**Learning C  
The following code was able to print my name in the console   
Was also able to open and read my file in d:\Documents\Vs Code Projects\Coding Assignments\Group 1\school.c  
Was also able to prints out the content in that .c file in the console**  
#include<stdio.h>

int main()

{

printf("hello Katchiets");

FILE \*file;

char line[100]; *// Assuming lines in the file are no longer than 100 characters.*

*// Open the file for reading.*

file = fopen("D:**\\**Documents**\\**Vs Code Projects**\\**Coding Assignments**\\**Group 1**\\**school.c", "r");

if (file == NULL)

{

printf("Error opening the file.**\n**");

return 1;

}

*// Read and print each line until the end of the file.*

while (fgets(line, sizeof(line), file) != NULL)

{

printf("%s", line);

}

*// Close the file.*

fclose(file);

}

In C strings, backslashes () are used as escape characters to represent special characters. For example, the escape sequence **\n** is used to represent a newline character, **\t** for a tab character, etc. Because backslashes have this special meaning, if you want to include an actual backslash character in a C string, you need to escape it with another backslash.

Here's a brief explanation of how escaping works in C strings:

1. **Single Backslash**: If you want to include a single backslash in a C string, you need to escape it by using another backslash. For example, to represent a string containing a single backslash, you write **"\\\\"**. The first backslash escapes the second one, resulting in a single backslash in the actual string.
2. **Forward Slash**: Forward slashes (/) do not have a special meaning in C strings, so they don't need to be escaped. You can use forward slashes as regular characters in C strings. For example, **"folder/file.txt"** is a valid C string.
3. **Path Separators**: In Windows file paths, backslashes are used as path separators (e.g., **C:\Documents\file.txt**). However, since backslashes are escape characters in C strings, you need to use double backslashes (\) to represent a single backslash in the path. So, in C strings, a Windows file path would look like **"C:\\Documents\\file.txt"**.

Here's an example to illustrate this with a C string containing a Windows file path:

#include <stdio.h>

int main()

{

*// A C string representing a Windows file path*

char path[] = "D:**\\**Documents**\\**Vs Code Projects**\\**Coding Assignments**\\**Group 1**\\**school.c";

*// Print the path to the console*

printf("File Path: %s**\n**", path);

return 0;

}

In this example, the **path** variable contains a C string representing a Windows file path. We use double backslashes to escape each backslash, ensuring that the path is correctly represented as a C string. When we print the **path** variable using **printf()**, it will display the correct file path:

File Path: D:\Documents\Vs Code Projects\Coding Assignments\Group 1\school.c

Remember that if you're using forward slashes in your file path, you don't need to escape them. They can be used as regular characters in C strings. However, using double backslashes for path separators is a common practice to ensure portability and consistency across different platforms.

**This failed execution**

#include <stdio.h>

#include <dirent.h>

#include <string.h>

void listDirectoriesAndFiles(const char \*path);

int main()

{

printf("Listing directories and files in D:**\\**Documents:**\n**");

listDirectoriesAndFiles("D:**\\**Documents");

return 0;

}

void listDirectoriesAndFiles(const char \*path)

{

DIR \*dir;

struct dirent \*entry;

char folders[1000][FILENAME\_MAX]; *// Array to store unique folder names*

int numFolders = 0; *// Counter for the number of unique folders*

dir = opendir(path);

if (dir == NULL)

{

printf("Error opening directory: %s**\n**", path);

return;

}

*// Loop through the directory entries and find unique folder names.*

while ((entry = readdir(dir)) != NULL)

{

*// Skip the "." and ".." entries and non-directory entries.*

if (entry->d\_type != DT\_DIR || strcmp(entry->d\_name, ".") == 0 || strcmp(entry->d\_name, "..") == 0)

continue;

int found = 0;

*// Check if the folder name is already in the list of unique folders.*

for (int i = 0; i < numFolders; i++)

{

if (strcmp(folders[i], entry->d\_name) == 0)

{

found = 1;

**break**;

}

}

if (!found)

{

*// If the folder name is not found in the list, add it to the list.*

strcpy(folders[numFolders], entry->d\_name);

numFolders++;

}

}

closedir(dir);

*// Print the summary of unique folder names.*

printf("Folders present in D:**\\**Documents include (");

for (int i = 0; i < numFolders; i++)

{

printf("%s", folders[i]);

if (i < numFolders - 1)

printf(", ");

}

printf(")**\n**");

}

**The following code failed execution:**

#include <stdio.h>

#include <windows.h>

void listDirectoriesInRoot(const wchar\_t \*path);

int main()

{

printf("Listing directories in D:**\\**Documents:**\n**");

listDirectoriesInRoot(L"D:**\\**Documents"); *// Note the L before the string to create a wide-character string*

return 0;

}

void listDirectoriesInRoot(const wchar\_t \*path)

{

WIN32\_FIND\_DATAW findFileData; *// Use the wide-character version of the structure*

HANDLE hFind = FindFirstFileW(path, &findFileData); *// Use the wide-character version of the function*

if (hFind == INVALID\_HANDLE\_VALUE)

{

printf("Error finding directories in: %S**\n**", path); *// Use %S to print wide-character strings*

return;

}

*// Loop through the directory entries and find directories.*

do

{

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)

{

wprintf(L"Directory: %s**\n**", findFileData.cFileName);

}

}

} while (FindNextFileW(hFind, &findFileData) != 0); *// Use the wide-character version of the function*

FindClose(hFind);

}

**This also failed Execution**

#include <stdio.h>

#include <windows.h>

void listDirectoriesInRoot(const char \*path);

int main()

{

printf("Listing directories in D:**\\**Documents:**\n**");

listDirectoriesInRoot("D:**\\**Documents");

return 0;

}

void listDirectoriesInRoot(const char \*path)

{

WIN32\_FIND\_DATA findFileData;

HANDLE hFind = FindFirstFile((LPCWSTR)path, &findFileData);

if (hFind == INVALID\_HANDLE\_VALUE)

{

printf("Error finding directories in: %s**\n**", path);

return;

}

*// Loop through the directory entries and find directories.*

do

{

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)

{

wprintf(L"Directory: %s**\n**", findFileData.cFileName);

}

}

} while (FindNextFile(hFind, &findFileData) != 0);

FindClose(hFind);

}

**The following code was able to access my D:/Documents path and return all the directories in the root**

#include <stdio.h>

#include <windows.h>

void listDirectoriesInRoot(const wchar\_t \*path);

int main()

{

printf("Listing directories in D:**\\**Documents:**\n**");

listDirectoriesInRoot(L"D:**\\**Documents");

return 0;

}

void listDirectoriesInRoot(const wchar\_t \*path)

{

WIN32\_FIND\_DATAW findFileData;

HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

if (hFind == INVALID\_HANDLE\_VALUE)

{

printf("Error finding directories in: %S**\n**", path);

return;

}

*// Loop through the directory entries and find directories.*

do

{

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)

{

wprintf(L"Directory: %s**\n**", findFileData.cFileName);

