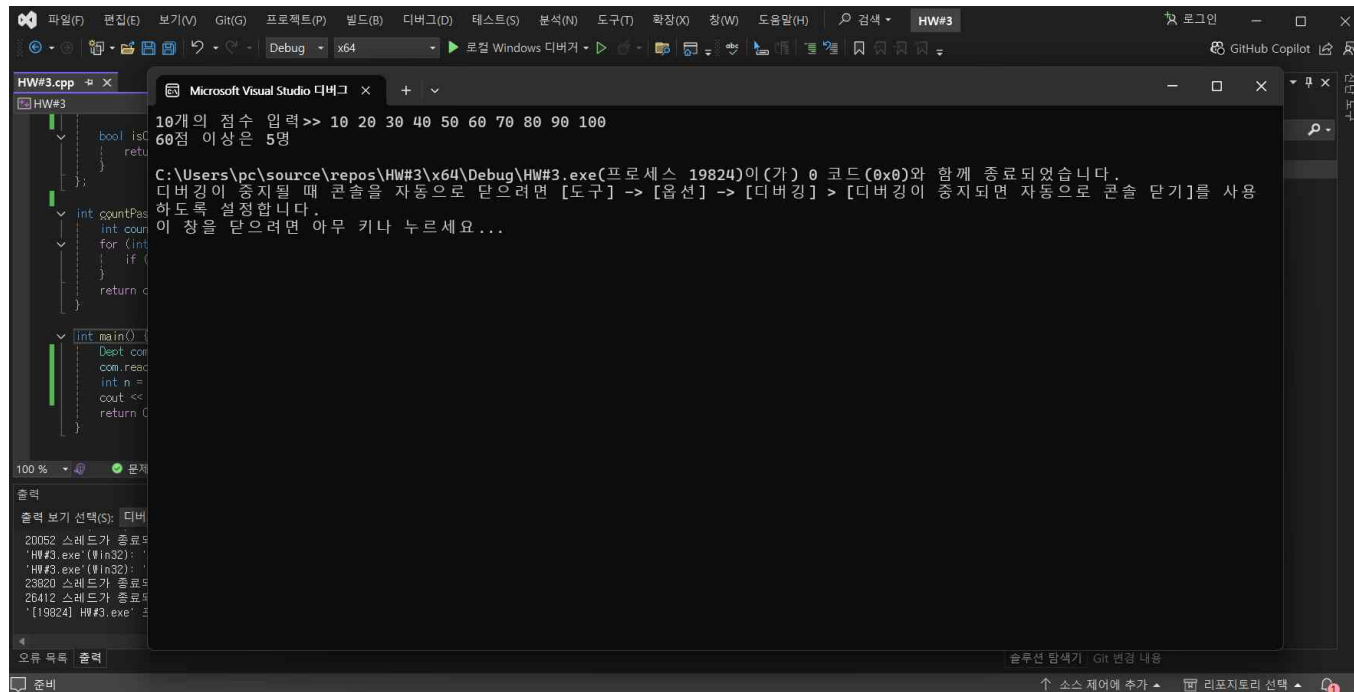


1) 소스코드 수행결과 화면 캡처



```
HW#3.cpp
10개의 점수 입력>> 10 20 30 40 50 60 70 80 90 100
60점 이상은 5명

C:\Users\pc\source\repos\HW#3\x64\Debug\HW#3.exe(프로세스 19824)이(가) 0 코드(0x0)와 함께 종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

int main()
{
    Dept com;
    com.read();
    int n = com.getN();
    cout << n << endl;
    return 0;
}
```

2) 소스 구현 설명

2-1) 문제 정의

주어진 프로그램은 Dept 클래스를 사용하여 학생들의 점수를 입력받고, 그 중에서 60점 이상인 학생의 수를 계산하는 기능을 요구한다.

이 과정에서

동적 메모리 관리: 배열을 통한 점수 저장

객체 복사 문제: 기본 복사 생성자를 사용하면 동적 메모리 할당된 배열의 얇은 복사가 이루어져, 두 객체가 동일한 메모리를 가리키게 되는 문제가 발생할 수 있다.

참조를 통한 전달: 객체 복사를 방지하고 성능 향상을 위해 함수 호출 시 객체를 복사하지 않고 참조로 전달해야 한다.

또한, 참조로 전달된 객체를 수정하지 않도록, const 한정자를 통해 상수 객체로 다뤄야 하는 문제도 발생했다.

2-2) 문제 해결 방법

1. 동적 메모리 관리:

Dept 클래스에서 동적 메모리로 scores 배열을 할당하고, 소멸자에서 이 메모리를 해제하여 메모리 누수를 방지한다.

2. 복사 생성자 제거:

복사 생성자를 구현하지 않고, 객체를 함수에 전달할 때 참조를 사용하여 불필요한 복사가 발생하지 않도록 한다.

3. const 사용:

객체의 데이터를 읽기만 하고 수정하지 않는 함수(getSize()와 isOver60())에 const 한정자를 추가하여 const 참조로 호출할 수 있도록 한다.

2-3) 아이디어 평가

1. 동적 메모리 관리:

동적 배열을 사용하는 Dept 클래스는 new 연산자를 사용해 메모리를 할당하고, 소멸자에서 delete[]를 통해 할당된 메모리를 해제한다. 이 방식은 메모리 누수를 방지하며, 객체가 더 이상 사용되지 않을 때 적절하게 메모리를 해제한다.

2. 복사 생성자 제거 및 참조 전달:

복사 생성자를 제거하고, Dept 객체를 const 참조로 함수에 전달함으로써 불필요한 객체 복사를 방지했다. 이를 통해 성능이 향상되었고, 동적 메모리를 사용하는 객체에서 발생할 수 있는 얇은 복사로 인한 문제를 회피했다.

3. const 한정자 추가:

getSize()와 isOver60() 함수에 const 한정자를 추가함으로써, 상수 객체에 대한 접근을 허용했고, 함수가 객체를 수정하지 않는다는 점을 명확히 했다. 이로 인해, 참조를 통한 함수 호출에서도 컴파일 오류를 방지했다.

2-4) 문제를 해결한 키 아이디어 또는 알고리즘 설명

키 아이디어: 참조를 통한 객체 전달과 const 한정자의 활용

이 문제에서 가장 중요한 해결 방법은 객체를 참조로 전달함으로써 불필요한 복사를 방지하고, 동적 메모리 할당과 해제 문제를 피한 것이다. 또한, const 한정자를 함수에 추가하여 상수 객체를 안전하게 처리할 수 있도록 했다.

참조 전달: 객체를 복사하지 않고 참조로 전달하는 방식은 성능을 최적화하는 데 중요한 역할을 했다. 특히, 동적 메모리를 사용하는 객체에서는 복사 생성자를 통한 얇은 복사가 위험할 수 있으므로, 참조를 사용해 이러한 문제를 원천적으로 방지했다.

const 한정자: const 한정자는 함수가 객체의 상태를 변경하지 않음을 명시적으로 알리는 역할을 하며, 상수 객체 또는 상수 참조로 호출되는 경우에도 오류 없이 실행될 수 있게 해준다.