**알고리즘 설계 과제 1**

**알고리즘 001분반 [김영호 교수님]**

**jk.park.tech@gmail.com**

**12150981 박중규**

**1. 개요**

**- 설계의 목적**

레드블랙트리(red-black tree)를 이용하여 병원의 환자 정보를 호율적으로 관리하는 프로그램 설계

**- 요구사항**

* 레드블랙트리 자료구조 이용
* 주어진 환자 정보 형식에 따라 데이터 저장
* 수행할 기능(신규 가입, 환자 검색, 추가 진료, 유행병 조사)들의 구현

**- 개발 환경**

* OS : Windows 10(64-bit)
* IDE : Visual Studio 2019
* Language : C++

**2. 필요한 자료구조 및 기능**

**- 필요한 자료구조**

* **<vector> STL**  
  - RBT의 node list에 node를 동적으로 삽입하기 위하여 사용  
  - 각 환자의 질병을 동적으로 삽입하기 위하여 사용
* **‘node’ 클래스**  
  - 이진트리를 구현하기 위해 사용  
  - 부모와 왼쪽 자식, 오른쪽 자식의 주소를 담는 변수들 사용  
  - 환자의 정보들을 담을 변수들 사용
* **‘RBT’ 클래스**  
  - 모든 node가 담겨있는 node list 사용  
  - root 노드를 나타내는 변수 사용  
  - ‘신규 가입’ 관련 메서드  
  insert, insertForSearch, checkDoublered, restructuring, recoloring  
  - ‘환자 검색’ 관련 메서드  
  search  
  - ‘유행병 조사’ 관련 메서드  
  epiForSearch  
  - 노드의 깊이를 구하기 위한 메서드  
  printDepthAndInfo, returnDepth

**3. 기능별 알고리즘 명세**

**- 기능을 위한 알고리즘 설명(복잡도 분석)**

<복잡도 분석을 위한 가정>

트리의 모든 노드의 개수 : n

트리의 높이 : d = log n(∵ RBT는 balanced tree이므로)

