1. 문제 정의
A: Dept 클래스의 맴버들을 구현해 코드를 완성하고 call by value, call by reference의 차이를 이해하는 문제입니다.

2. 문제 해결 방법
A: 먼저 클래스의 각 맴버들이 각각 무슨역활을 하는지 정리했습니다. 정의해야되는 맴버는 총 4개로 복사생성자, 소멸자, void read(), bool isover60(int index)가 있는데 이들은 각각 생성자로 생성한 객체를 복사하는 역할, 소멸할 때 동적 메모리를 지워주는 소멸자, 입력한 정수를 size만큼 scores[]배열에 담아야하는 read, index라는 학생의 성적을 60을 기준으로 비교하여 true,false를 리턴하는 역할입니다. 먼저 복사 생성자 Dept(const Dept &dept)는 문제에서 주어진 함수 countPass에 값을 전달할 때 call by value가 일어날때는 복사가 일어나는데 이때 얕은 복사가 이루어지면 소멸자가 작동할 때 2번 지워져서 오류가 생기므로 깊은 복사가 일어나도록 구성했습니다. 소멸자는 생성자로 객체 생성이 일어날 때나 복사생성자가 생성될 때 new연산자를 통해 hip메모리를 할당 받기 때문에 이것을 지워주기 위해서 delete를 사용해서 메모리 반환을 해줬습니다. void read()는 유저가 입력한 size만큼의 숫자를 scores[]배열에 넣어줘야 함으로 for 반복문과 cin을 사용해서 scores[]배열에 들어가도록 구성했습니다. bool isover60(int index)가 index라는 학생이 60보다 클 때 true를, 그게 아니면 false를 리턴하면 되므로 if-else문을 사용해서 구성했습니다. (3)번 문제같은 경우엔 여기서 복사생성자를 지우고 실행하면 지금같은경우에는 countPass에 값으로 전달되고 있기 때문에 복사가 이루어지는 상황인데 깊은복사가 이루어지도록 복사생성자를 정의하면 상관없지만, 얕은 복사가 일어나면 나중에 메모리 반환이 이루어질 때 복사된걸 먼저 지우게되면 얕은복사는 같은 메모리 공간을 공유하기 때문에 원본객체를 반환할 때 메모리가 없는 문제가 생기게 됩니다. 따라서 복사가 일어나지 않도록 countPass에 call by reference가 일어나도록 해주면 복사본이 만들어지지 않기 때문에 메모리 반환문제가 일어나지 않게 됩니다.

3. 아이디어 평가
A: call by value의 개념과 call by reference의 차이에 대해서 정확히 알아서 각각 어떤 작용이 일어나는지 파악한게 중요한것 같습니다. 얕은복사와 깊은복사의 차이를 정확히 알아서 복사생성자를 생성할 때 무엇을 사용해야 하는지 구분한게 중요한 것 같습니다.