**Documentation: Huffman tree implementation**

Mohammed Towseef

Advanced Algorithms

Concordia

Farah Kamw

Aug 04, 2024

**Problem Analysis**

The task was to write a Huffman tree in C++ and develop a Huffman code for various characters with their frequencies. Huffman Coding is a type of data compression technique that belongs to the lossless category, and in this technique, characters of the input are assigned codes of string length depending on their frequency. It is assumed that the most frequent characters should be assigned the shortest codes, which will decrease the size of the encoded information. The issue is to build the Huffman tree, assign the codes so that no code is a prefix of another one (prefix-free property), and then output these codes in a particular order.

**Approach**

The solution implements a greedy approach with the help of a priority queue (min-heap) to construct the Huffman tree. It begins by creating a leaf node for each character, and these nodes are then inserted into a min-heap. Two nodes with a minor frequency are successively removed and combined into a new internal node, which is then inserted back into the heap until only one node is left, which forms the root of the Huffman tree. The last tree encodes every character by providing binary codes while maintaining the prefix-free property.

**Steps**

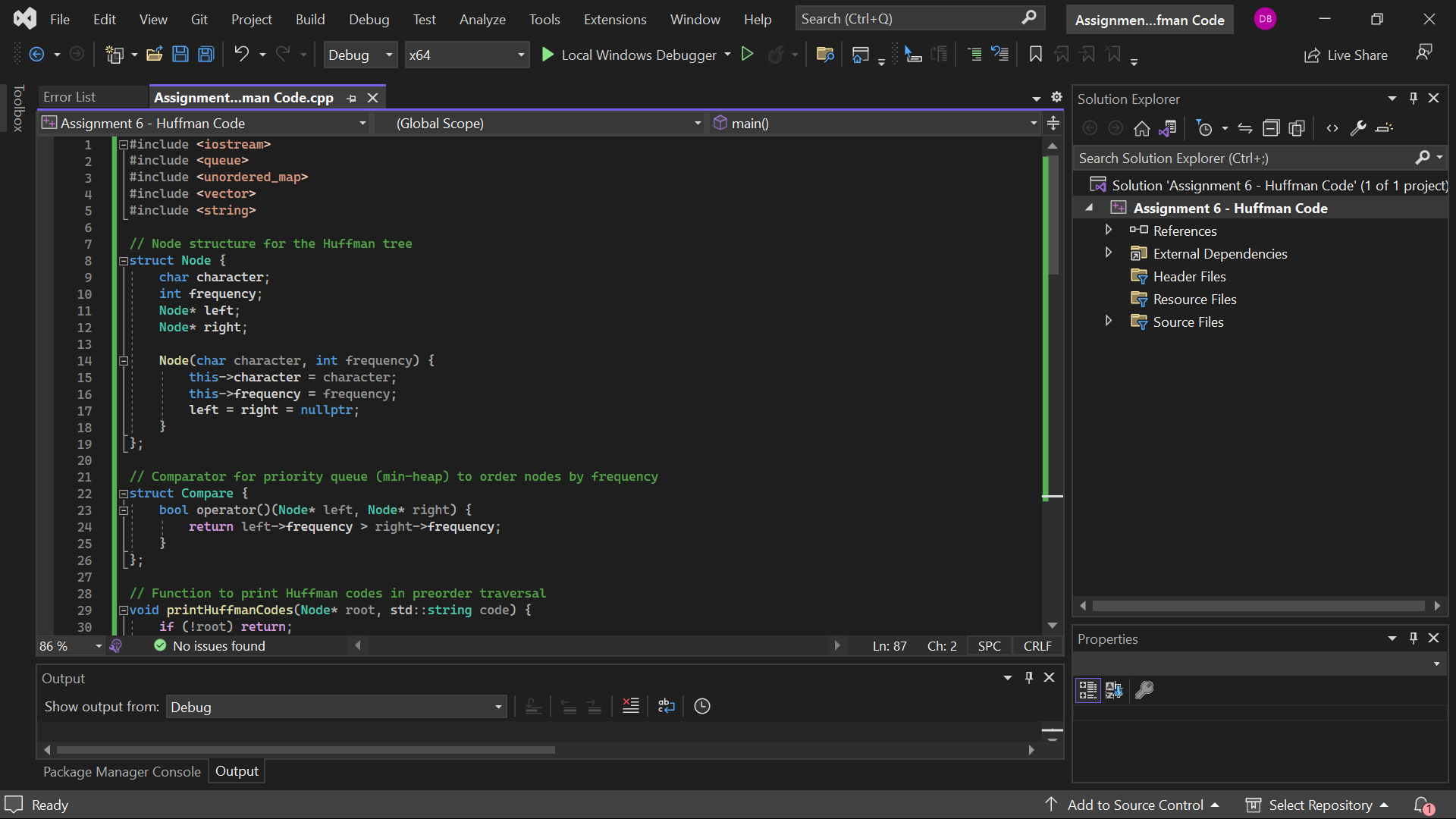
**1. Define the Huffman Tree Building Function: Define the Huffman Tree Building Function:**