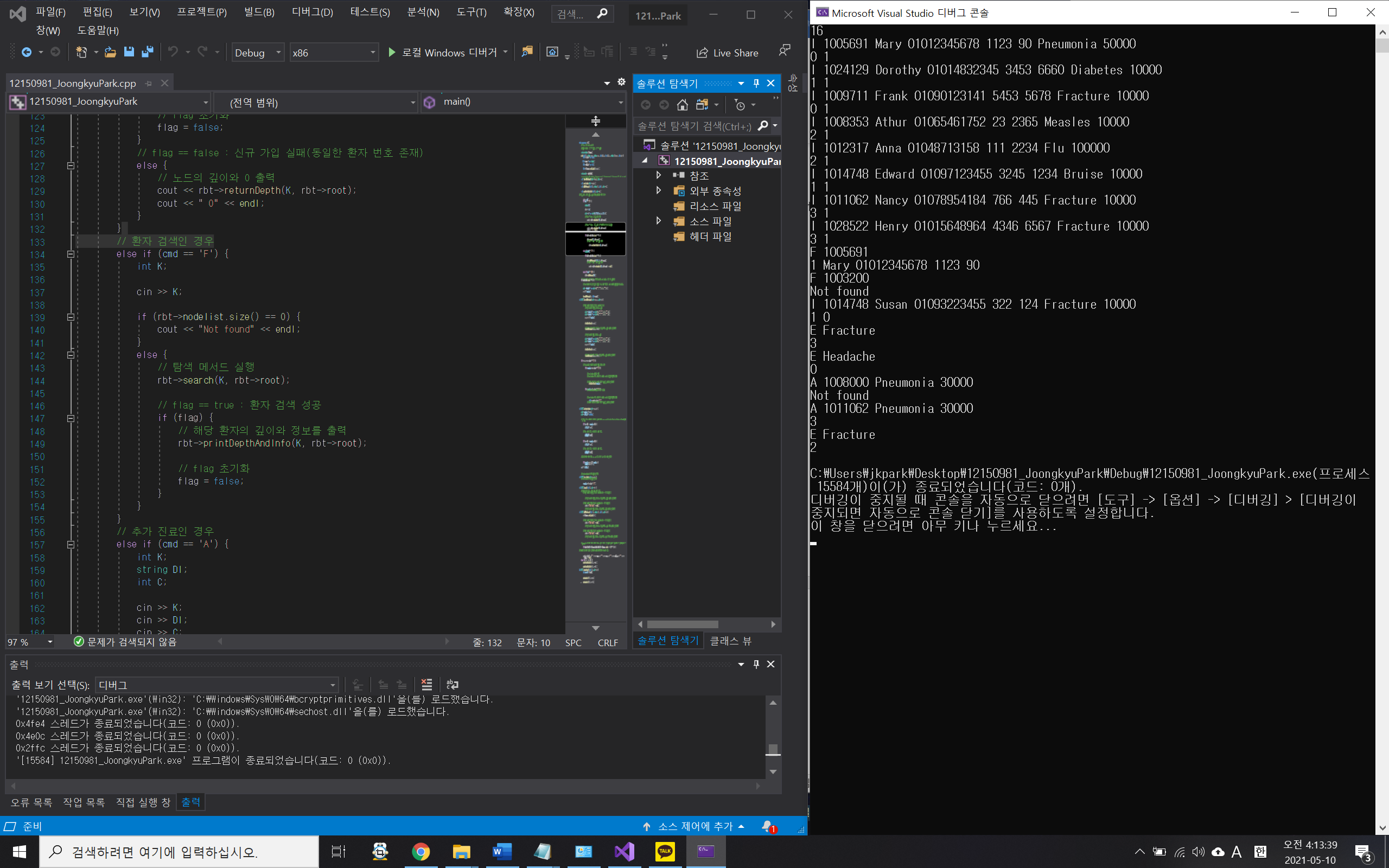
Basic operation : 비교연산

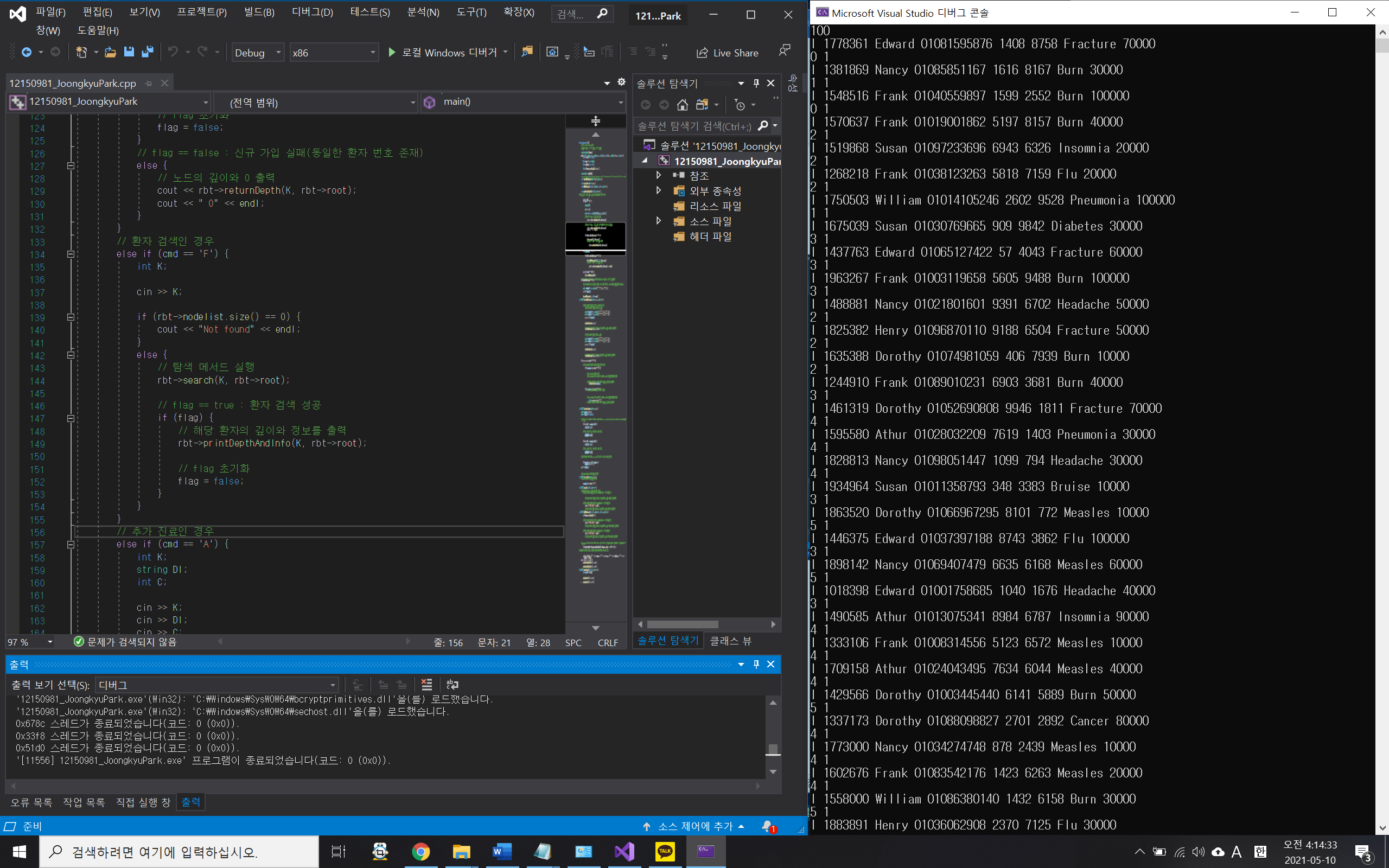
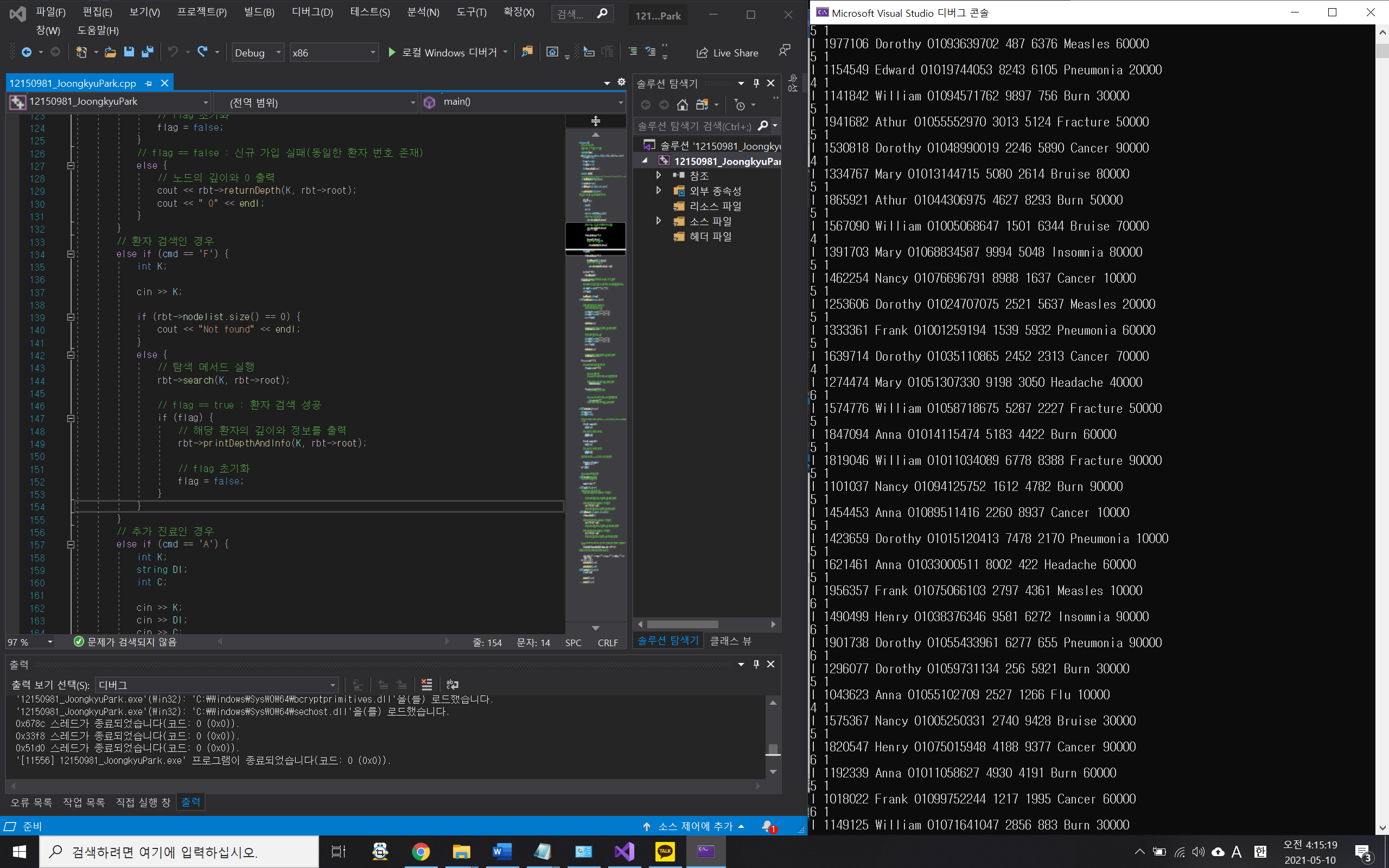
1. **신규 가입 관련 메서드**  
   - insert 메서드를 통하여 삽입과정 진입  
   - 트리가 비어있지 않다면, insertForSearch 메서드 진입  
   - 동일한 환자 번호가 존재하지 않는다면, 자신의 자리를 찾아 leaf노드 깊이까지 진입  
   **=> 트리의 높이만큼 비교연산 수행하므로, O(log n)**- 삽입 이후 checkDoublered 메서드 수행  
   - 더블레드가 아닐 경우 삽입과정 종료  
   - 더블레드인 경우 restructuring 또는 recoloring 메서드 수행  
   - restructuring 수행 시 상수 시간에 수행 후 삽입과정 종료  
   - recoloring 수행 시 상수 시간에 수행 후 할아버지 노드에 대해 checkDoublered 메서드 재귀수행  
   **=> 최악의 경우 root노드까지 가게되면 트리의 높이만큼 비교연산 수행하므로, O(log n)**- 삽입과정 종료 후, 삽입 성공/실패 두 경우 모두 깊이를 출력하기 위해 returnDepth 메서드 수행  
   - root 노드부터 시작하여 각 레벨별로 비교연산을 수행하며 해당 노드 깊이 탐색  
   **=> 최악의 경우 leaf노드까지 가게되면 트리의 높이만큼 비교연산 수행하므로, O(log n)  
     
