USE AND SIGNIFICANCE OF MODULES IN THIS TASK

In the context of this task, modules play a crucial role in organizing and separating code into reusable and maintainable units. Here's how modules are used and their significance in this task:

1. **Separation of Concerns**: Modules help in separating different concerns of the program. In this task, we have separate modules for reading a file, modifying its content, and writing the updated content to a new file. Each module focuses on a specific task, making the code easier to understand and maintain.
2. **Code Reusability**: Modules allow us to reuse code across different parts of the program. In this task, the **readFile.js**, **modifyFile.js**, and **writeFile.js** modules can be used in other parts of the codebase or in future projects without duplicating the code. This promotes code efficiency and reduces redundancy.
3. **Encapsulation**: Modules encapsulate related functionality, providing a clear boundary and preventing conflicts between different parts of the code. In this task, each module has its own scope, variables, and functions, ensuring that they don't interfere with each other.
4. **Abstraction**: Modules provide an abstraction layer, hiding the internal implementation details from other parts of the code. This allows other modules or files to use the exported functionality without worrying about how it is implemented. In this task, the main program file (**app.js**) doesn't need to know the internal workings of the **readFile.js**, **modifyFile.js**, and **writeFile.js** modules. It only needs to use their exported functions.
5. **Modularity**: Modules promote modularity, making it easier to test, debug, and update specific parts of the code. In this task, if there is a bug in the file reading logic, we can focus on the **readFile.js** module without affecting other parts of the codebase.

By utilizing modules, we can achieve better code organization, reusability, maintainability, and scalability in this task, as well as in larger codebases.