```cpp
#include "sdcardIO.h"
#include "../vars/constants.h"
#include "FS.h"
#include "SD.h"
#include "SPI.h"
#include <vector>
#include <tuple>

//@brief Mount the SDCard
//@return 0: If mounting was successful;
//@return 1: If communication with SDCard module failed;
//@return 2: If no card is attached
uint8_t sdcardIO::MountCard()
{
  if(!SD.begin(33))
  {
    //SD-Card mount failed
    return 1;
  }
  if(SD.cardType() == CARD_NONE)
  {
    //No card attached
    return 2;
  }
  return 0;
}

//@brief Get Size of SDCard
uint64_t sdcardIO::GetCardSize()
{
  return SD.cardSize();
}

//@brief Get Total Bytes of SDCard
uint64_t sdcardIO::GetTotalBytes()
{
  return SD.totalBytes();
}

//@brief Get Used Bytes of SDCard
uint64_t sdcardIO::GetUsedBytes()
{
  return SD.usedBytes();
}

//@brief Get Free Bytes of SDCard
uint64_t sdcardIO::GetFreeBytes()
{
  return GetTotalBytes() - GetUsedBytes();
}

//@brief Get list of all Files in a Directory
//@param &fs The Filesystem
```

```cpp
//@param *dirname The path to the directory
//@param levels How deep to search in subdirectories
//@param deep Internal use only, leave out default or set to zero (unless you know what you are doing)
//@return A Vector of a Tuple of uint8_t (EntryType), uint8_t (How deep the subdirectory is), string (path of entry), size_t (size of entry)
std::vector<std::tuple<uint8_t, uint8_t, std::string, size_t>> sdcardIO::Directory::List(fs::FS &fs, const char * dirname, uint8_t levels, uint8_t deep)
{
  //EntryType, Level, Path, Size
  std::vector<std::tuple<uint8_t, uint8_t, std::string, size_t>> entryList;
  fs::File root = fs.open(dirname);
  if(!root){
    entryList.push_back(std::make_tuple(DirectoryEntryType::UNKNOWN, deep, ("Failed to open directory: " + std::string(dirname)).c_str(), 0));
    return entryList;
  }
  if(!root.isDirectory()){
    entryList.push_back(std::make_tuple(DirectoryEntryType::UNKNOWN, deep, "Not a directory", 0));
    return entryList;
  }

  fs::File file = root.openNextFile();
  while(file){
    if(file.isDirectory()){
      entryList.push_back(std::make_tuple(DirectoryEntryType::Directory, deep, (std::string(file.path()) + "/").c_str(), file.size()));
      if(levels){
        std::vector<std::tuple<uint8_t, uint8_t, std::string, size_t>> tmpList = sdcardIO::Directory::List(fs, file.path(), levels - 1, deep + 1);
        entryList.insert(entryList.end(), tmpList.begin(), tmpList.end());
      }
    }
    else
    {
      entryList.push_back(std::make_tuple(DirectoryEntryType::File, deep, file.path(), file.size()));
    }
     file = root.openNextFile();
  }

  return entryList;
}

//@brief Create Directory
//@param &fs The Filesystem
```

```cpp
//@param path The path of the directory to create
//@return True if creation was successful, otherwise false
bool sdcardIO::Directory::Create(fs::FS &fs, const char * path)
{
  if(fs.mkdir(path))
  {
      return true;
  }
  return false;
}

//@brief Delete Directory
//@param &fs The Filesystem
//@param path The path of the directory to remove
//@return True if deletion was successful, otherwise false
bool sdcardIO::Directory::Remove(fs::FS &fs, const char * path)
{
    if(fs.rmdir(path))
    {
        return true;
    }
    return false;
}

//@brief Read file from SDCard and print to Serial Console
//@param &fs The Filesystem
//@param path The path of the file to read
void sdcardIO::File::Read(fs::FS &fs, const char * path)
{
    fs::File file = fs.open(path);
    if(!file)
    {
        Serial.println("Failed to open file for reading");
        return;
    }

    Serial.print("Reading from file: ");
    Serial.println(path);
    while(file.available())
    {
        Serial.write(file.read());
    }
    file.close();
    Serial.println();
}

//@brief Write something to a file
//@param &fs The Filesystem
//@param path The path of the file
//@param content The content to write to the file
//@return 0: If writing was successful;
//@return 1: If file failed to open for writing
//@return 2: If writing failed
```

```cpp
uint8_t sdcardIO::File::Write(fs::FS &fs, const char * path, const char * content)
{
  fs::File file = fs.open(path, FILE_WRITE);
  if(!file)
  {
    //Failed to open file for writing
    return 1;
  }

  if(file.print(content))
  {
    file.close();
    return 0;
  }
  file.close();
  //Write Failed
  return 2;
}

//@brief Append something to a file
//@param &fs The Filesystem
//@param path The path of the file
//@param content The content to append to the file
//@return 0: If appending was successful;
//@return 1: If file failed to open for appending
//@return 2: If appending failed
uint8_t sdcardIO::File::Append(fs::FS &fs, const char * path, const char * content)
{
  fs::File file = fs.open(path, FILE_APPEND);
  if(!file)
  {
    //Failed to open for appending
    return 1;
  }

  if(file.print(content))
  {
    file.close();
    return 0;
  }

  file.close();
  //Append failed
  return 2;
}

//@brief Rename a file
//@param &fs The Filesystem
//@param path1 The original path of the file
//@param path2 The new path of the file
//@return true if renaming was successful, otherwise return false
```

```cpp
bool sdcardIO::File::Rename(fs::FS &fs, const char * path1, const char * path2)
{
  if(fs.rename(path1, path2))
  {
    return true;
  }
  return false;
}

//@brief Delete a file
//@param &fs The Filesystem
//@param path1 The path of the file
//@return true if removing was successful, otherwise return false
bool sdcardIO::File::Delete(fs::FS &fs, const char * path)
{
  if(fs.remove(path))
  {
    return true;
  }
  return false;
}

//@brief Read the constanst from a file on the SD-Card
//@param &fs The Filesystem
//@return 0: If reading was successful
//@return 1: If file does not exist
//@return 2: If file failed to open
uint8_t sdcardIO::ReadConstants(fs::FS &fs)
{
  if(!fs.exists(constants::sdcardIO::ConfigFilePath))
  {
    //Config file does not exist
    return 1;
  }

  fs::File file = fs.open(constants::sdcardIO::ConfigFilePath, FILE_READ);
  if(!file)
  {
      //File failed to open
      return 2;
  }
  while(file.available())
  {
      String line = file.readStringUntil('\n');

      std::string key = line.substring(0, line.indexOf('=')).c_str();
      std::string value = line.substring(line.indexOf('=') + 1).c_str();

      if(!constants::setValue(key.c_str(), value.c_str()))
      {
            Serial.print("Error setting value: ");
            Serial.println(key.c_str());
        }
    }
    file.close();
    return 0;
}
```