   따라서 ‘신규 가입’에 대한 메서드의 시간 복잡도는 O(log n) time에 bound 된다.**
2. **환자 검색 관련 메서드**  
   - 트리가 비어있지 않다면 search 메서드를 통하여 탐색과정 진입  
   - 환자 검색에 성공한 경우 높이를 출력하고, 실패할 경우 ‘Not found’ 출력  
   **=> 최악의 경우 탐색 실패로 leaf노드까지 가거나, leaf노드의 부모가 탐색하려던 노드일 때 트리의 높이만큼 비교연산 수행하므로, O(log n)**- 탐색과정 종료 후, 탐색 성공 시 깊이를 출력하기 위해 printDepthAndInfo 메서드 수행  
   - root 노드부터 시작하여 각 레벨별로 비교연산을 수행하며 해당 노드 깊이 탐색  
   **=> 최악의 경우 leaf노드까지 가게되면 트리의 높이만큼 비교연산 수행하므로, O(log n)  
     
   따라서 ‘환자 검색’에 대한 메서드의 시간 복잡도는 O(log n) time에 bound 된다.**
3. **추가 진료 관련 메서드**  
   - 트리가 비어있지 않다면 addForSearch 메서드를 통하여 탐색과정 진입  
   - 환자 검색에 성공한 경우 질병 및 진료비 데이터 처리 이후 높이를 출력하고, 실패할 경우 ‘Not found’ 출력  
   **=> 최악의 경우 탐색 실패로 leaf노드까지 가거나, leaf노드의 부모가 탐색하려던 노드일 때 트리의 높이만큼 비교연산 수행하므로, O(log n)**- 탐색과정 종료 후, 탐색 성공 시 깊이를 출력하기 위해 returnDepth 메서드 수행  
   - root 노드부터 시작하여 각 레벨별로 비교연산을 수행하며 해당 노드 깊이 탐색  
   **=> 최악의 경우 leaf노드까지 가게되면 트리의 높이만큼 비교연산 수행하므로, O(log n)  
     