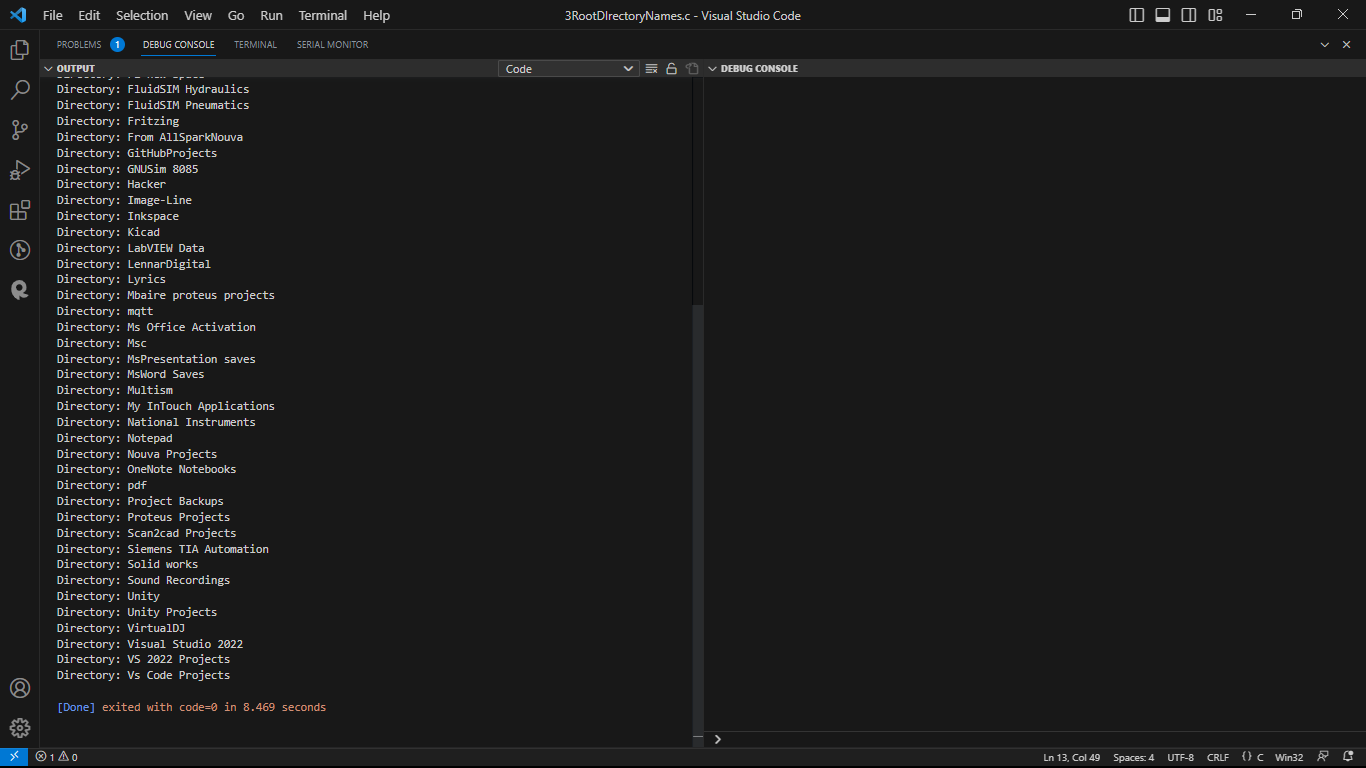
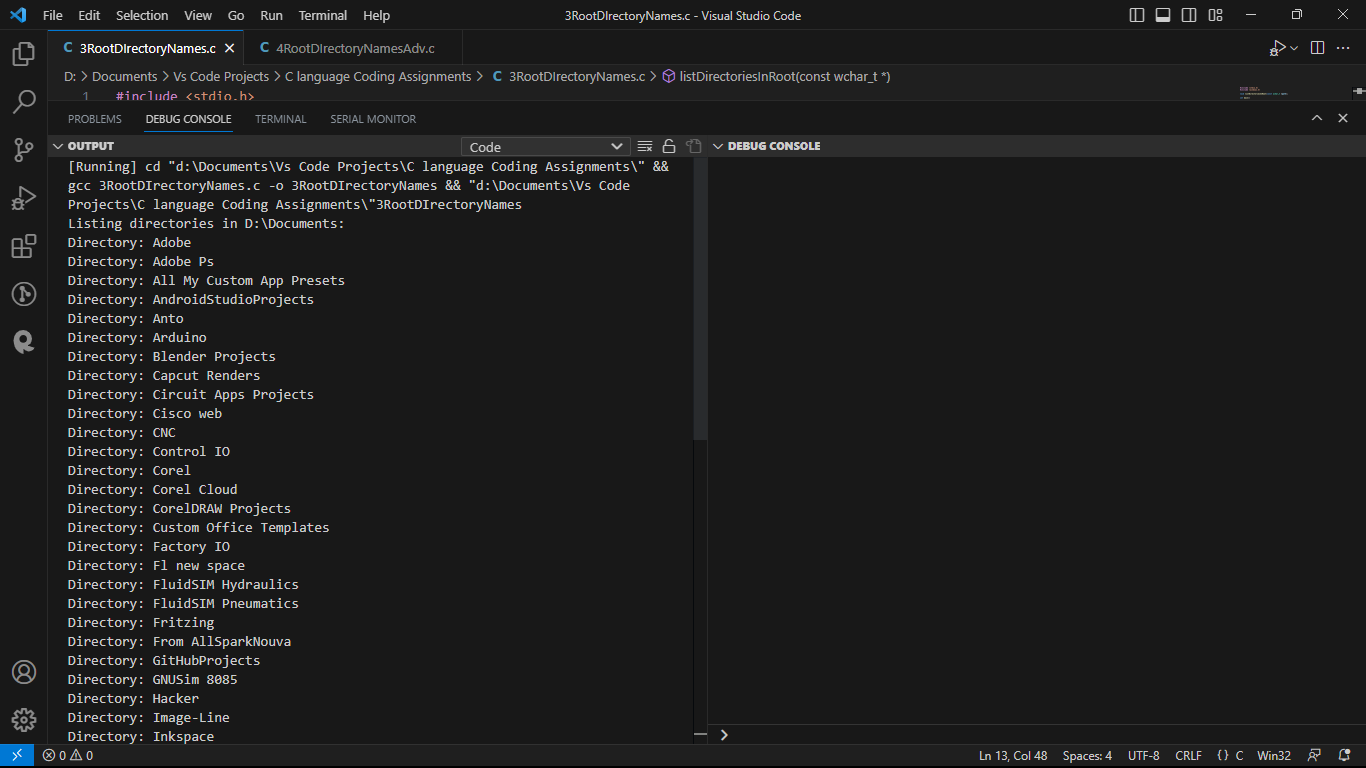
}

}

} while (FindNextFileW(hFind, &findFileData) != 0);

FindClose(hFind);

}



**The following code returned me the files and directories present in the root but in alphabetical order:**

#include <stdio.h>

#include <windows.h>

void listDirectoriesAndFilesInRoot(const wchar\_t \*path);

int main()

{

printf("Listing directories and files in D:**\\**Documents:**\n**");

listDirectoriesAndFilesInRoot(L"D:**\\**Documents");

return 0;

}

void listDirectoriesAndFilesInRoot(const wchar\_t \*path)

{

WIN32\_FIND\_DATAW findFileData;

HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

if (hFind == INVALID\_HANDLE\_VALUE)

{

printf("Error finding directories and files in: %S**\n**", path);

return;

}

*// Loop through the directory entries and find directories and non-directory files.*

do

{

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)

{

wprintf(L"Directory: %s**\n**", findFileData.cFileName);

