객체지향프로그래밍 실습

OOP 3-2 과제

학과 : 컴퓨터정보공학부

학번 : 2021202043

이름 : 이은서

날짜 : 05월 24일 (화)

문제 1.

1. 문제 설명

   이진탐색트리를 사용하는 문제이다. 삽입 명령을 받았을 경우, root와 삽입할 숫자를 비교하여, 삽입할 숫자가 root보다 작으면 왼쪽에, 크면 오른쪽에 숫자를 삽입한다. 삭제하는 경우에는 3가지 경우의 수를 따져서 확인해야한다. 만약, 삭제할 숫자의 자식노드가 존재하지 않는다면, 그냥 지워버리면 되고, 자식노드가 1개일 경우에는 삭제할 숫자의 부모노드와 자식노드를 연결시킨다음 지우면 된다. 자식노드가 2개일 경우에는 삭제할 숫자의 가장 오른쪽 자식노드 중에서 가장 큰 수를 복사한 다음 지우면 된다. Find의 경우에는 숫자를 찾는 과정을 출력하면 경로가 된다.

2. 결과 화면

```
"/Users/ieunseo/Downloads/practice C++/cmake-build-debug/3-3-1"
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 15
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 6
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 3
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 2
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 4
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 18
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 7
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 17
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 20
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 13
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 1 9
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 3 9
15->6->7->13->9
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 2 7
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 3 9
15->6->13->9
Enter Any Command (1: Insert, 2: Delete, 3: Find, 4: Exit): 4

종료 코드 0(으)로 완료된 프로세스
```

3. 고찰

Delete부분에서 3가지 상황을 모두 고려해서 코드를 짜야해서 틀린 부분이 많이 생겼지만, 자식노드가 모두 없을 경우와 자식노드가 하나일 때, 자식노드가 두개일 때를 구분해야했다. 그리고 Insert 함수가 잘 적용되었는지 확인하기 위해서 Print 함수를 따로 구현하여 중간중간 확인하였다. 그리고 재귀함수를 사용하여 코드를 간편히 하였다.

문제 2.

1. 문제 설명

텍스트 파일에 있는 역들을 이중연결리스트와 원형이중연결리스트를 이용하여 연결리스트에 값을 저장한다. 그리고 1호선은 이중 연결리스트로 양방향에서 움직일 수 있고, 2호선은 원형이중연결리스트로 양방향에서 움직일 수 있음과 동시에 끝에서 맨 앞을 연결하여 이동할 수 있게 하였다. 환승은 "시청"역에서만 가능하기 때문에 시청을 기준으로 나눈다. 가장 짧은 거리를 탐색해야 하는데, 1호선은 짧은 수단을 생각할 필요는 없고, 2호선 원형이중연결리스트에서 고려해야 한다. 이때, 정방향과 역방향에서 도착역까지 지나치는 역을 카운트하고, 각각을 비교한 다음 더 짧은 거리로 갈 수 있는 방향으로 역들을 출력한다.

2. 결과 화면

```
"/Users/ieunseo/Downloads/practice C++/cmake-build-debug/3-3-2"
출발역: 녹천
도착역: 이대
녹천->월계->광운대->석계->신이문->외대앞->회기->청량리->제기동->신설동->동묘앞->동대문->종로5가->종로3가->종각->시청->충청로->아현->이대
종료 코드 0(으)로 완료된 프로세스
```