   따라서 ‘추가 진료’에 대한 메서드의 시간 복잡도는 O(log n) time에 bound 된다.**
4. **유행병 조사 관련 메서드**- RBT의 nodelist 벡터를 이용해 모든 노드를 조사하여 최근의 질병이 인자로 받은 유행병과 일치하는 사람의 수를 출력  
   **=> 모든 노드에 대해 최근 질병에 대해 비교연산을 수행해야하므로, O(n)  
     
   따라서 ‘유행병 조사’에 대한 메서드의 시간 복잡도는 O(n) time에 bound 된다.**

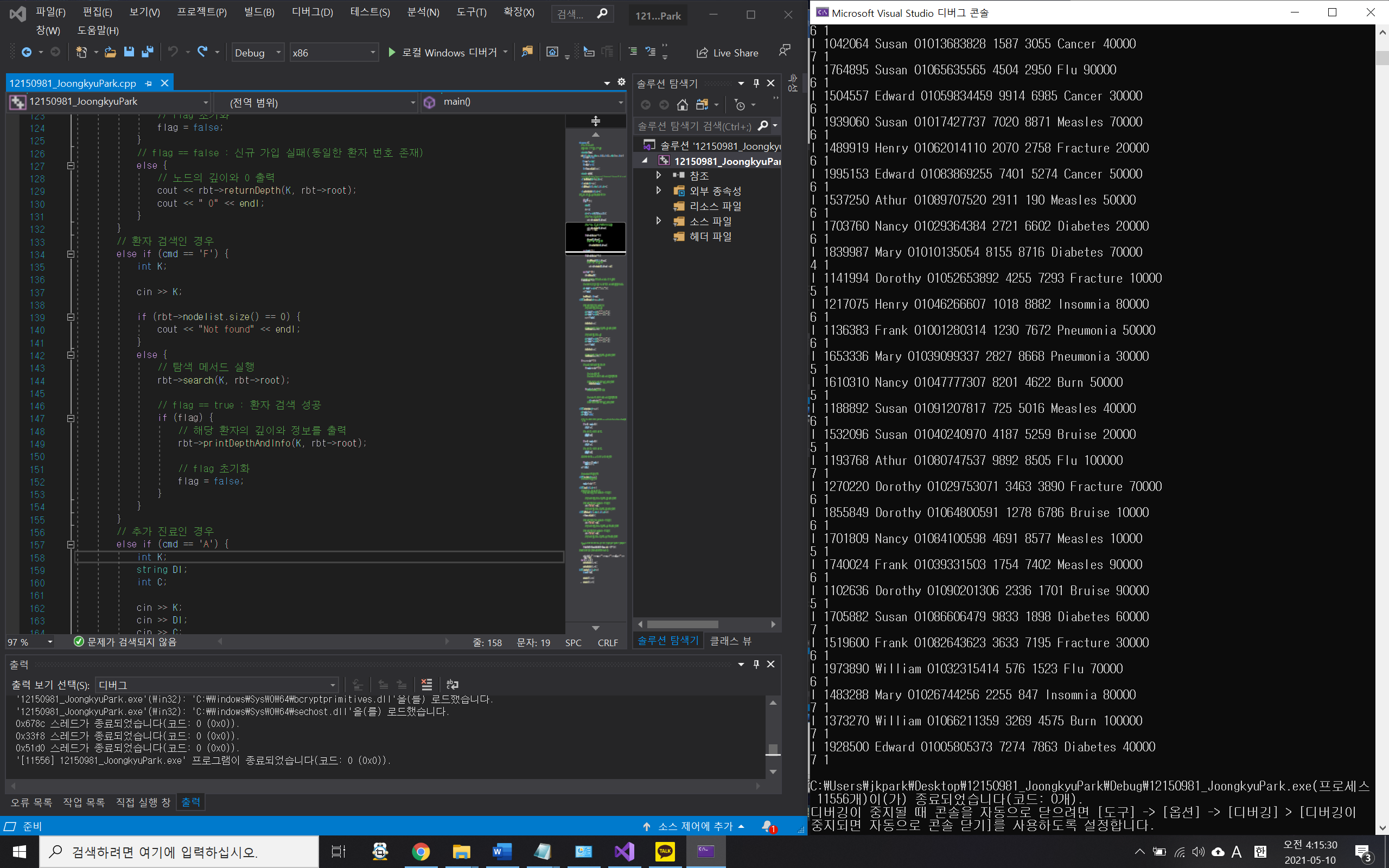
**4. 인터페이스 및 사용법**

**- 간단한 사용법 설명**

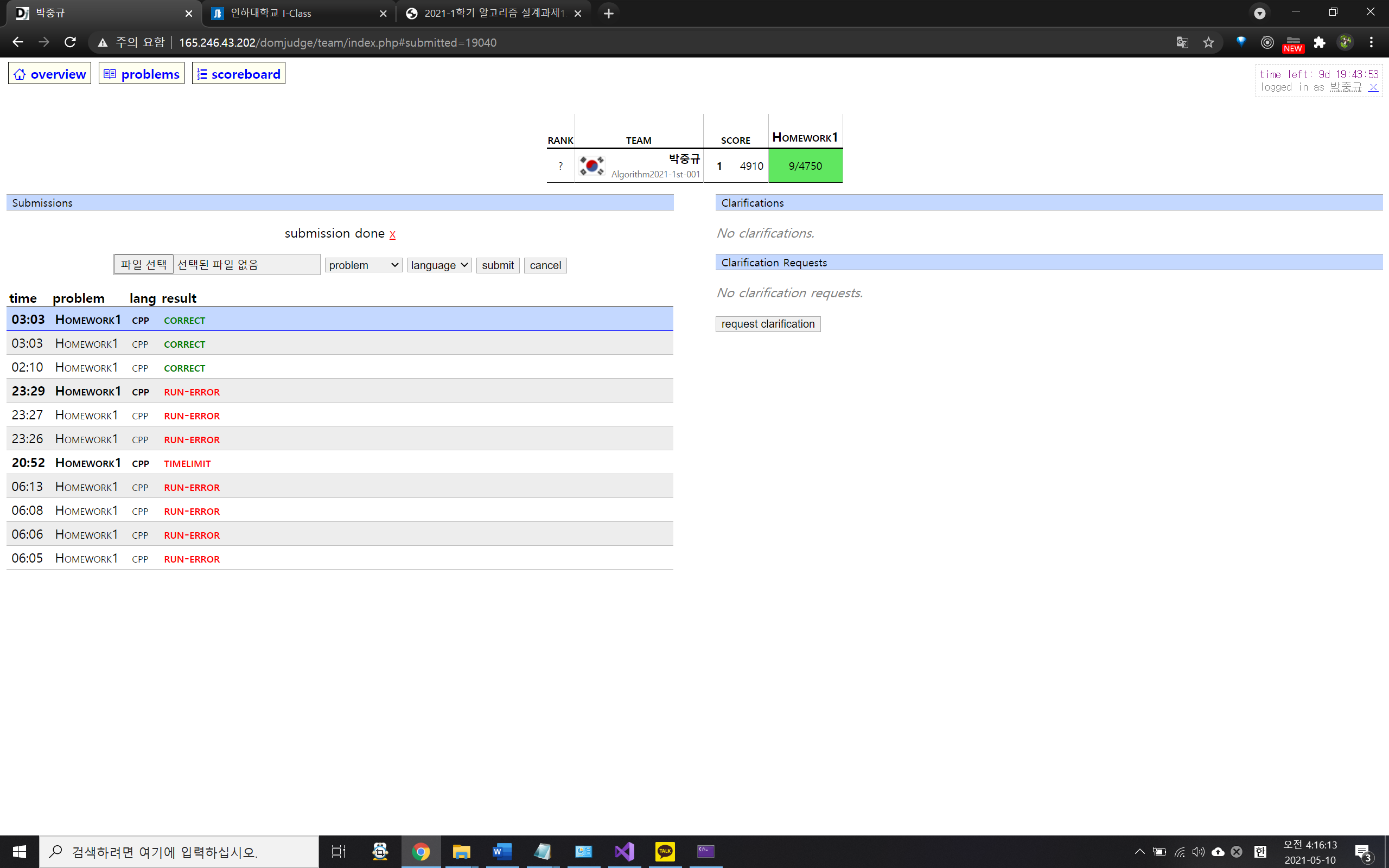
* **‘신규 가입’ 기능**  
  입력)  
  “I K N H Ax Ay DI C”의 질의형식으로 삽입하고자 하는 환자의 정보를 삽입  
  I: 환자를 신규 가입하는 질의를 나타내는 기호  
  K: 환자번호   
  N: 이름   
  H: 연락처   
  Ax: 주소의 x 좌표   
  Ay: 주소의 y 좌표   
  DI: 병명   
  C: 진료비  
  출력)  
  “D S”와 같은 출력형식으로 트리에 환자 정보가 저장된 노드의 깊이와(root의 깊이는 0) 신규가입 처리를 나타내는 정수(0: 거절, 1: 승인)을 출력  
  D: 트리에서 환자 정보가 저장된 노드의 깊이 (root의 깊이는 0)   
  S: 신규 가입 처리를 나타내는 정수(0 또는 1) (0: 거절, 1: 승인)  
  \* 만약 동일한 환자번호가 이미 존재한다면, 그 노드의 깊이를 출력하고 신규 가입은 거절된다.
* **‘환자 검색’ 기능**  
  입력)  
  “F K”의 질의형식으로 탐색하고자 하는 환자의 번호 삽입  
  F: 특정 환자를 검색하는 질의를 나타내는 기호   
  K: 환자번호  
  출력)  
  환자가 존재하면 “D N H Ax Ay”, 존재하지 않으면 “Not found” 출력  
  D: 트리에서 환자 정보가 저장된 노드의 깊이   
  N: 이름   
  H: 연락처   
  Ax: 주소의 x 좌표   
  Ay: 주소의 y 좌표
* **‘추가 진료’ 기능**  