}

}

else

{

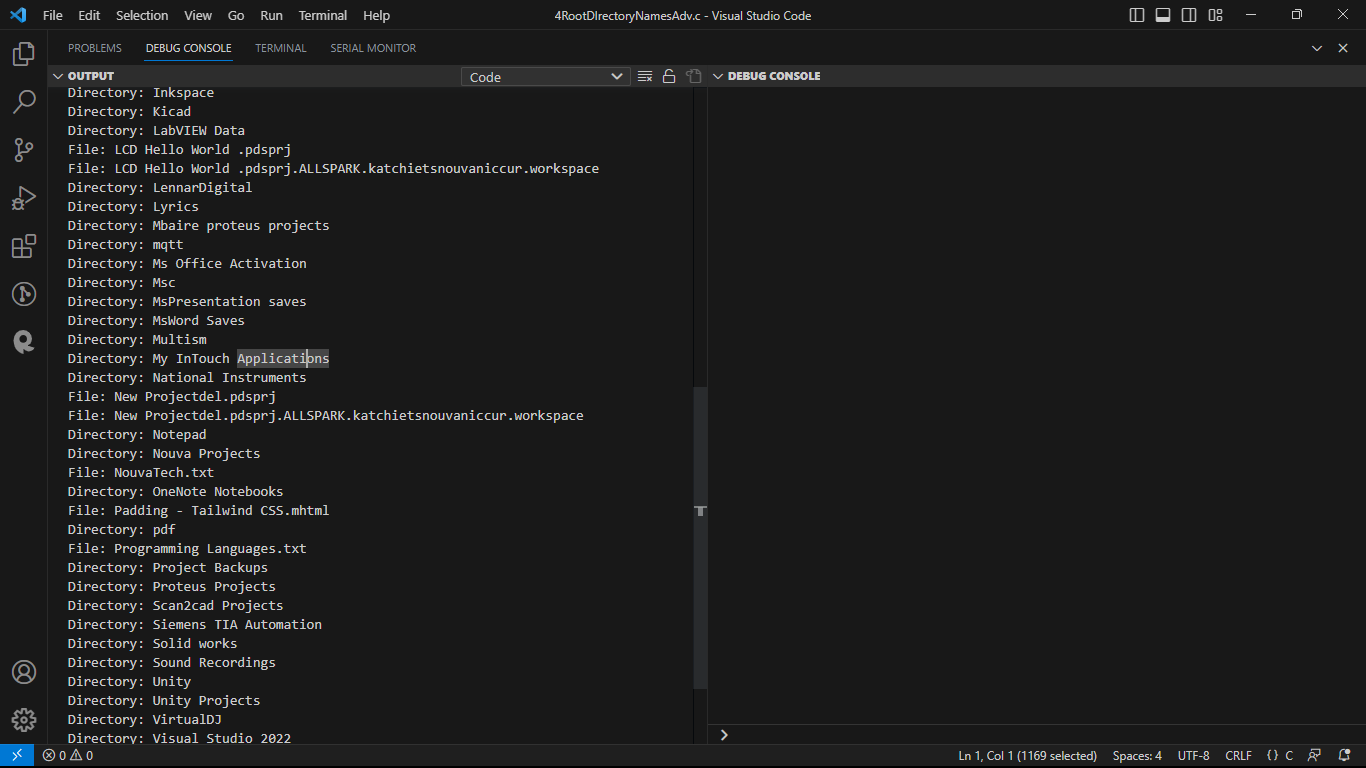
wprintf(L"File: %s**\n**", findFileData.cFileName);

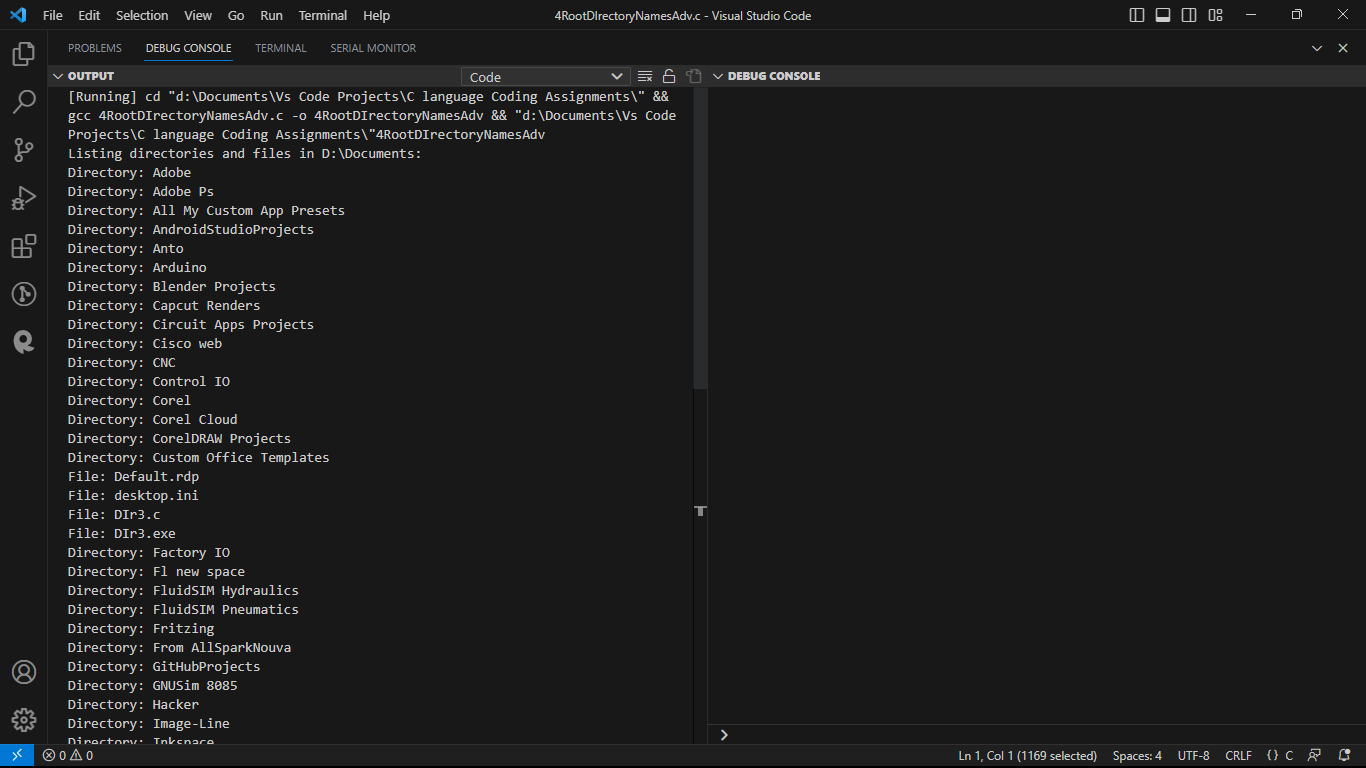
}

} while (FindNextFileW(hFind, &findFileData) != 0);

FindClose(hFind);

}



**gg**

**The following returned in a list form:**

#include <stdio.h>

#include <windows.h>

void listDirectoriesAndFilesInRoot(const wchar\_t \*path);

int main()

{

printf("Listing directories and files in D:**\\**Documents:**\n**");

listDirectoriesAndFilesInRoot(L"D:**\\**Documents");

return 0;

}

void listDirectoriesAndFilesInRoot(const wchar\_t \*path)

{

WIN32\_FIND\_DATAW findFileData;

HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

if (hFind == INVALID\_HANDLE\_VALUE)

{

printf("Error finding directories and files in: %S**\n**", path);

return;

}

*// Collect directories and non-directory files separately*

wprintf(L"Directories:**\n**");

do

{

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)

{

wprintf(L"Directory: %s**\n**", findFileData.cFileName);

}

}

} while (FindNextFileW(hFind, &findFileData) != 0);

*// Close the handle and reopen it to reset the search*

FindClose(hFind);

hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

*// Collect non-directory files*

wprintf(L"Files:**\n**");

do

{

if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))