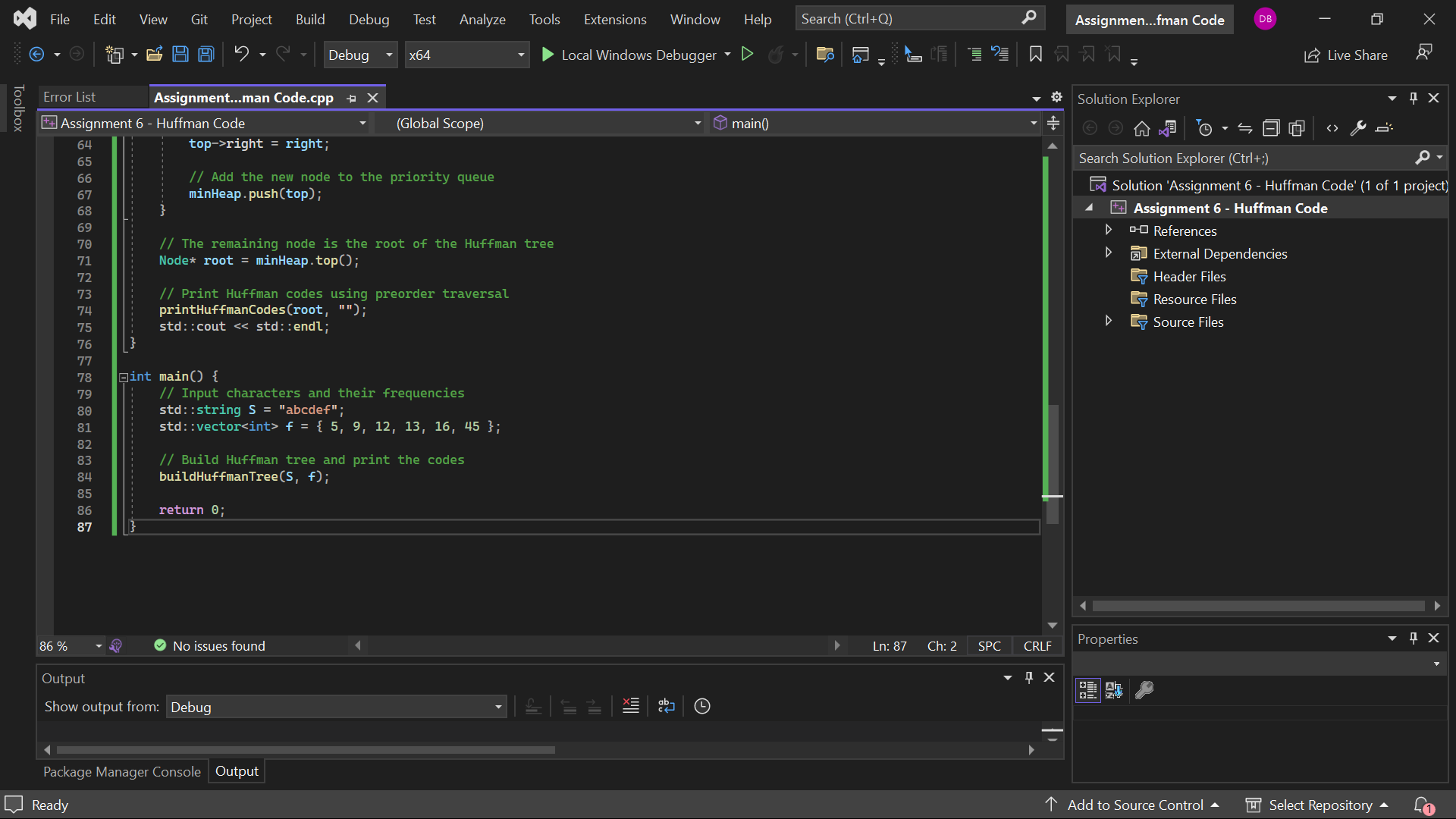
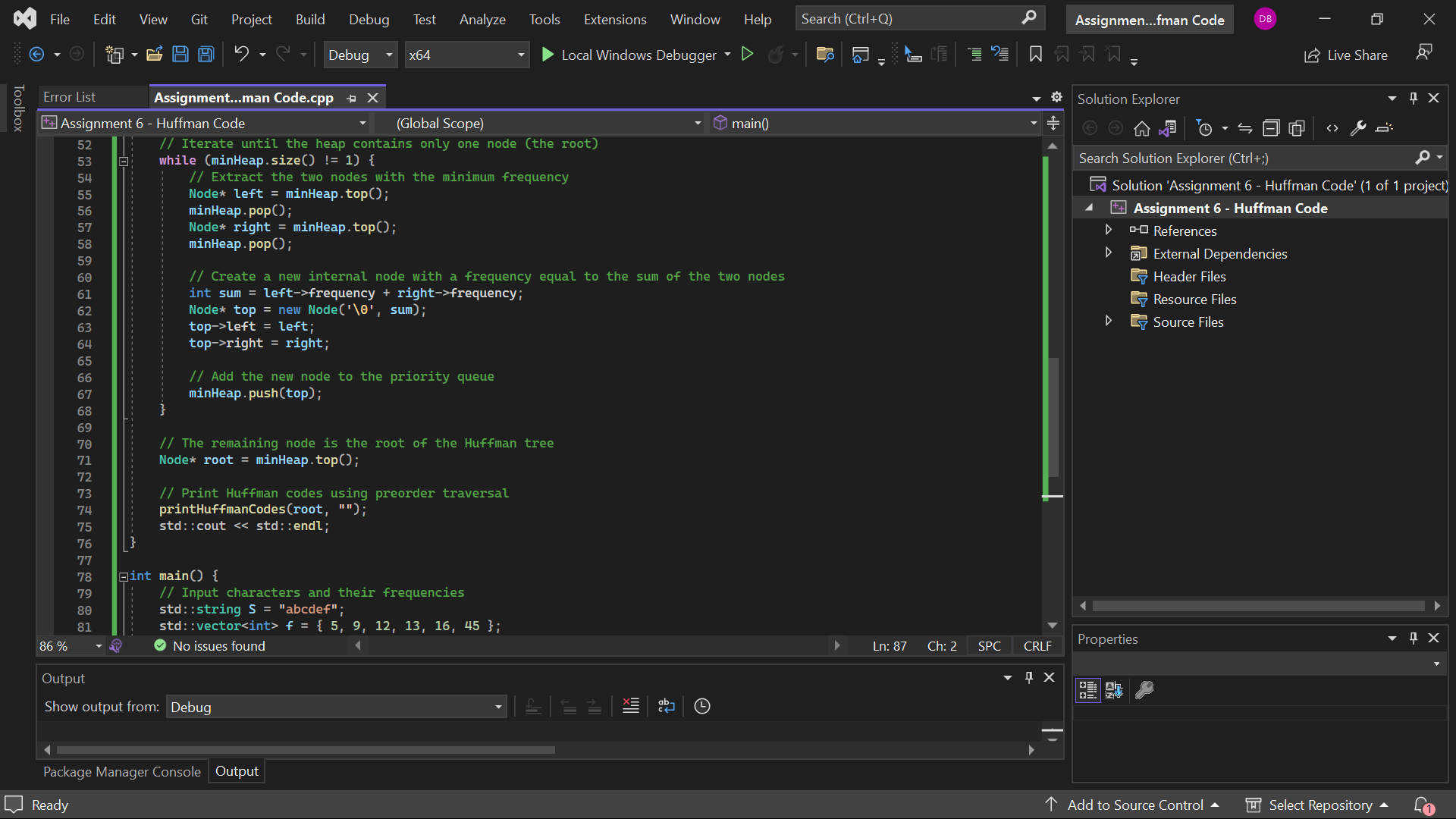
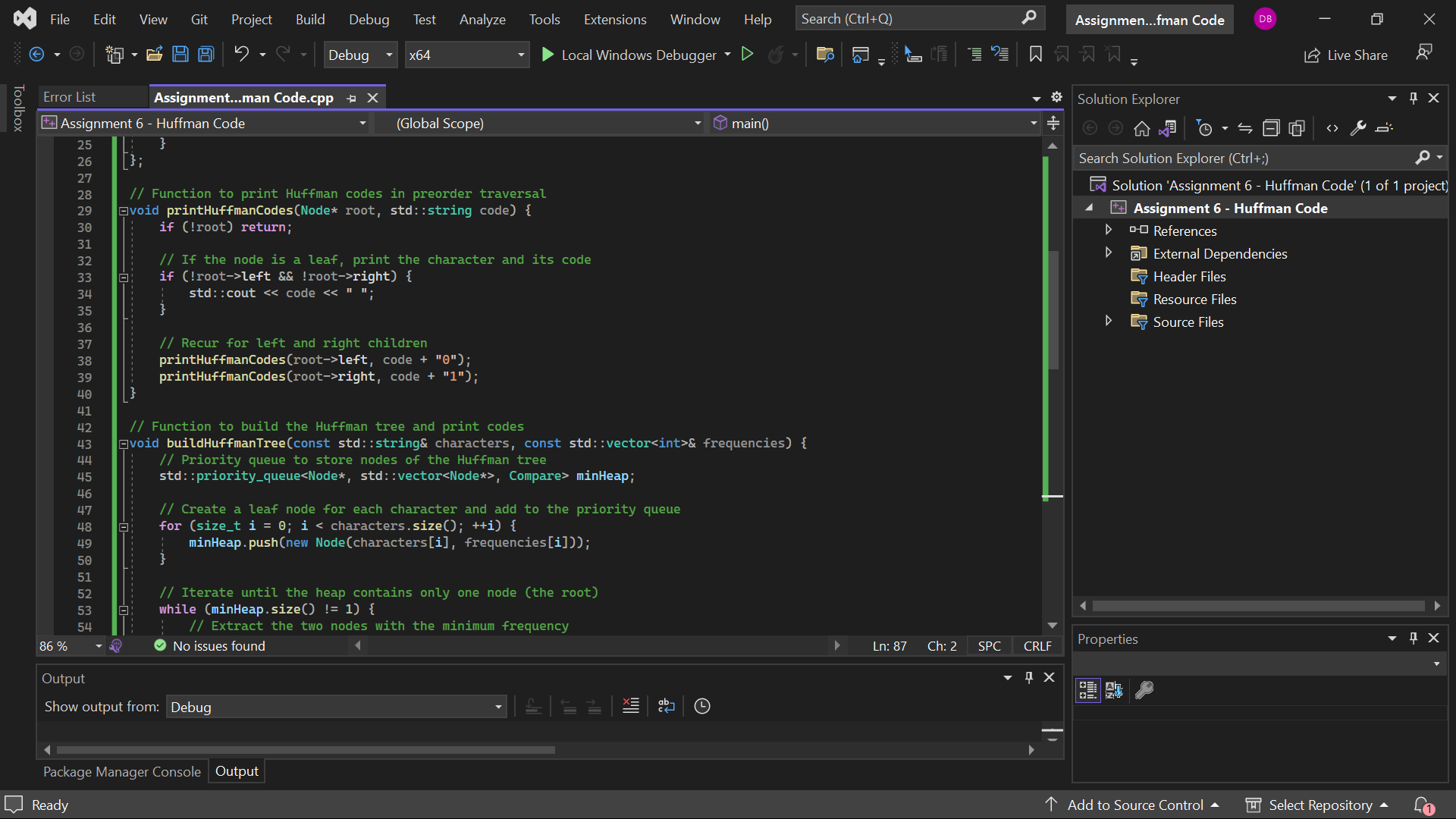
The following function sets up the priority queue, where each node in the queue is a character along with its frequency. It handles the joining process of nodes with the least frequencies to build the Huffman tree.