3. 고찰

환승 부분에서 막혔었는데, 출발역과 도착역을 입력받고 이를 각각 1호선, 2호선으로 나눈 다음, 출발역과 도착역이 모두 1호선일때와 2호선일때, 그리고 1호선과 2호선, 혹은 2호선과 1호선일 때의 경우를 모두 나누어 설계하였다. 모두 1호선인 경우에는 정방향으로 갔을 때 도착역이 나오면 정방향으로 출력하고, 나오지 않으면 역방향으로 출력하게 하였는데, 출발역과 도착역을 정방향으로 두고 각각의 카운트를 세서 비교한 다음 방향을 정하도록 하였으면 더 효율적으로 코드를 짰을 것 같다. 모두 2호선일때는 어렵지 않았는데, 환승을 해야하는 경우에는 어려웠다. 시청에서 환승을 해야하는데, 그럴 경우 경로에 시청이 두번 출력이 되서 예외와 조건을 나눠서 한번만 출력되도록 했다. 또한, 2호선 파일의 맨 마지막 역인 문래에서 첫번째 역인 신도림으로 갈 때, 화살표가 두 번 나오는 문제가 발생했었다. 이는 원형이중연결리스트를 구현할 때, head와 tail(head->prev)가 비어져 있는 더미를 사용하여 만들었기 때문이었다. 그래서 한번 더 next 또는 prev로 이동한 다음 출력하도록 변경하였고, 문제를 해결하였다. 또 다른 문제는 출발역이나 도착역이 시청이고, 다른 하나는 2호선일때의 문제이다. 가장 먼저 1호선인지를 판별하다보

니 시청역은 1호선으로 판단이 되는데, 2호선에도 시청이 존재하여 출력할때 '시청시청' 이라는 값이 출력되거나 시청이 아예 출력되지 않았다. 이때는 출발역이나 도착역이 시청일 때 다른 하나가 2호선인 경우만을 따로 예외로 두어 문제를 해결하였다.

문제 3.

1. 문제 설명

   파일로부터 3명의 요리사가 할 수 있는 음식과 그 음식을 요리하는데 걸리는 시간을 각각 저장한 후, 주문 파일을 읽어와 그 주문을 완료할 요리사를 사용자로부터 입력받는다. 그리고 그 요리사가 할 수 있는 음식인지 비교한 후, 음식리스트에 있으면, 그 음식과 시간을 참조한다. 각 요리사별로 주문할 음식은 큐의 형식으로 저장되어 있고, 1초가 지날 때마다 3명의 요리사의 요리 진행 상황을 모두 보여주어야 한다. 그리고 남은 요리사가 마지막 요리를 다 끝낼때까지의 total time을 기록하여 보여준다. 큐는 링크드리스트를 이용하여 구현하고, 각 요리사가 할 수 있는 음식과, 요리 소요 시간을 저장한 배열을 사용할 것이다.

2. 결과 화면

```
"/Users/ieunseo/Downloads/practice C++/cmake-build-debug/3-3-3"
New order : pizza
Pick Cooker : Jack
New order : rawfish
Pick Cooker : John
New order : frenchfries
Pick Cooker : Bob
New order : pasta
Pick Cooker : Bob
New order : takoyaki
Pick Cooker : John
New order : coke
Pick Cooker : Jack
New order : coke
Pick Cooker : Bob
New order : takoyaki
Pick Cooker : Jack
New order : steak
Pick Cooker : John
New order : frenchfries
Pick Cooker : Bob
```

```
                Jack                    John                    Bob
        Cooking(pizza 7/8)      Cooking(rawfish 2/3)            Done!(frenchfries)
        Cooking(pizza 6/8)      Cooking(rawfish 1/3)        Cooking(pasta 3/4)
        Cooking(pizza 5/8)      Done!(rawfish)          Cooking(pasta 2/4)
        Cooking(pizza 4/8)      Cooking(takoyaki 4/5)       Cooking(pasta 1/4)
        Cooking(pizza 3/8)      Cooking(takoyaki 3/5)           Done!(pasta)
        Cooking(pizza 2/8)      Cooking(takoyaki 2/5)           Done!(coke)
        Cooking(pizza 1/8)      Cooking(takoyaki 1/5)           Done!(frenchfries)
            Done!(pizza)            Done!(takoyaki)
        Cooking(coke 2/3)       Cooking(steak 4/5)
        Cooking(coke 1/3)       Cooking(steak 3/5)
            Done!(coke)         Cooking(steak 2/5)
        Cooking(takoyaki 3/4)       Cooking(steak 1/5)
        Cooking(takoyaki 2/4)       Done!(steak)
        Cooking(takoyaki 1/4)
            Done!(takoyaki)

Total Time is : 15
종료 코드 0(으)로 완료된 프로세스
```