{

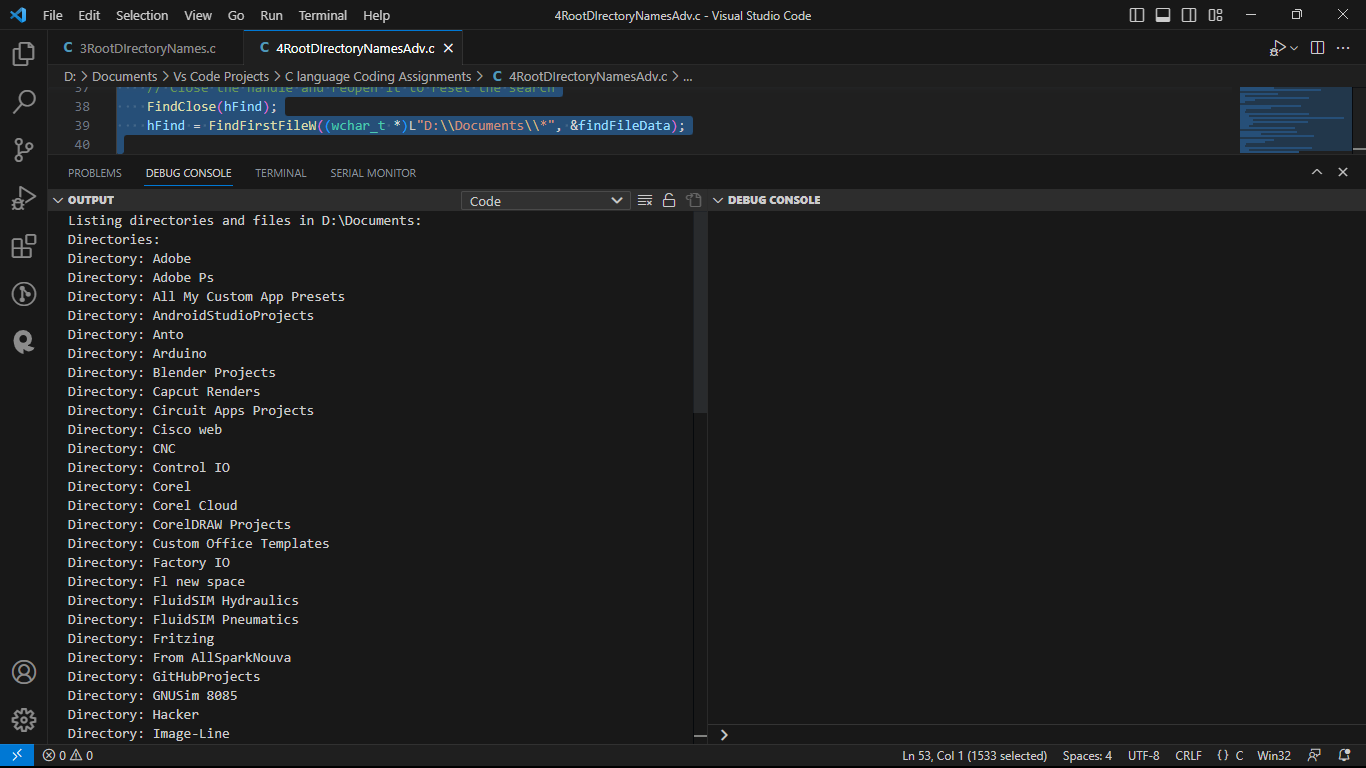
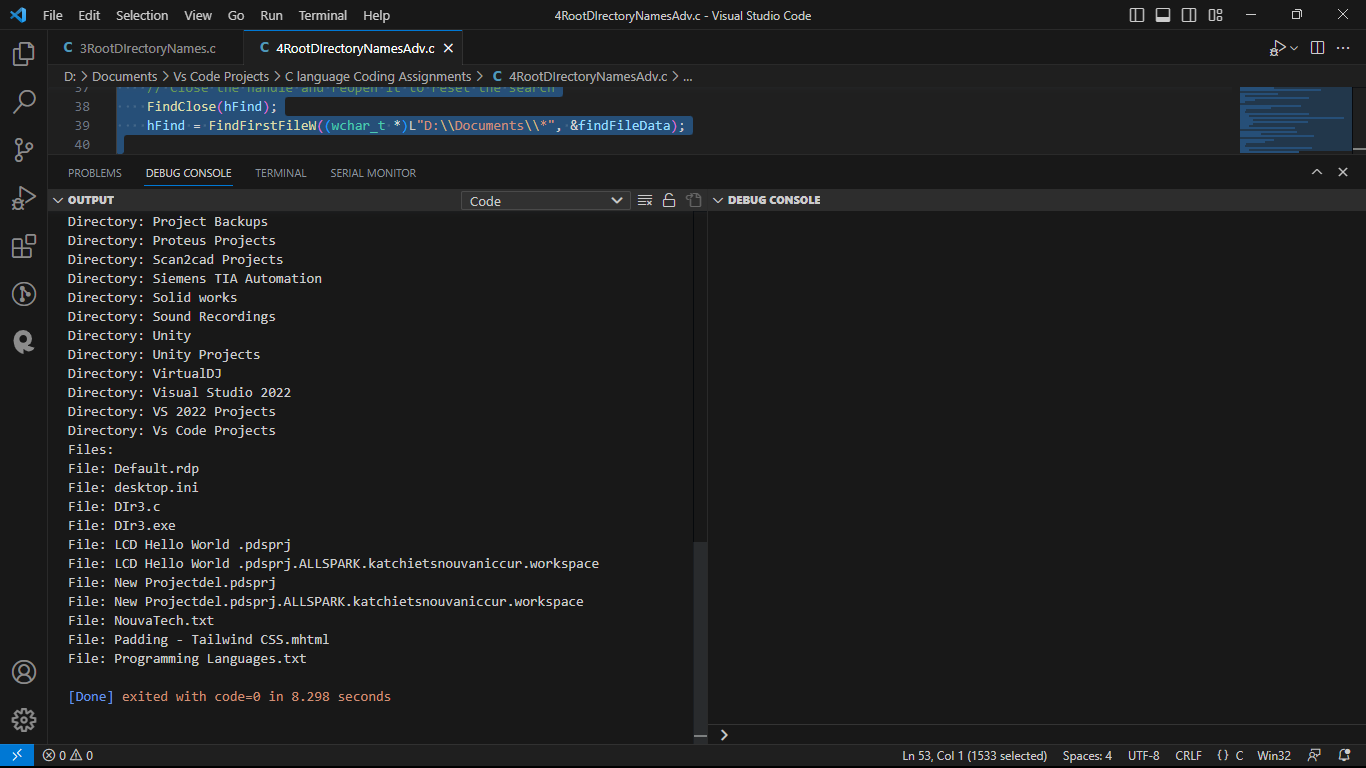
wprintf(L"File: %s**\n**", findFileData.cFileName);

}

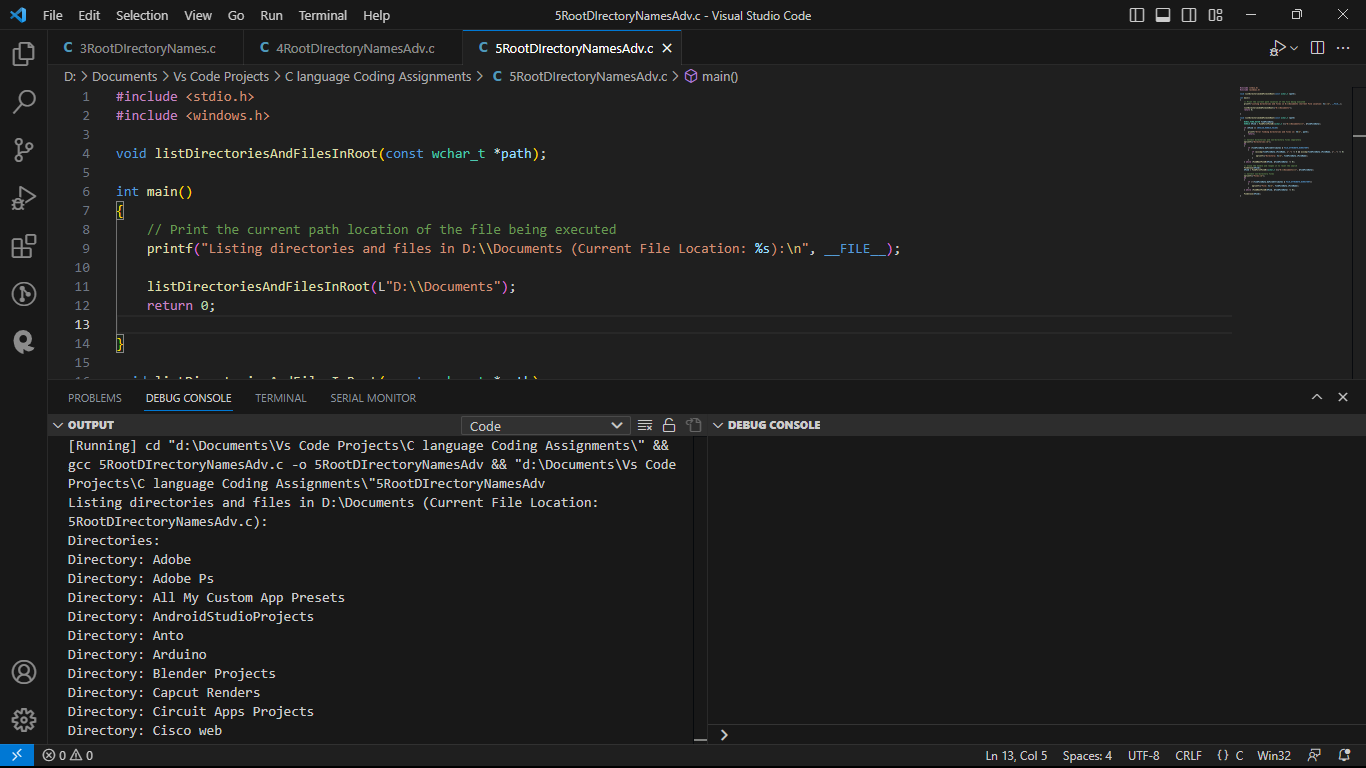
} while (FindNextFileW(hFind, &findFileData) != 0);

