□ 1-1 문자열 뒤집기 (C)

✓ 정답 여부
- 문자열을 뒤집는 기본 로직은 잘 구성하셨습니다.

▲ 개선할 점
-반복문의 조건에서 `i <= len`은 범위를 한 칸 초과합니다. 이로 인해 `answer[len]`에 접근하는 시점에서 이미 `my_string[-1]`에 해당하는 접근이 발생해 ₩*₩*미정의 동작(오류 가능성)₩*₩*이 생깁니다.
-올바른 조건은 `i < len`입니다.

□ 보완 제안

``c for (int i = 0; i < len; i++) { answer[i] = my_string[len - 1 - i]; } answer[len] = '₩0'; ```	c for (int i = 0; i < len; i++) { answer[i] = my_string[len - 1 - i]; } answer[len] = '\0';

□ 2-1 p와 y 개수 비교 (C++)

✓ 정답 여부
* 문제는 정확하게 해결하셨습니다.

▲ 개선할 점
-`answer = true/false`로 조건을 나누고 있지만, 불필요하게 코드를 길게 쓰고 계십니다.

□ 보완 제안
-불필요한 조건문을 제거하고 아래처럼 간결하게 작성 가능합니다.

``cpp return count == count2; ```	cpp return count == count2;

□ 2-2 올바른 괄호 (C)

✓ 정답 여부
-전체적인 구조는 잘 잡으셨고, 괄호 수가 맞는지 판단하는 논리는 들어가 있습니다.

▲ 개선할 점
- `count < 0`을 검사하는 조건은 루프의 맨 앞이 아닌, `count--` 이후로 이동하는 것이 더 안정적입니다.
- 조건이 너무 중복되고 길어져 있어 가독성이 떨어집니다.
- `if (count > 0 || count < 0)` → 그냥 `count != 0`으로 간단히 처리할 수 있습니다.

□ 보완 제안

``c for (int i = 0; i < len; i++) { if (s[i] == '(') count++; else if (s[i] == ')') count--; if (count < 0) return false; } return count == 0; ```	c for (int i = 0; i < len; i++) { if (s[i] == '(') count++; else if (s[i] == ')') count--; if (count < 0) return false; } return count == 0;

□ 3-1 문자열 숫자 여부 확인 (C)

✓ 정답 여부
* 숫자 체크를 정확히 구현하셨습니다.

▲ 개선할 점
- `return answer = false;`와 같은 표현은 혼동을 줄 수 있습니다. 단순히 `return false;`가 더 명확하고 안전합니다.
- 마지막에 `return answer;`이 아닌 `true`를 바로 반환해도 됩니다.

□ 보완 제안

``c if (len != 4 && len != 6) return false; for (int i = 0; i < len; i++) { if (!isdigit(s[i])) return false; } return true; ```	c if (len != 4 && len != 6) return false; for (int i = 0; i < len; i++) { if (!isdigit(s[i])) return false; } return true;

□ 3-2 JadenCase 문자열 만들기 (C++)

✅ 정답 여부
- 문제를 정확하게 풀었습니다. 단어의 첫 글자를 대문자로, 나머지는 소문자로 바꾸는 방식이 잘 구현되어 있습니다.

▲ 개선할 점
- 로직 자체는 좋지만, 변수명 `nextword`는 `isNewWord`나 `newWord`와 같이 의미가 더 명확한 이름으로 바꾸는 것이 좋습니다.
- `string answer = s;`에서 불필요하게 복사하지 않고 바로 수정할 수도 있습니다.

□ 보완 제안
- 성능은 괜찮지만, C++에서는 `ostringstream`을 이용해 문자열을 조립하는 방식도 고려해볼 수 있습니다.

📌 종합 피드백 요약
1-1 반복문 범위 초과 수정 필요 (`i <= len` → `i < len`)
2-1 불필요한 조건 제거 가능, `return count == count2;`
2-2 중복 조건 정리 및 `count != 0` 활용
3-1 불필요한 변수 제거, 직접 `true/false` 반환 추천
3-2 변수명 개선, 문자열 생성 방식 다양화 가능