3. 고찰

파일을 읽을 때 항상 공백 또는 ₩n을 기준으로 문자열을 나눠서 읽어왔는데, 이번에는 각각의 정보를 각자 저장해야 해서 고민을 많이 했다. 결국 문자열 배열을 사용해서 한줄씩 읽어온 값을 통째로 넣은 다음, 두 번째 주문 파일을 읽어올 때 한줄씩 자른 값을 공백 기준으로 잘라서 비교하였다. 그리고, 일치하는 경우에는 요리사의 큐에 넣고, 그렇지 않은 경우에는 다시 입력받도록 문제를 해결하였다. 여기서 나는 한줄 씩 읽은 다음 공백 기준으로 나눠서 다시 저장했는데, 처음부터 파일로부터 각각의 정보에 입력받게 만들었으면 더 간단하게 할 수 있었을 것이다. 1초 지날때마다 걸리는 시간을 한 사람당 1초씩 3명의 요리사의 진행상황을 한번에 출력해야해서 시행착오를 겪었지만, 각 요리사별 time 변수를 새로 설정하고 pop될 때마다 다음 큐에 저장된 음식의 시간 정보를 불러와서 1초마다 진행하게 하였다. 또 다른 문제는 bob이 가장 먼저 요리를 끝냈는데, 다 끝난 이후에는 불러낼 게 없어서 그 상태로 프로그램이 종료되는 문제였다. If else문에 isEmpty 함수를 조건으로 하였는데, 조건이 명확하지 않아서 그런것임을 확인하고 고쳤다. 중복되는 코드가 많은데, 이를 다시 함수로 묶으면 조금 더 간결해질 것같다. 다만, 출력할 때 가운데 정렬을 하는 방법을 모르겠어서 눈대중을 '₩t'를 사용하여 정렬하였지만, 다음번에는 정렬을 하는 방법을 배울 수 있으면 좋겠다.

문제 4.

1. 문제 설명

학생정보를 관리하는 프로그램을 구현하시오. 프로그램은

Figure 1과 같이 전공, 학번으로 이루어진 Linked List들로 구현되고 학번으로 이루어진 Node마

다 이름을 저장하는 Binary Search Tree를 가진 구조로 학생정보를 저장한다. 모든 구조는 숫자

는 오름차순, 문자는 사전식 정렬을 따른다.

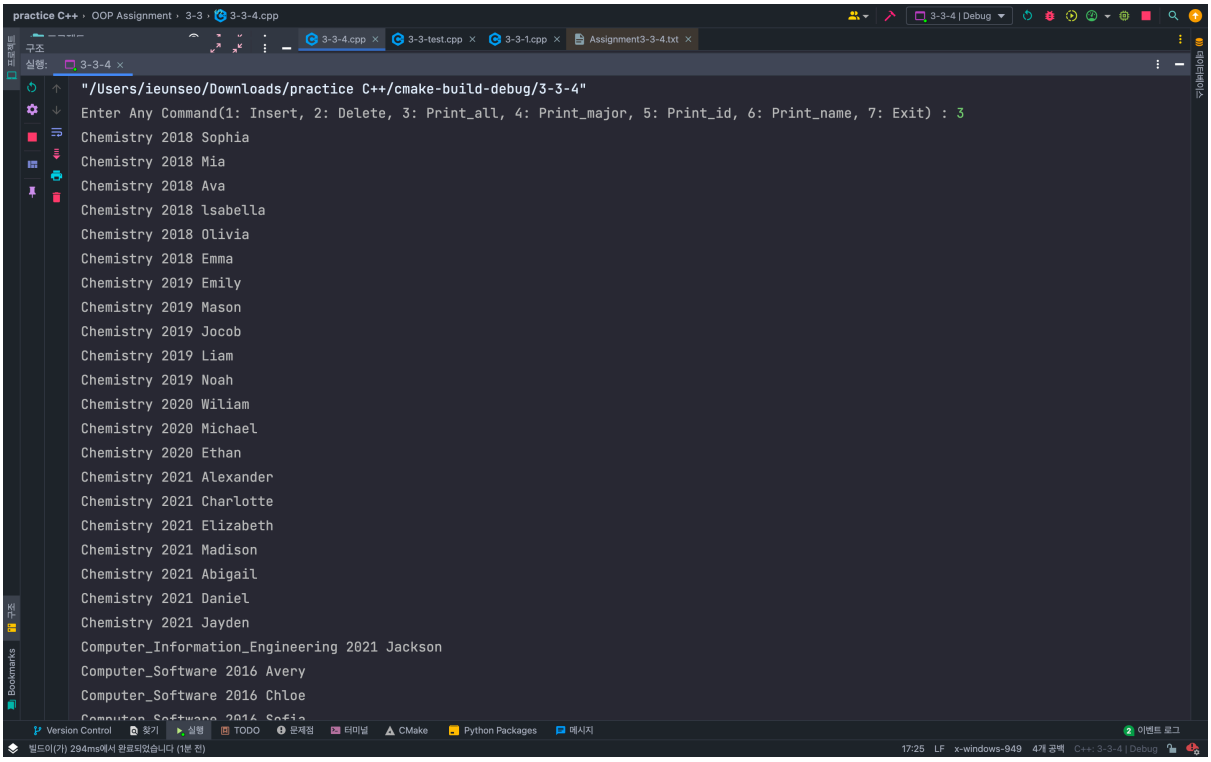
프로그램은 시작 시 Assignment3-3-4.txt로부터 학생정보를 읽어와 자료구조를 구축하고 Table