FindClose(hFind);

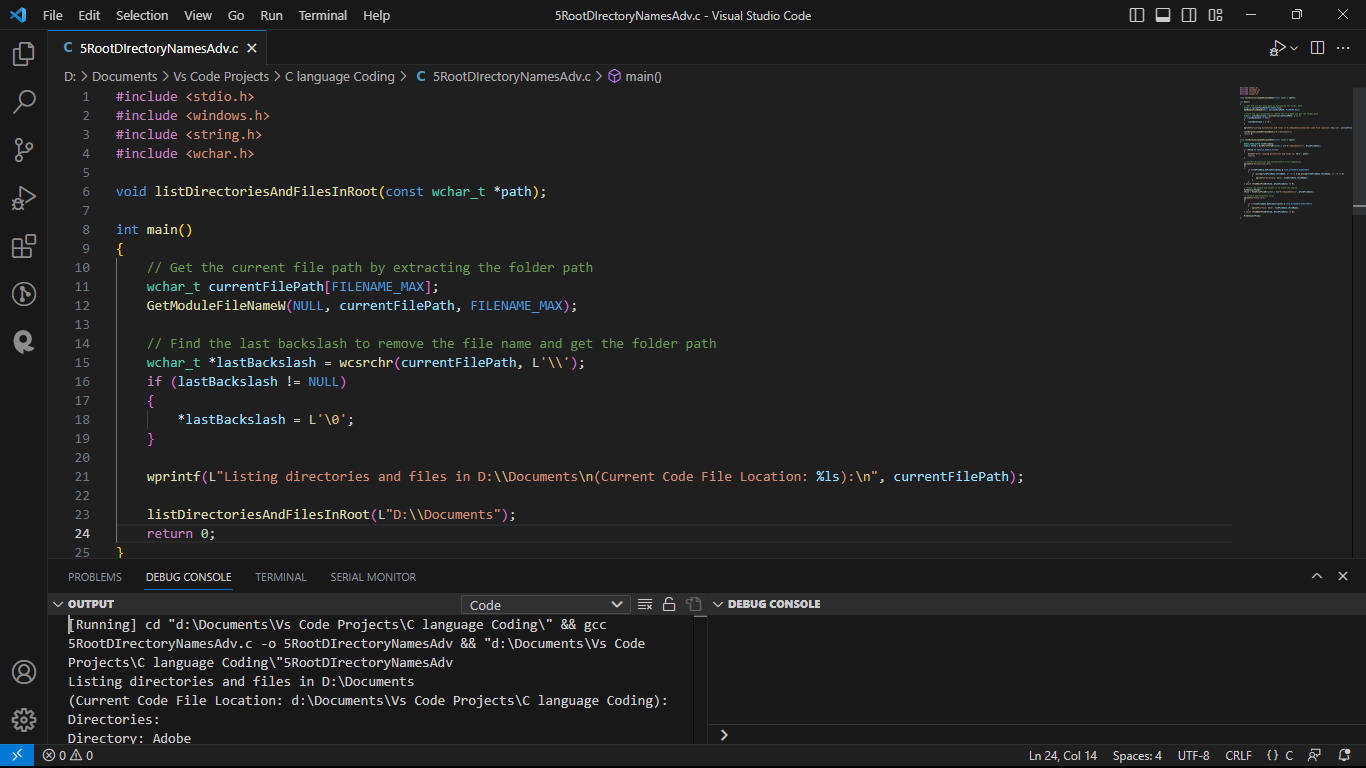
}

**Retrieving code path:**



**Retrieving code path advanced v1:**



**Retrieving code path advanced v2:**The code was able to find:

1. Find the Current Code File Location of the parent directory:   
   *(Current Code File Location: d:\Documents\Vs Code Projects\C language Coding)*
2. Find the path of the executable file contained:   
   **(***Current Code File: d:\Documents\Vs Code Projects\C language Coding\5RootDIrectoryNamesAdv.exe)*
3. Provide a listing of directories and files in D:\Documents, and it 1st lists the directories then files, in alphabetical ordere.g.,
4. Directories:  
   Directory: Adobe

Directory: Adobe Ps  
Directory: All My Custom App Presets

Directory: Blender Projects

Directory: Unity Projects

1. Files:

File: desktop.ini

File: DIr3.c

File: DIr3.exe

File: LCD Hello World .pdsprj

#include <stdio.h>

#include <windows.h>

#include <string.h>

#include <wchar.h>

void listDirectoriesAndFilesInRoot(const wchar\_t \*path);

int main()

{

*// Get the current executable file path*

wchar\_t currentExecutablePath[FILENAME\_MAX];

GetModuleFileNameW(NULL, currentExecutablePath, FILENAME\_MAX);

*// Find the last backslash to remove the file name and get the folder path*

wchar\_t \*lastBackslash = wcsrchr(currentExecutablePath, L'**\\**');

if (lastBackslash != NULL)

{

\*lastBackslash = L'**\0**';

}

*// Get the current code file path*

wchar\_t currentCodeFilePath[FILENAME\_MAX];

GetModuleFileNameW(NULL, currentCodeFilePath, FILENAME\_MAX);

*// Print the current code file location and the current code file*

wprintf(L"Listing directories and files in D:**\\**Documents**\n**(Current Code File Location: %ls)**\n**(Current Code File: %ls):**\n**", currentExecutablePath, currentCodeFilePath);

listDirectoriesAndFilesInRoot(L"D:**\\**Documents");

return 0;

}

void listDirectoriesAndFilesInRoot(const wchar\_t \*path)

{

WIN32\_FIND\_DATAW findFileData;

HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

if (hFind == INVALID\_HANDLE\_VALUE)

{

printf("Error finding directories and files in: %S**\n**", path);

return;

}

*// Collect directories and non-directory files separately*

wprintf(L"Directories:**\n**");

do

{

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)

{

wprintf(L"Directory: %s**\n**", findFileData.cFileName);

}

}

} while (FindNextFileW(hFind, &findFileData) != 0);

*// Close the handle and reopen it to reset the search*

FindClose(hFind);

hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

*// Collect non-directory files*

wprintf(L"Files:**\n**");

do

{

if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))

