```cpp
#include <iostream>
#include <vector>
using namespace std;

bool fail_lecture (const vector<int>& attendance_records){
    int absent_count=0;
    for (int i=1; i<attendance_records.size(); ++i){
        absent_count += attendance_records[i]==0;
    }
    cout << "actual absent_count: ";
    cout <<  absent_count << endl;
    return absent_count >=3;
}
bool fail_lecture_fixed(const vector<int>& attendance_records){
    int absent_count=0;
    for (int i=0; i<attendance_records.size(); ++i){
        absent_count += attendance_records[i]==0;
    }
    cout << "expected absent_count: ";
    cout <<  absent_count << endl;
    return absent_count >=3;
}
int main() {
    vector<int> attendance_records = {0, 1, 1, 1, 0, 0, 1, 0, 1, 1};

    cout << "Attendance records: ";
    for (int v : attendance_records) cout << v << " ";
    cout << endl;

    bool actual_output = fail_lecture(attendance_records);
    bool expected_output = fail_lecture_fixed(attendance_records);

    cout << "Expected output: " << (expected_output ? "true" : "false") << endl;
    cout << "Actual output:   " << (actual_output ? "true" : "false") << endl;

    return 0;
}

```

1. There is a fault in this code. In line 3 of the failt_lecture function (actual line 7), the loop starts at 1, meaning that the first instance in attendance_records is skipped, incorrectly representing the absent_count.
2. If the student was not absent on the first day (attendance_records[i]!=0), then the fault will not execute. An example test case for this is {1, 0, 1, 0, 1, 0, 1, 0, 1, 0}. Since the first value is 1 (not absent) the fault will not execute as it would not add an absence to absent_count either way. In this case, the expected and actual output will match and the internal absent_count will also match in both the original and fixed functions.
3. The fault will execute but not result in an error state if the first value of attendance_records represents something other than attendance. This is because the only way for the fault to execute in the program is for an error state to also be present, as the internal count (absent_count) will not be the same as the expected one. This means that if we consider the first value in the test case to represent something different, then a test case such as {0,1,1,1,1,0,0,1,0,1} will have the same expected and actual value, as the first value is regarded differently. Without this change to how the first value is interpreted, there is not test case that executes the fault without also resulting in an error state.
4. The program will result in an error state but not a failure if the student was absent 4 or more times. This is because the first day will not matter, as there will be 3 other times (at least) that the student was absent, resulting in the same result either way. A test case where there is an error state but no failure would be {0,0,0,0,1,1,1,1,1,1} because there are 4 total absences, and even though the first day is absent and wouldn't be counted, there will still be >=3 absences, resulting in the Actual and Expected output both returning true. In this case, the internal absent_count for the original function will be 3 which, while still appearing to work, does not have the same internal value as the fixed function without the fault.
5. A test case that results in a failure would be anytime the student is absent 3 times and one of those times is the first day. This would look something like {0,0,0,1,1,1,1,1,1,1}. In this test case, the original function will skip the first day, returning **false** as there are only 2 instances of absences. The expected output, however, would be **true** since there are 3 instances of absences, since the first day must be included. This causes a mismatch between the actual and expected outputs, which is a failure in the program.