1.

a)

Bugs:

Line 6: Way too much memory is allocated.

Line 8: No newline at the end of printed string.

Line 9: Program returns without freeing allocated memory.

b)

Issues with this code:

Half of the allocated memory is never used. The second for loop uses array accesses to read the values set in the first for loop instead of using the iterator variable, which has the same value as the value at that index, which means the second access to all of the addresses used in the first loop can be omitted. Also, the content of the second loop can be executed in the first loop instead, saving the overhead of doing two loops. The memory isn't freed up either.

c)

Things to note about the reverseParams source code

Line 11: unsigned char is used as the data type for the allocated memory. This might lead to problems when using strict rules about type conversion.

Line 14: The length of the sting being copied can affect the results of this line. Copying a string that is too long can result in writing to memory that has not been allocated.

Line 16: Iteration starts at index 10, which is outside of the allocated 10 bytes. This means that the implementation relies on there being a null there, which is not the case for input strings of length greater than 10 thanks to the blind copy on line 14.

Line 20: The string terminating null value is searched for from the back of the string. Apparently when allocating memory after another chuck has been freed, the same memory will be used. However when writing, only 8 bytes get written at a time. This means that data written to indices 8 or higher will remain there unless the next write also writes to this chuck of 8 bytes. The string terminating null is enough to overwrite a chunk.

Line 23: Not breaking might solve some issues since a proper null value should be present as long as the string we intend to reverse was not too long.

Line 28: lastChar might not be initialized. This only happens when the scanned string had only null values. However nothing actually happens in this case.

Line 31: Case handling assumes that any Ascii values higher than or equal to that of 'a' (97) represent lower case letters. While there are not a lot of other characters with asci values higher than 97, it should be checked.

Line 35: Similarly for upper case. This affects a couple more characters, since there are a some with values between the value of 'a' and 'Z'

Line 42: Technically the same bug as on line 28. The result would be that only null values get printed, which leads to the output being empty for that string.

Line 59: Not entirely sure if this is the case here, but a comment at the end of the file with no newline at the end might lead to problems.

I tested some inputs both on Linux and on Windows. Apparently Windows does not null memory that is allocated for you, so there might be random junk in there. This junk is visible in the tests. On Linux on the other hand, the allocated memory only seem to have null values. The effect of this is that all of the junk visible on windows is not visible on Linux, the program is implemented with the assumption that memory that it does not touch only has null values. This observation is useful for debugging cross platform programs, since making assumptions about how the system works is bad.

1.

b)

I ran my program on its015-13. The time command reported that the time spent was 9.144s in real, 7.323 in user and 0.044s in sys mode.

c)

When I ran on CMB, I got the results 42.10s, 156.50J and 6588.54Js.

d)

Running on Vilje, I got the results of 14.335s in real, 13.981s in user and 0.312s in sys mode when using the time command.

e)

All pixels correct, no corners cut.

f)

For each color, for each iteration of the function call, the sum += … operation happens 57506436 times when size is 2, and 662042304 times when size is 8. The function also carries out five iterations rather than having the function called five times with the same size, so in reality the operation happens five times more.

g)

Optimizations being used:

- Variable declarations in the performIdealIteration function moved to the top.
- Summing all colors in one pass to divide number of passes by 3.
- Swapped row-column order of the for loops, so the loops over x are on the inside of the loops over y.
- Moved the bounds check on y out of the x loop.
- Strength reduction on offset variable for the out-image, no need to calculate from scratch every time when it only increments by 1.
- Moved the five passes to inside the function to avoid the overhead of calling the function five times.
- Finalizing images earlier, to allow freeing earlier.

- Freeing allocated memory after last usage.