{

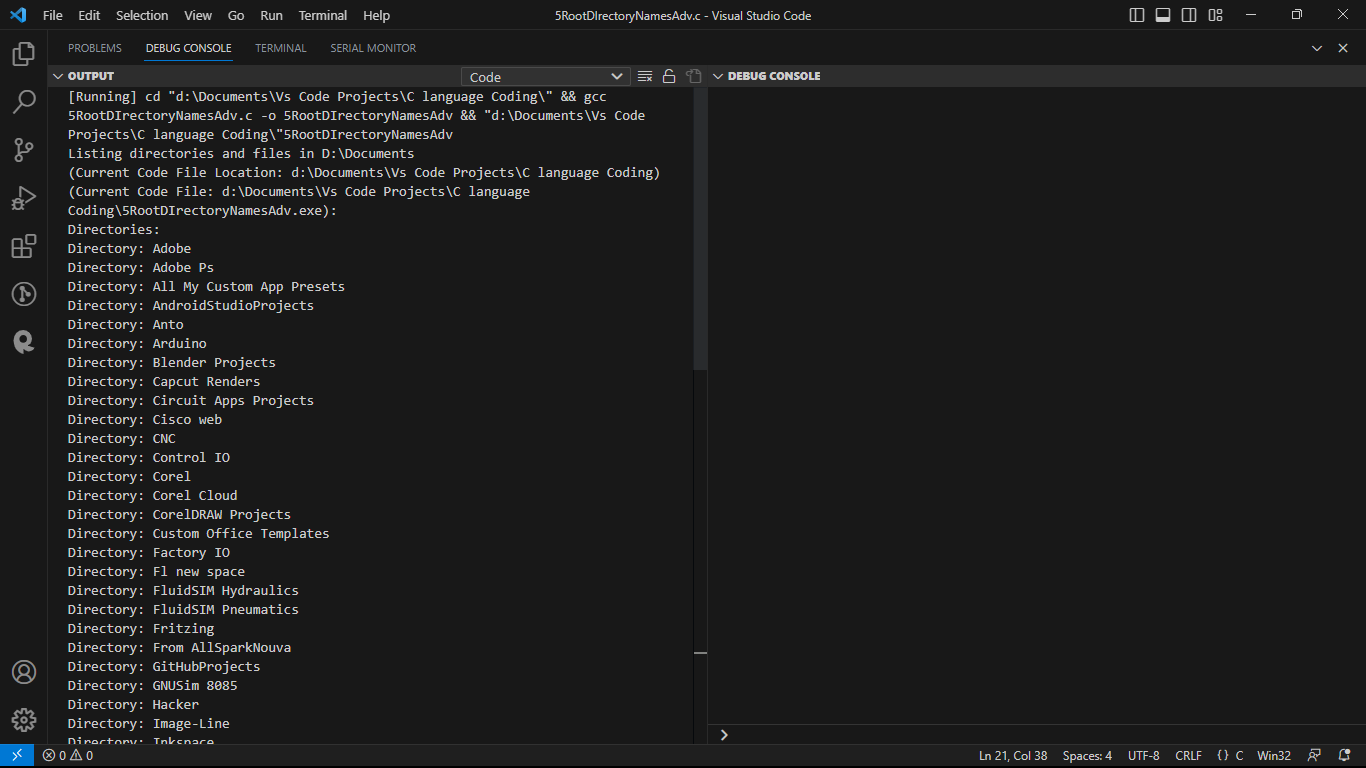
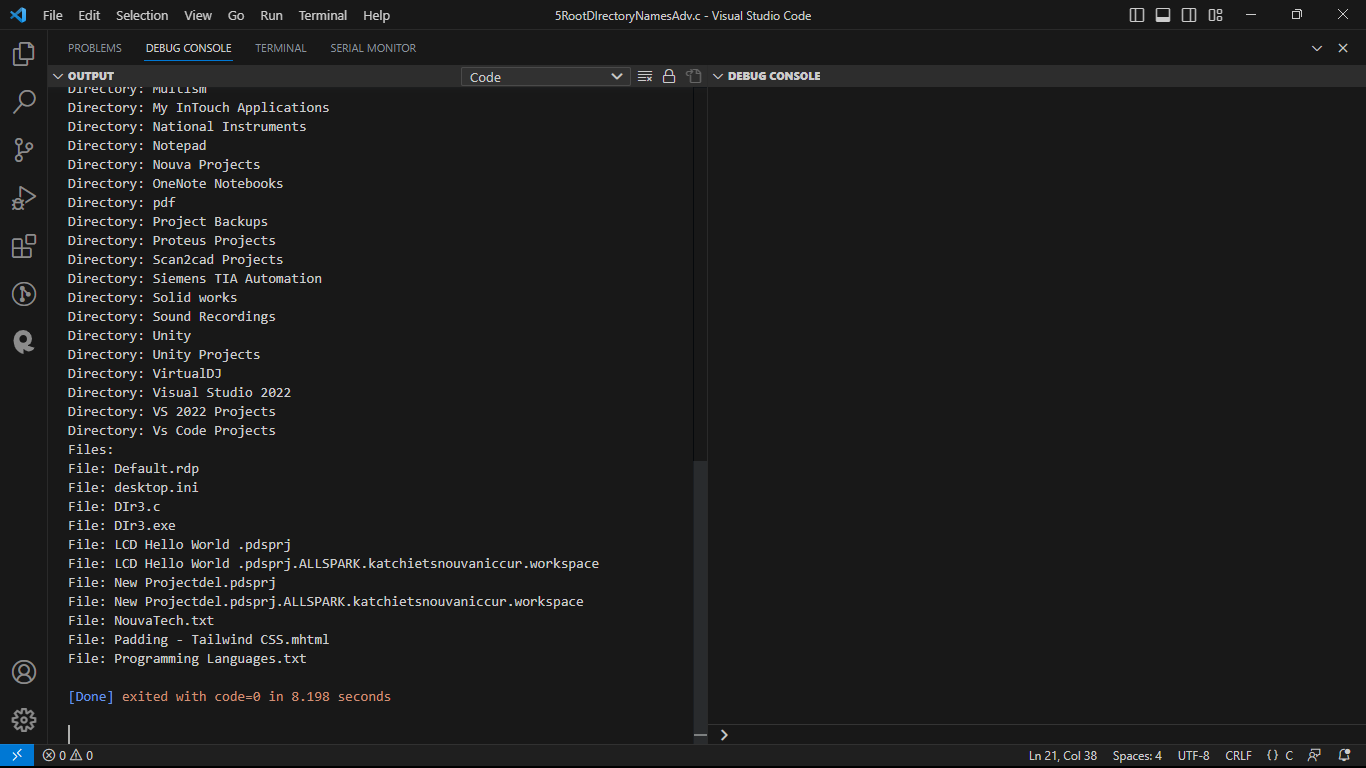
wprintf(L"File: %s**\n**", findFileData.cFileName);

}

} while (FindNextFileW(hFind, &findFileData) != 0);

FindClose(hFind);

}

The following code was able to add a bin folder: NB added in the int main() part only . however the fie executable path wasn’t obtained correctly

#include <stdio.h>

#include <windows.h>

#include <string.h>

#include <wchar.h>

void listDirectoriesAndFilesInRoot(const wchar\_t \*path);

int main()

{

*// Get the current executable file path*

wchar\_t currentExecutablePath[FILENAME\_MAX];

GetModuleFileNameW(NULL, currentExecutablePath, FILENAME\_MAX);

*// Find the last backslash to remove the file name and get the folder path*

wchar\_t \*lastBackslash = wcsrchr(currentExecutablePath, L'**\\**');

if (lastBackslash != NULL)

{

\*lastBackslash = L'**\0**';

}

*// Create a copy of the current code file path before modifying it*

wchar\_t currentCodeFilePath[FILENAME\_MAX];

wcscpy(currentCodeFilePath, currentExecutablePath);

*// Get the current code file path*

wchar\_t currentCodeFileFullPath[FILENAME\_MAX];

GetModuleFileNameW(NULL, currentCodeFileFullPath, FILENAME\_MAX);

