Patrick Gallagher

September 21, 2025

Project 1 Report

LLM Usage

For Project 1, I used AI for debug assistance. The chat completions setting was disabled in Visual Studio Code Copilot, and I did not generate any section of my code. I did not capture any screenshots during this project as I missed that portion of the project description. The first two prompts are from real development, while the third prompt is an example.

Prompts

1. "I am attempting to read a text file using redirected input. My compiler is Visual Studio. How should I achieve this?"
    a. Rationale: During Lab 1 and 2, I was unable to figure out how to test input in my programs, which required me to use the auto grader to test my program. Once I completed the content of my first method, initialize, I wanted to test its functionality.
    b. Incremental Development: VSC Copilot was unable to help me resolve my issue. It told me to call the program from the command line using the redirected input format, "program.exe < input.txt." At this point in development, I was using the debug button in VSC to compile and then run the program. During lab 2, I quickly completed the lab and spent the rest of my time attempting to find a solution. Eventually, I switched to operating the compiler from the command line, which allowed me to use the redirected input format properly.
2. "I am reading a text file as input. I want to read in every character individually, including whitespace characters, but my current code does not read in carriage returns. Why is this happening?"
    a. Rationale: During testing, I continued to have issues with reading in characters properly. I was struggling to determine if the problem was with my compiler, my coding environment, or my code. I hoped that Copilot might be able to provide a solution to a problem that I was not aware of.
    b. Incremental Development: The AI response did not help me to resolve the issue. According to the AI, the issues that were occurring should not have been occurring given the state of my code. To verify, I submitted my code to the auto grader, where the code passed all tests, indicating that issue was not with my program, but with another element of my coding setup.
3. "This method decompresses a sequence of characters based on a provided rule. The method is functional in its current state, but I would like to make it more performant. What are some steps that I could take to accomplish this?
    a. Rationale: If I was to continue developing this program, I would want to optimize space and time complexities. To learn more about possible solutions, it would be helpful to provide an AI model with my method code and the above prompt. I

could investigate each of the options presented to me in further detail to determine how best to better my program.

Debug/Testing

Debugging

Throughout the course of development for Project 1, I encountered many issues. I tested each method individually after completing the code for that section using the provided input and output tests. Once each section was functioning as intended, I moved onto the next method. However, the first time that I attempted to run my modified boilerplate, I encountered a stack overflow issue due to a massive array in the stack. To resolve this, I changed my Text2Compress object to be dynamically allocated, which resolved any stack overflow issues.

The first method that I developed was the Initialize method. I struggled to get started because I was having difficulty with redirected input, which impeded my ability to understand the cin operator. After a little online research, and taking inspiration from the boilerplate, I decided to use cin.get() to read in each character individually. However, regardless of what I tried, I could not get carriage returns to read in properly. I verified that this issue only occurred in my testing environment, as the code passed the auto grader without issue.

The next two methods I developed were the trivial display methods. Each of these methods functioned as intended with no issues. The next method, Train, did have several trouble spots. In my first few tests, the rules learned did not match the expected rules learned, so I investigated the frequency matrix portion of my method. I discovered that I had forgotten to reassign the max value variable each time the if statement that checked if the current value is greater than the max value, which resulted in the first pair of values being identified as having the greatest frequency. Once I inserted the proper assignment statement, the code worked as expected. I also found an issue with the encode behavior at the end of the encode strand. Because my code overwrites the _seq array rather than creating a new copy array, if I do not overwrite each entry properly, then the sequence will not return as expected. My encode solution checks the previous entry and the current entry and then assigns the previous entry to a compressed value or the original value depending on the rule. A displacement variable is incremented each time a pair is compressed to ensure that the insertions maintain sequential order. However, when the code reached the final pair, it only reassigned the second to last entry, and because the size of the array was likely to shrink due to compression, the last entry would be incorrect. Because only the last entry was incorrect in testing, I quickly identified that there was an issue with that case and added a special last case if statement for a compression pair and a regular pair that assigned the last entry properly.

The encode method was essentially completed inside of the train method, so my encode development was straightforward once my train method was validated. The only changes that I needed to make was to change the rule identification to read from the rule array rather than implementing each rule as they were discovered. The final method to develop then was the

decode method, which is the method that gave me the greatest difficulty, outside of the carriage return issue. To implement the decode function, I wanted to overwrite the current array rather than making a copy. To do this, I created a small backload array that holds values as the sequence is extended. Once a compressed value is expanded, the expanded pair will overwrite data in the sequential entry. In order to preserve data, I place any values that may be overwritten in the backload array before evaluating if it is a compressed value, which ensures that I do not experience data loss. On first completion, I did not realize that the decode was reading new material from the input file, rather than decoding the trained sequence. As a result, I needed to rework my code to read in all of the rules and the compressed sequence before decoding. The final issue that I encountered was again with an end case. Because the sequence expands and the for loop length is dictated by the length of the sequence, a number of values equal to the number of decompressions would be excluded from the final sequence. To address this, I added a small loop inside of the for loop that would execute on the final iteration of the parent loop that would unpack the remaining backload values. With this final addition, my code passed all of the input and output examples and passed the auto grader.

Example Input/Output

My first input tested what would happen if the iterations exceeded the length of the sequence and what would happen if the compressed sequence was filled with compressed values.

100 1

Grabbit crossing ahead!

1

109 56 128

45 83 99 124 128 128 128 128 56 128 128

This test revealed two issues with my current program. First, the program produced rules even when there was only value in the sequence. I corrected this by adding a break statement once the length equaled 1. Second, the compressed sequence did not print properly, due to an accidental case in which a value is dropped during the last cycle of the for loop. After completing these edits, the code produces this ideal output:

Rules learned from Compression:

32 97 128

32 99 129

33 10 130

71 114 131

97 98 132

97 100 133

98 105 134

101 133 135

103 128 136

104 135 137

105 110 138

111 115 139

114 139 140

115 138 141

116 129 142

131 132 143

134 142 144

136 137 145

140 141 146

143 144 147

145 137 148

146 148 149

147 149 150

Compressed sequence:

150

Decompressed Text:

-Sc|m8m8m8m88m8m8