저...
      전화번호: 010-1989-1205
PhoneBook >
```

- 0 개면 (즉, 없으면), 데이터 파일 속의 모든 레코드들을 전부 이름 순, 메모 순, 레코드 위치 (행)

순으로 정렬하여 출력하고 주 프롬프트로 돌아갑니다.¹⁷⁾

7.5 추가 명령어군

문법 형식: 이 명령은 인자가 반드시 1 개 이상 있어야 하고, 모든 인자들이 [전화번호](#)의 문법 형식에 부합해야 합니다.

의미 규칙: 이 명령은 문법 형식 외에 더 준수해야 할 의미 규칙이 없습니다. 단, 입력 자체를 ‘잘못된 인자’로 판정하고 거부하는 의미 규칙은 없지만, 자동 처리나 경고 문구 출력을 위한 부가 확인 항목은 더 있습니다.

부가 확인 항목 1: 인자가 2 개 이상을 경우, 표준공백 1 개를 경계로 각 인자들을 이어붙여 만든 [전화번호 목록](#) 속에 [6.3.1 절](#)에서 설명한 ‘불필요한 중복’이 있는지 확인합니다.

부가 확인 항목 2: 데이터 파일의 기존 레코드들 중에, 인자들 중 하나와 ‘동치’인 전화번호를 이미 갖고있는 레코드가 있는지 확인합니다.

비정상 결과: 인자가 문법 형식에 위배되면 추가 명령어군에 대한 상세 도움말을 (즉, “help add” 입력에 대한 정상 출력 결과를) 출력하고 주 프롬프트로 돌아갑니다.

```
PhoneBook > ➔ 손승완 Enter ↵
... 데이터 파일에 새 연락처를 추가/저장하는 명령어
동의어: + 추가 추 ➔ ➔ ➔ add ad a
인자: 1 개 이상 임의의 갯수의 전화번호(들)
동작: 이름과 메모를 입력받고, 인자들을 전화번호 갖는 연락처를 추가합니다.
PhoneBook >
```

정상 결과: 입력된 인자들이 문법 규칙을 만족시키면:

1. 부가 확인 항목 1 번을 통해 중복을 발견하면, 즉시 [5.2.1 절](#)의 ‘표준형’ 규칙대로 표준화합니다.
2. 부가 확인 항목 2 번을 통해 기존 번호와 동치인 번호들을 발견하면, “새 연락처와 같은 전화번호를 공유하게 될 기존 연락처들이 있습니다”라는 경고(안내) 메시지를 출력하고 해당되는 기존 연락처들을 모두 출력합니다.
3. 첫번째 부 프롬프트를 출력하고 이름을 입력받습니다. 이에 관해서는 [7.5.1 소절](#)에서 부연합니다.
4. 입력받은 이름과 ‘동치’인 레코드가 이미 있는지 확인해서, 만일 있으면 해당되는 기존 연락처들을 모두 출력하고 “동명이인이 있으니 메모를 다르게 입력하길 권장합니다”라는 경고(안내) 메시지도 출력합니다.
5. 두번째 부 프롬프트를 출력하고 메모를 입력받습니다. 이에 관해서는 [7.5.2 소절](#)에서 부연합니다.
6. 세번째 부 프롬프트를 출력하고 저장 여부를 최종 확인합니다. 이에 관해서는 [7.5.3 소절](#)에서 부연합니다.

7.5.1 부 프롬프트 1: 이름 입력

부가 확인 항목 1 번과 2 번을 모두 확인/처리하고 나면, 이름을 키 입력하는 부 프롬프트가 출력됩니다:

¹⁷⁾ 빈 문자열은 모든 문자열의 부분문자열이므로, 모든 레코드의 모든 데이터 요소에 합치됩니다.