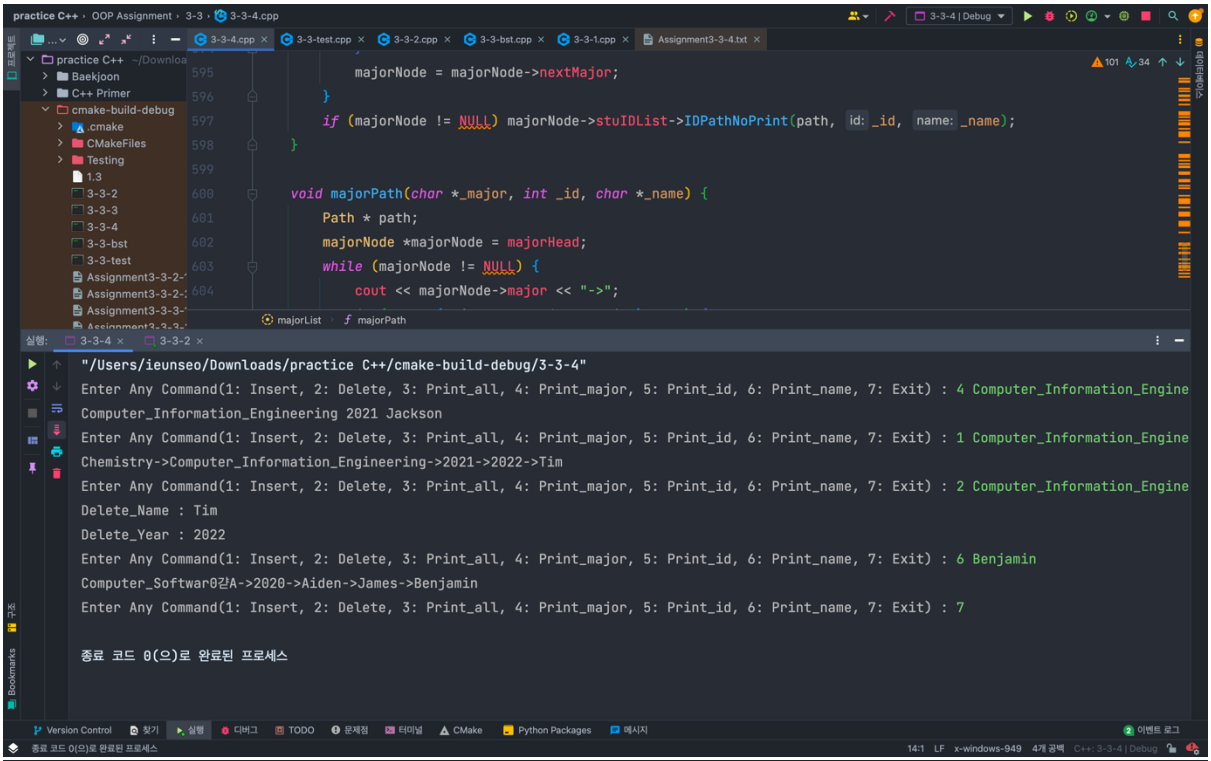
1에 적혀져 있는 Command에 따라서 동작한다. Assignment3-3-4.txt는 제공된 텍스트 파일로