*// Remove the executable file name from the current code file path*

wchar\_t \*lastBackslashCodeFile = wcsrchr(currentCodeFileFullPath, L'**\\**');

if (lastBackslashCodeFile != NULL)

{

\*lastBackslashCodeFile = L'**\0**';

}

*// Create the "bin" directory in the current code file path*

wcscat(currentCodeFilePath, L"**\\**bin");

CreateDirectoryW(currentCodeFilePath, NULL);

*// Get just the file name from the current code file path*

wchar\_t \*fileName = wcsrchr(currentCodeFileFullPath, L'**\\**');

if (fileName != NULL)

{

fileName++; *// Move past the backslash to the actual file name*

}

else

{

fileName = L"Unknown"; *// Fallback in case of any issue with the file name*

}

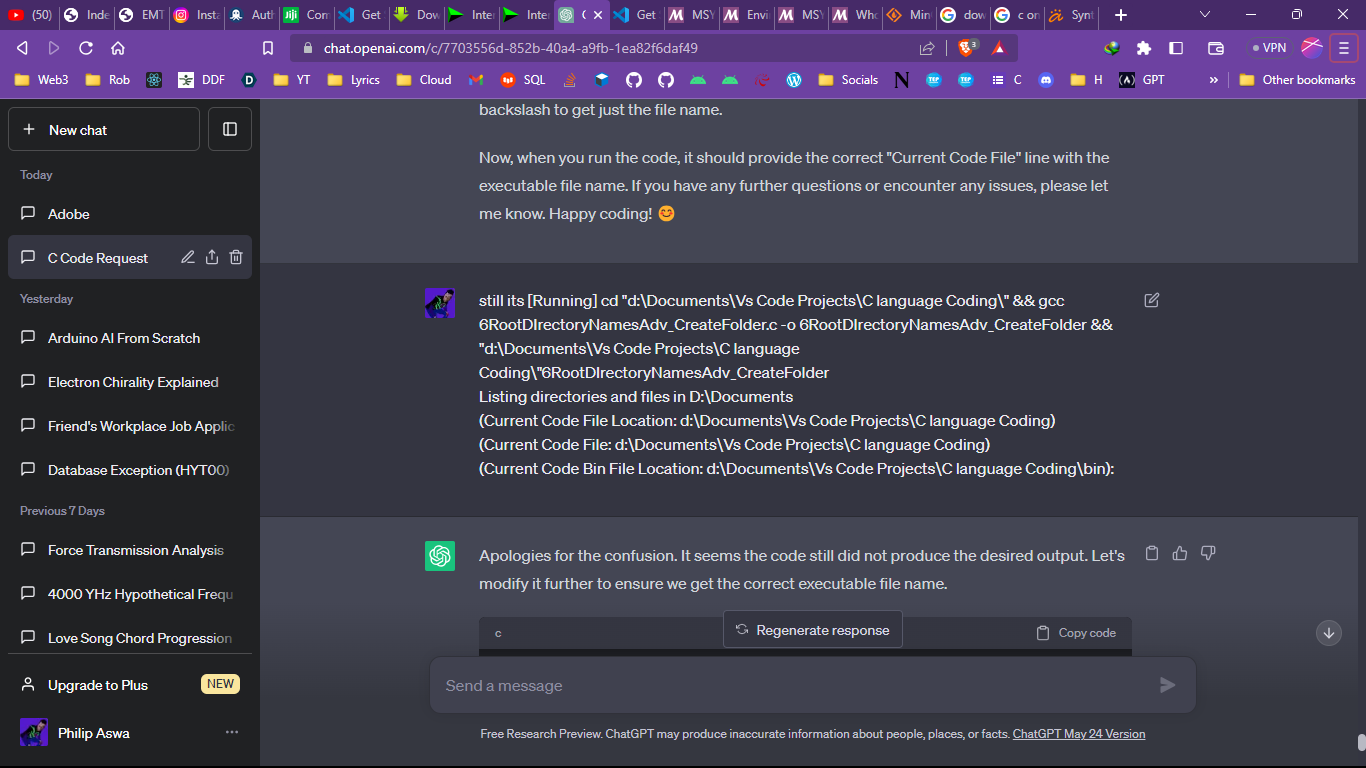
*// Print the current code file location and the current code file*

wprintf(L"Listing directories and files in D:**\\**Documents**\n**(Current Code File Location: %ls)**\n**(Current Code File: %ls)**\n**(Current Code Bin File Location: %ls):**\n**", currentExecutablePath, currentCodeFileFullPath, currentCodeFilePath);

listDirectoriesAndFilesInRoot(L"D:**\\**Documents");

return 0;

}



**This was able to return the right output:**

#include <stdio.h>

#include <windows.h>

#include <string.h>

#include <wchar.h>

void listDirectoriesAndFilesInRoot(const wchar\_t \*path);

int main()

{

*// Get the current executable file path*

wchar\_t currentExecutablePath[FILENAME\_MAX];

GetModuleFileNameW(NULL, currentExecutablePath, FILENAME\_MAX);

*// Find the last backslash to remove the file name and get the folder path*

wchar\_t \*lastBackslash = wcsrchr(currentExecutablePath, L'**\\**');

if (lastBackslash != NULL)

{

\*lastBackslash = L'**\0**';

}

*// Get just the file name from the current executable path*

wchar\_t \*fileName = wcsrchr(currentExecutablePath, L'**\\**');

if (fileName != NULL)

{

fileName++; *// Move past the backslash to the actual file name*

}

else

{

fileName = L"Unknown"; *// Fallback in case of any issue with the file name*

}

*// Create a copy of the current code file path before modifying it*

wchar\_t currentCodeFilePath[FILENAME\_MAX];

wcscpy(currentCodeFilePath, currentExecutablePath);

*// Create the "bin" directory in the current code file path*

wcscat(currentCodeFilePath, L"**\\**bin");

CreateDirectoryW(currentCodeFilePath, NULL);

*// Get the current code file name with .exe extension*

wchar\_t currentCodeFileName[FILENAME\_MAX];

GetModuleFileNameW(NULL, currentCodeFileName, FILENAME\_MAX);

*// Print the current code file location and the current code file*

wprintf(L"Listing directories and files in D:**\\**Documents**\n**(Current Code File Location: %ls)**\n**(Current Code File: %ls)**\n**(Current Code Bin File Location: %ls):**\n**", currentExecutablePath, currentCodeFileName, currentCodeFilePath);

listDirectoriesAndFilesInRoot(L"D:**\\**Documents");

return 0;

}

void listDirectoriesAndFilesInRoot(const wchar\_t \*path)

{

WIN32\_FIND\_DATAW findFileData;

HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

if (hFind == INVALID\_HANDLE\_VALUE)

{

printf("Error finding directories and files in: %S**\n**", path);

return;

}

*// Collect directories and non-directory files separately*

wprintf(L"Directories:**\n**");

do

{

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)

{

wprintf(L"Directory: %s**\n**", findFileData.cFileName);

}

}

} while (FindNextFileW(hFind, &findFileData) != 0);

*// Close the handle and reopen it to reset the search*

FindClose(hFind);

hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

*// Collect non-directory files*

wprintf(L"Files:**\n**");

do

{

if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))