**2. Implement Methods:**

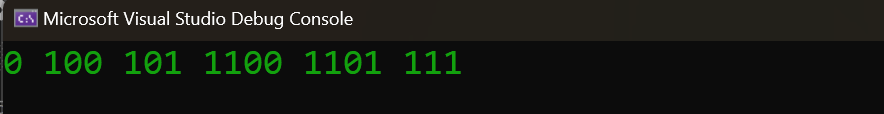
* **add Element (int row, int col, int value):** While not directly translated, a similar function is used to add nodes to the priority queue by their frequencies.
* **print Sparse Matrix ():** Similarly, the print Huffman Codes () function prints the Huffman codes in a preordered manner, showing the resulting prefix codes for the characters.
* **print Matrix ():** This is equivalent to printing the Huffman codes for all the symbols in the set. The Huffman tree’s traversal enables the reconstruction and output of the codes assigned to each character, demonstrating the efficiency and accuracy of the coding system.

**Code Explanation:**





**OUTPUT :**



**Time Spent on Assignment**

This took me roughly 4 hours to complete the assignment. It took this time to familiarize myself with the Huffman coding algorithm, code the solution in C++, and check the coded solution's correctness. The first steps included, therefore, a revision of the theoretical elements related to Huffman coding, such as the building of the Huffman tree and the derivation of prefix codes. I then set aside time to write the C++ code, especially for developing data structures and the priority queue. Last, I used my time to remove the bugs and check the results against what I expected.

**Self-Evaluation of Effort**

Concerning the effort I devoted to comprehending the problem, coding it, and checking whether the solution complies with the specifications stated, I rate myself as 'A.' I ensured I revisited the necessary concepts and worked as hard as possible to ensure I had the working solution. I also wanted a clear picture of each part of the problem to ensure I comprehensively attacked the assignment. The time management and problem-solving strategies proved efficient as I could accomplish the task before the due date.

**Self-Evaluation of Solution**

Looking at the solution's correctness and the given task guidelines, I think I got an 'A' for the outcome. The program works as Huffman coding is correctly implemented, the output is as expected from the algorithm, and the input data is processed appropriately. The solution encompasses all the components of the problem, which are the construction of a Huffman tree and Huffman code generation. The output is as required, and the code is well formatted and documented.

**Summary of Issues and Challenges**

I can generate the Huffman codes for the specified characters and frequencies in my solution. However, I faced some issues in the priority queue imp the lamentation, e, especially in arranging the nodes in order of frequency. I faced challenges when working with the tree nodes, such as freeing the memory, which is a rather important issue in C++. Debugging the tree construction logic was also challenging, especially ensuring all the nodes were connected correctly and traversed. In order to solve these problems, I had to read the C++ documentation and find examples of priority queues and node manipulation. However, these are the challenges it faces, and the final solution provides the expected output for the given input.

**Conclusion**

The scheme I proposed for the Huffman Coding implementation is rather resilient and effective for the given problem. It proves to be a good approach to building the Huffman tree and producing the best prefix-free codes that assign the shortest binary to the most frequently occurring characters. This implementation uses the greedy algorithm and guarantees that the compressed data will not lose any data. Nevertheless, the solution is good for the given set of characters and their frequencies, but there is always room for improvement, especially when two characters have the same frequency or when the input data set is rather large. Moreover, expanding the scope of the user input validation and the error-handling mechanisms would also improve the program’s reliability.

This assignment well illustrated the practicality of priority queues and tree structures in C++. It also extended the instruction about memory management and how data should be handled in C++. This exercise highlighted the importance of appreciating algorithm development, especially approaches that reduce vagueness when deciphering the solution. Overall, this project helped me gain more experience as a programmer and enhance my knowledge of data compression and algorithm optimization.