```
PhoneBook > + 010-1994-0221 02-2014-0801 01019940221 Enter ↵
..! 입력한 전화번호 목록에 중복이 있어서 다음과 같이 표준화했습니다:
    010-1994-0221 02-2014-0801
..! 새 연락처와 같은 전화번호를 공유하게 될 기존 연락처들이 있습니다:
==>    이름: 강 슬기
        메모: 곰, 깡슬
        전화번호: 010-1994-0210 02-2014-0801
PhoneBook: 새 연락처의 이름 >
```

문법 형식: 5.1 절의 이름 문법 규칙에 맞는 문자열을 <이름>이라고 할 때, 문법적으로 올바른 키 입력 문자열은 다음 형식에 부합하는 문자열입니다:

<형공백류열⁰><이름><형공백류열⁰>

의미 규칙: 이 입력은 문법 형식 외에 더 준수해야 할 의미 규칙이 없습니다.

부가 확인 항목: 데이터 파일에 동명이인이 있는지 확인합니다.

비정상 결과: 이름이 문법 형식에 위배되면 그에 상응하는 오류 메시지를 출력하고 이름 입력 부 프롬프트를 다시 출력합니다.

```
PhoneBook: 새 연락처의 이름 > Enter ↵
..! 이름은 빈 문자열일 수 없습니다.
PhoneBook: 새 연락처의 이름 > ␣→␣ Enter ↵
..! 이름은 (각종) 공백류들로만 구성될 수 없습니다.
PhoneBook: 새 연락처의 이름 > 손→승완 Enter ↵
..! 이름의 유효한 첫 글자와 유효한 마지막 글자 사이에는 탭이 들어갈 수 없습니다.
PhoneBook: 새 연락처의 이름 >
```

정상 결과: 인자가 문법 형식에 부합하면 부가 확인 항목(동명이인)의 결과에 따라 적절한 경고 메시지를 출력하고 다음 단계인 메모 입력 단계로 넘어갑니다.

7.5.2 부 프롬프트 2: 메모 입력

이름을 올바르게 입력하고 부가 확인 항목을 확인/처리하고 나면, 메모를 키 입력하는 부 프롬프트가 출력됩니다:

```
PhoneBook: 새 연락처의 이름 > ␣→손␣승완→␣ Enter ↵
..! 동명이인이 있으니 메모를 다르게 입력하길 권장합니다:
==>    이름: 손 승완
        메모: 한빛 스타즈
        전화번호: 010-1981-0323
PhoneBook: 새 연락처의 메모 >
```

문법 형식: 5.3 절의 이름 문법 규칙에 맞는 문자열을 <메모>라고 할 때, 문법적으로 올바른 키 입력 문자열은 다음 형식에 부합하는 문자열입니다:

〈횡공백류열⁰〉〈메모〉〈횡공백류열⁰〉

메모는 빈 문자열일 수도 있기 때문에, 결국 어떤 키 입력 문자열이라도 모두 올바릅니다.

의미 규칙: 이 입력은 문법 형식 외에 더 준수해야 할 의미 규칙이 없습니다. (6.3.2 절에서 정의하고 앞서 이름 입력 직후에 확인했던 ‘동명이인일 경우 메모를 다르게 한다’는 부가 확인 항목은 어디까지나 경고 대상일 뿐 오류가 아닙니다.)

비정상 결과: 메모는 어떤 문자열을 입력하더라도 비정상 결과가 나오지 않습니다.

정상 결과: (빈 문자열이나 공백류열을 포함하여) 아무 문자열이나 입력하면 다음 단계인 저장 확인 단계로 넘어갑니다.

7.5.3 부 프롬프트 3: 저장 확인

메모까지 입력하고 나면, 지금까지 입력한 전화번호(들), 이름, 메모를 다시 출력하고 저장할지 여부를 묻습니다:

```
PhoneBook: 새 연락처의 메모 > 완모예드 Enter ↵
... 현재까지 입력한 내역입니다:
==>   이름: 손 승완
      메모: 완모예드
      전화번호: 010-1994-0221 02-2014-0801
PhoneBook: 정말 저장하시겠습니까? (.../No) >
```

문법 형식: 어떤 키 입력 문자열도 모두 문법적으로 올바릅니다.

해석: 키 입력 문자열이 “No”라는 2 글자에 정확히 (앞·뒤 공백류도 없이, 대·소문자도 똑같이) 일치해야 부정으로 해석하며, 그 외의 모든 입력은 (그냥 Enter ↵만 눌러 입력한 빈 문자열도 포함해서) 전부 긍정으로 해석합니다.

의미 규칙: 별도의 의미 규칙은 없습니다.

비정상 결과: 이 입력의 결과로는 어떤 경우에도 비정상 결과가 나오지 않습니다.

정상 결과: 입력한 답을 해석한 결과가:

- 긍정이면, 지금까지 입력했던 연락처 내역을 레코드 형식의 문자열로 만들어서 파일 맨 끝에 새 줄로 추가하여 저장합니다. 그 후 곧바로 6.4 절의 파일 무결성 확인/처리를 수행하고 그 결과에 따라 이후 단계를 진행합니다.
- 부정이면, 지금까지 입력했던 연락처 내역을 그냥 폐기하고 주 프롬프트로 돌아갑니다.

7.6 삭제 명령어군

문법 형식: 이 명령은 인자가 몇 개여도 상관 없고, (만일 있다면) 각각이 어떤 문자열이어도 상관 없습니다. 즉, 문법 규칙이 전혀 없습니다.

의미 규칙: 인자들을 검색 명령의 인자들로 간주하고 검색했을 때, 그 검색 결과가 딱 한 건으로 특정되어야 합니다.

공통 중간 결과: 인자들이 의미 규칙에 부합하든 위배되든 간에, 인자들을 검색어들로 삼아서 검색한 결과를 화면에 출력합니다.

비정상 결과: 인자가 의미 규칙에 위배되면, 즉, 검색 결과가 없거나 두 건 이상이면, “삭제 대상을 특정할 수 없습니다”라는 오류 메시지와 검색 결과를 출력하고 주 프롬프트로 돌아갑니다.

```
PhoneBook > - 손승완 Enter ↵
.!! 삭제할 대상을 하나로 특정할 수 없습니다:
==>   이름: 손 승완
      메모: 완모예드
      전화번호: 010-1994-0221 02-2014-0801
==>   이름: 손 승완
      메모: 한빛 스타즈
      전화번호: 010-1981-0323
PhoneBook >
```

정상 결과: 인자가 의미 규칙에 부합하면, 즉, 검색 결과가 딱 한 건으로 특정되면, 부 프롬프트를 출력하고 해당 레코드의 삭제 여부를 최종 확인합니다. 이에 관해서는 7.6.1 소절에서 부연합니다.

7.6.1 부 프롬프트: 삭제 확인

삭제할 레코드가 특정되면, 해당 레코드의 정보를 출력하고 정말 삭제할지 물어봅니다:

```
PhoneBook > - 한빛 손승완 Enter ↵
... 삭제할 대상이 하나로 특정되었습니다:
==>   이름: 손 승완
      메모: 한빛 스타즈
      전화번호: 010-1981-0323
PhoneBook: 정말 삭제하시겠습니까? (I mean it!/) >
```

문법 형식: 어떤 키 입력 문자열도 모두 문법적으로 올바릅니다.

해석: 키 입력 문자열이

I mean it!

이라는 10 글자에 정확히 (앞·뒤 공백류도 없이, 대·소문자와 느낌표도 똑같이) 일치해야 긍정으로 해석하며, 그 외의 모든 입력은 (그냥 Enter ↵ 만 눌러 입력한 빈 문자열도 포함해서) 전부 부정으로 해석합니다.

의미 규칙: 별도의 의미 규칙은 없습니다.

비정상 결과: 이 입력의 결과로는 어떤 경우에도 비정상 결과가 나오지 않습니다.

정상 결과: 입력한 답을 해석한 결과가:

- 긍정이면, 특정된 레코드를 데이터 파일에서 삭제한 후 6.4 절의 파일 무결성 확인/처리를 수행하고, 그 결과에 따라 이후 단계를 진행합니다.
- 부정이면, 그냥 주 프롬프트로 돌아갑니다.

8 기타 처리

이 프로그램은 앞서 명세한 사항들 외에 추가적으로 처리하는 일이 없습니다.

이 절은 필수 사항이 아니라, 그저 기획서 형식의 다양성을 예시하기 위해 괜히 추가한 절입니다. 기획서의 취지와 품질에 부합하는 한, 각 팀의 필요에 따라 자유롭게 필요한 절을 추가/삭제/변경할 수 있다는 점만 참고하면 됩니다. 다시 한 번 강조하지만, **예시는 어디까지나 예시일 뿐임을 잊지 말기 바랍니다.**