  입력)  
  “A K DI C”의 질의형식으로 추가할 진료기룍과 탐색하고자 하는 환자의 번호 삽입  
  A: 특정 환자에 대해 새로운 진료 정보를 추가하는 질의를 나타내는 기호   
  K: 환자번호   
  DI: 병명   
  C: 진료비  
  출력)  
  환자가 존재하면 “D”, 존재하지 않으면 “Not found” 출력  
  D: 트리에서 환자 정보가 저장된 노드의 깊이
* **‘유행병 조사’ 기능**  
  입력)  
  “E DI”의 질의형식으로 조사하고자 병명 삽입.  
  트리에 저장된 모든 환자들에 대해, 마지막으로 진단받은 병명이 입력으로 주어진 병명과 동일하면 집계하고, 집계된 수를 출력하여 유행성의 정도를 파악. 편의상, 골절 같은 전염성이 없는 병명도 유행병으로 간주.  
  E: 마지막에 진료받은 병명을 기준으로 특정 병명을 수를 집계하는 질의를 나타내는 기호   
  DI: 병명  
  출력)  
  “T”와 같은 형식으로 출력  
  T: 모든 환자들의 진료기록에서 마지막으로 추가된 병명이 입력으로 주어진 병명과 같은 환자의 수(T ≥ 0)

**- 실행 화면 캡쳐 포함**1) pdf 예제  


2) 추가 test case 예제  




3) DOM 서버



**5. 평가 및 개선 방향**

**- 구현한 알고리즘의 장점**

* Red-Black tree 자료구조를 이용하여 트리를 balanced하게 만들어 탐색 수행시 최악수행시간을 O(log n)에 bound 할 수 있다.

**- 구현한 알고리즘의 단점**

* 구현한 기능 중 유행병 조사에 관한 메서드인 ‘epiForSearch’의 시간복잡도는 O(n) time이었다.
* 코드의 길이가 길어 가독성이 떨어진다.
* 변수 및 함수명이 부분적으로 직관적이지 않아 가독성이 떨어질 수 있다.

**- 향후 개선 방향**

* ‘epiForSearch’의 시간복잡도를 O(log n) time으로 bound시킬 수 있다면 더 좋은 알고리즘이 될 것이다.
* 코드의 길이를 줄여 가독성을 높이고 변수 및 함수명을 조금 더 직관적으로 이해하기 쉽게 만들면 좋은 코드가 될 것이다.
* 적절한 기준을 세우고 기준에 따라 코드를 작성하면 더 가독성이 좋고 안전한 코드가 될 것이다.