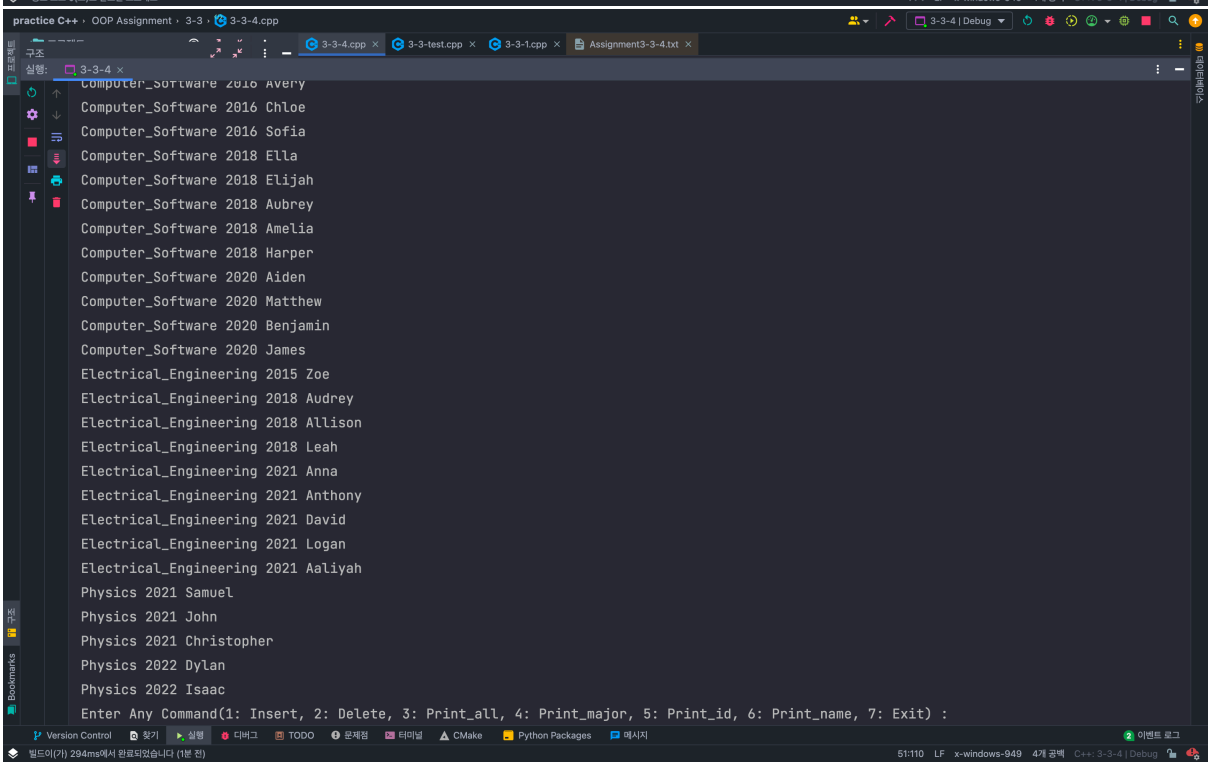
프로그램은 해당 파일을 읽어 대해서 동작해야 하며 학생 정보는 Figure 1외에 다른 구조로 저

장해서 사용할 수 없다.

2. 결과 화면







3. 고찰

Insert에서 전공, 학번이 처음일 때와 맨 마지막일 때, 그리고 없는 정보일 때 넣는 과정에서 시행착오를 겪었는데, 경우의 수를 세세하게 나눠서 따졌더니 해결할 수 있었다. 마찬가지로 delete, find, 모두 경우의 수를 제대로 생각하고 해결하는 것이 키포인트였다.