LLM and GitHub Copilot Usage Documentation

Prompts and Suggestions

1. **Prompt**:
   "How to implement a compute method for a Chip class that performs operations like addition, subtraction, multiplication, and division in C++?"

Suggestion:
The LLM suggested creating a method that computes the result based on the chip type (A for addition, S for subtraction, M for multiplication, and D for division). It recommended handling edge cases like division by zero by including a conditional check and printing an error message when such cases occur.

2. **Prompt**:
   "How do I handle dynamic memory cleanup for an array of Chip objects in C++?"

Suggestion:
The LLM recommended creating a loop that iterates through the array of Chip pointers and uses delete to free each dynamically allocated object. After the objects are deleted, the array itself should be freed by deleting the pointer to the array.

3. **Prompt**:
   “How do I handle input redirection in Visual Studio Code?”

Suggestion:

The suggestion was a bit similar to the one in in the Project instructions, however I came across an issue with running the programming by redirecting the input normally because I used a PowerShell terminal instead of other terminals like cmd or bash. To fix that I started using “Get-content input1.txt | ./project2.exe” to take advantage of input redirection on a PowerShell terminal.

4. **Prompt**:
   "How can I display connections between chips in a readable format?"

Suggestion:
The LLM helped implementing a display() method for the Chip class that shows the chip's ID and the connections with other chips, indicating which chip provides input and which receives output. I had trouble with the formatting but later on I figured out that I was just adding extra spaces, thus I was able to fix that.

Rationale

- The prompt regarding the compute method helped streamline the logic for performing chip operations, especially by emphasizing the importance of handling special cases like division by zero. The suggestion to check for input1 and input2 before performing operations ensured robustness.

- The dynamic memory management suggestion was crucial for ensuring that no memory leaks occur. This prompt guided the development of a proper memory cleanup strategy by using delete[] for array pointers and preventing memory issues in larger circuits.
- Input redirection suggestion for a PowerShell terminal helped me run the code easily since the instructions given in the project description weren't applicable for my terminal
- The display method suggestions helped enhance the clarity and readability of chip connection outputs, which was important for debugging and visual verification of the established circuit.

Incremental Development

- **Stage 1**: After receiving the LLM suggestion for the compute method, I implemented basic operations like addition, subtraction, multiplication, division and negation. Each chip's result is calculated based on its inputs, with additional safety checks for division by zero.
- **Stage 2**: I refined the method further to allow the connection of chips by using pointers (input1, input2, and output). Suggestions helped to ensure that the input and output relationships between chips were correct.
- **Stage 3**: The display() method was implemented incrementally based on suggestions, ensuring that each chip correctly showed its inputs and outputs. The method was refined to handle different chip types, like input chips (I), output chips (O), and negation chips (N).
- **Stage 4**: The LLM also helped with handling dynamic memory and ensuring the proper deletion of allocated chip objects at the end of the program, preventing memory leaks.

Debugging and Testing Plan

Specific Tests

1. **Basic Operation Tests**:
    - **Addition Chip**: Tested a chip with two inputs connected to addition chips to verify the correct summation of input values.
    - **Subtraction Chip**: Created a subtraction chip and connected two input chips with different values to verify that subtraction was performed correctly.
    - **Multiplication Chip**: Tested a multiplication chip with two input chips connected to ensure multiplication worked as expected.

- **Division Chip**: Tested the division operation with valid inputs and handled division by zero cases.

2. **Edge Cases**:
    - **Division by Zero**: Tested a division chip where the second input chip has a value of zero. The program correctly printed an error message and did not crash.
    - **Negation Chip**: Tested a negation chip to ensure that it correctly negated the value of the input chip (positive and negative values were tested).
3. **Input Chip Tests**:
    - Verified that input chips stored the values correctly when set via the input commands and passed those values to other chips in the circuit for further operations.
4. **Connection Tests**:
    - Tested connections between chips, such as connecting input chips to various operation chips (addition, subtraction, multiplication) and verifying that the outputs were correctly computed based on the inputs.

Issues and Resolutions

1. **Division by Zero**:
    - **Issue**: During testing, when dividing by zero, the program attempted to divide without checking, causing it to fail.
    - **Resolution**: I added a check in the compute() method to handle division by zero. If the second input chip had a value of zero, the program printed an error message and set the result to zero to prevent the crash:
2. **Memory Management**:
    - **Issue**: The program created dynamic Chip objects but did not initially delete them, leading to memory leaks.
    - **Resolution**: I added a cleanup section to the program to ensure that all dynamically allocated Chip objects were deleted at the end of the program, **using delete for individual chips and delete[] for the array of chips.**
3. **Output value from the circuit**
    - **Issue**: My code was producing the correct output for the rest of the code apart from the value from the circuit which was always 0 for all the 3 tested input files.
    - **Resolution**: Previously I had made 3 methods that were responsible for assisting the computation and readiness of chips for computation, but they

were actually causing some form of redundancy in the code, and the output value wasn't being assigned even though the debugging process showed that the inputs were assigned properly, therefore I ended up removing those methods and sticking to a simpler implementation of the code.

Verification

1. **Running Provided Input.txt files** :
    - I ran the provided input text files to check if the code produced the required output
2. **Gradescope Test Cases**:
    - As I uploaded the code file to Gradescope I checked if the code was passing all the test cases. The one I struggled a lot with was the test case that checked spaces and ", " but at the end I was able to fix it.