Mini-project: Configuration File Reader

1. Project Goal:

In this mini-project, you will build a configuration file reader that can parse and store various settings such as key-value pairs. The goal is to create a class that reads a configuration file and stores its values in a structured format.

2. Information Needed and Classes:

Configuration files typically store settings in key-value pairs and sections, which are grouped logically. In a webserver configuration, for example, settings like port, server name, root directory, and error pages are common.

Here is an example configuration file format for a webserver:

```
# Webserver configuration example

server {
    listen 8080;               # Server listens on port 8080
    server_name localhost;       # Server name

    root /var/www/html;          # Root directory for files
    index index.html;            # Default file if no file is specified

    error_page 404 /errors/404.html; # 404 error page
    error_page 500 /errors/500.html; # 500 error page
```

```
    client_max_body_size 10M;     # Maximum client body size

    location /uploads {
        allowed_methods POST;     # Only POST method allowed for uploads
        upload_directory /var/www/uploads; # Directory to save uploads
    }

    location / {
        allowed_methods GET DELETE; # Allow GET and DELETE methods
    }

    location /cgi-bin/ {
        cgi_path /usr/bin/php-cgi;  # Path to CGI executable (e.g., PHP)
        allowed_methods GET POST;   # Allow GET and POST for CGI
    }
}

server {
    listen 9090;                # Another server listening on port 9090
    server_name myserver.com;     # Server name

    root /var/www/myserver;       # Root directory for this server
    index main.html;              # Default file for this server
}
```

This structure allows you to configure different aspects of a web server, including ports, server

names, file paths, error handling, and more.

Class 1: ConfigurationReader

- This class is responsible for reading the configuration file and storing the settings.
- It should contain methods for reading the file, parsing the contents, and storing key-value pairs in an appropriate data structure.

Attributes:

- file_path: The path to the configuration file.
- settings: A map or dictionary that stores key-value pairs (e.g., std::map<std::string, std::string> in C++).

Methods:

- readFile(): Reads the file from the disk and processes its contents.
- parseLine(): Parses individual lines to extract the key and value.
- getSetting(key): Returns the value associated with a given key.

Class 2: Setting

- This class represents an individual setting in the configuration file, consisting of a key and a value.

Attributes:

- key: The name of the setting (e.g., "port", "server_name").
- value: The value associated with the key (e.g., "8080", "localhost").

Methods:

- getKey(): Returns the key.

- getValue(): Returns the value.

- setValue(new_value): Updates the value for the setting.

3. Project Structure and Steps:

Step 1: Implement the ConfigurationReader class with attributes for the file path and a data structure to store the key-value pairs.

Step 2: Implement the readFile() method to open the file, read its contents line by line, and pass each line to the parseLine() method.

Step 3: Implement the parseLine() method to separate the key and value based on a delimiter (such as '=' or ':').

Step 4: Store each key-value pair in a map or dictionary for quick access.

Step 5: Implement methods to retrieve settings by key and update them if necessary.

Step 6: Test the ConfigurationReader class by creating a sample configuration file and verifying that the key-value pairs are read and stored correctly.