1. There is a fault in the program, which is that the iterator variable “i” in the “for” loop is initialized to 1, and then used to access vector values at the index of “i”. This is a fault because, in C++, vector indices start from 0, so this function that is meant to add 1 or 0 to a total depending on every value in the vector actually misses the first value in the vector. In other words, the function’s expected behavior is that it adds to the total using all numbers in the vector, while its actual behavior does not use all numbers in the vector because it misses the first one, which is faulty behavior.

2. There is no possible test case that can avoid executing the fault. This is because the “for” loop that contains the faulty initialization of “i” is not contained within a conditional control statement, but is instead in the main body of the function which always executes. Because the “for” loop always executes, and the “for” loop contains the fault, the fault always executes.

3. A test case that executes the fault but does not result in an error state is:
   {1, 1, 1, 0, 1, 0, 1, 1, 1, 1}
   An error state is when anything internal to the software is not what it is meant to be. For this function, the internal variable “absent_count” should always be the sum of the total number of 0s up to and including the current iteration index “i”. In this test case, the function never adds a 1 or 0 for the number 1 at index 0, but, because the number is a 1, a correct version of the function would add a 0, which is more correct in terms of lacking faults, but does not affect the internal state. Therefore, the internal state of this incorrect function remains correct throughout the running of the function for this test case.

4. A test case that results in an error state but not a failure is:
   {0, 1, 0, 0, 1, 1, 1, 0, 1, 1}
   A failure is when the software returns an incorrect value to the output or to the rest of the software. A correct version of this function would count all 4 of the 0s in the test case and output 1 because 4 is greater than or equal to 3, and there would be no failure. The incorrect version of this function counts 3 of the 0s in the test case and outputs 1 because 3 is greater than or equal to 3. The internal counter of 3 is wrong at the end of the program, as it should be 4. However, the output is still 1 like it would be without the fault, meaning that there was no failure.

5. A test case that results in failure is:
   {0, 1, 1, 1, 0, 1, 1, 1, 1, 0}

A correct version of the function would count all 3 of the 0s and output 1 because 3 is greater than or equal to 3. However, the actual version of the function never counts anything at index 0, so it would count 2 of the 0s and output 0 because 2 is not greater than or equal to 3. This output is wrong, so the result counts as a failure.