{

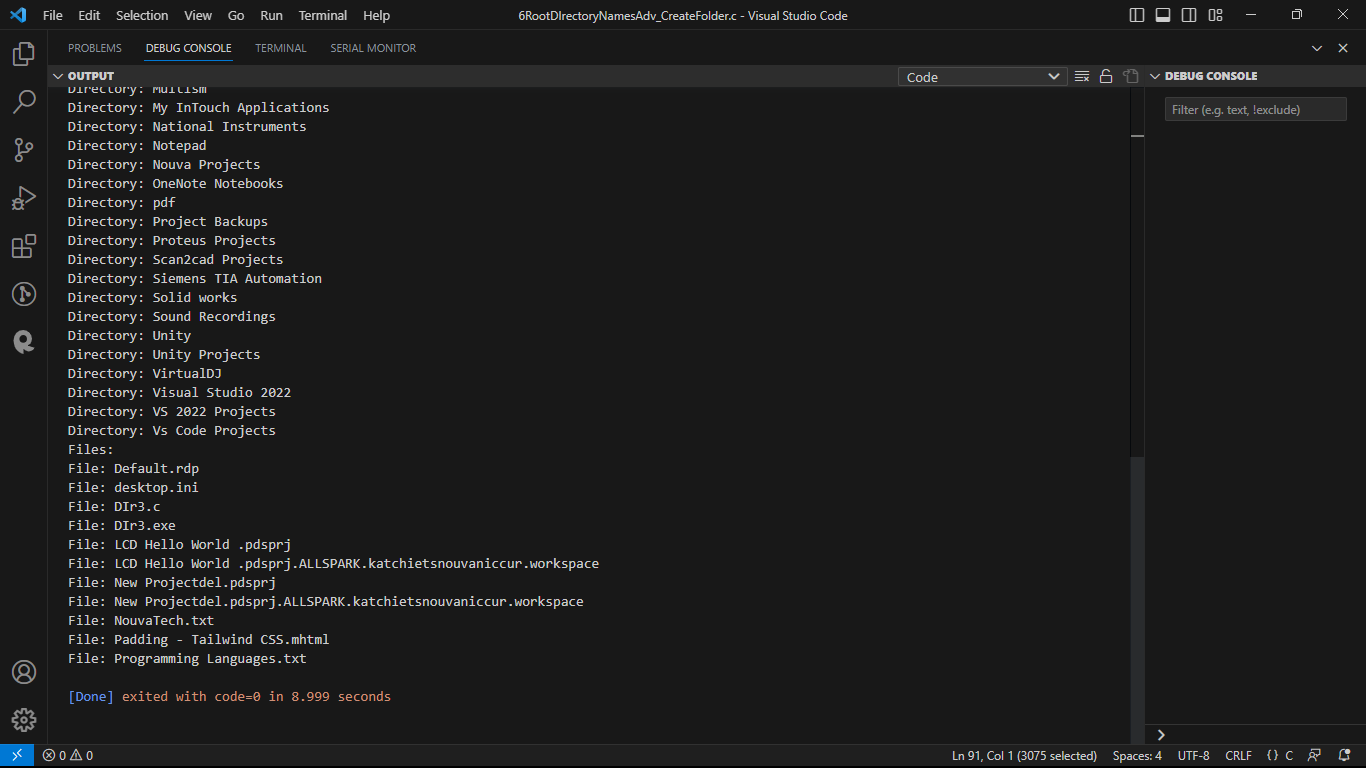
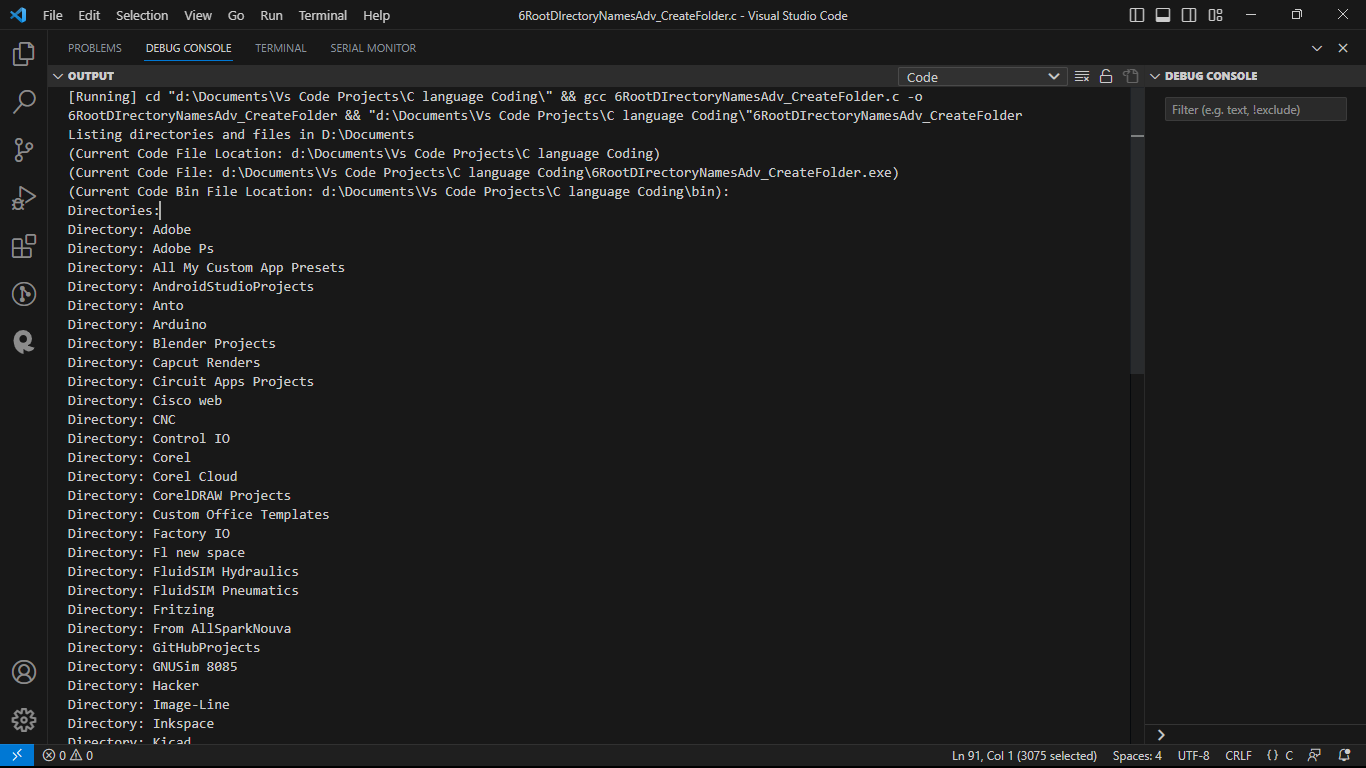
wprintf(L"File: %s**\n**", findFileData.cFileName);

}

} while (FindNextFileW(hFind, &findFileData) != 0);

FindClose(hFind);

}

\*\***The following code didn’t create a bin folder and a 1.txt file inside it. But it ran successfully after multiple tries. It also removed some functionalities**

#include <stdio.h>

#include <windows.h>

#include <string.h>

void listDirectoriesAndFilesInRoot(const wchar\_t \*path);

int main()

{

wchar\_t currentCodeFilePath[FILENAME\_MAX];

GetModuleFileNameW(NULL, currentCodeFilePath, FILENAME\_MAX);

const wchar\_t \*lastBackslash = wcsrchr(currentCodeFilePath, L'**\\**');

currentCodeFilePath[lastBackslash - currentCodeFilePath + 1] = L'**\0**';

wprintf(L"Listing directories and files in D:**\\**Documents**\n**(Current Code File Location: %ls)**\n**", currentCodeFilePath);

listDirectoriesAndFilesInRoot(L"D:**\\**Documents");

return 0;

}

void listDirectoriesAndFilesInRoot(const wchar\_t \*path)

{

WIN32\_FIND\_DATAW findFileData;

HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

if (hFind == INVALID\_HANDLE\_VALUE)

{

wprintf(L"Error finding directories and files in: %ls**\n**", path);

return;

}

*// Collect directories and non-directory files separately*

wprintf(L"Directories:**\n**");

do

{

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)

{

wprintf(L"Directory: %ls**\n**", findFileData.cFileName);

}

}

} while (FindNextFileW(hFind, &findFileData) != 0);

*// Close the handle and reopen it to reset the search*

FindClose(hFind);

hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

*// Collect non-directory files*

wprintf(L"Files:**\n**");

do

{

if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))

{

wprintf(L"File: %ls**\n**", findFileData.cFileName);

}

} while (FindNextFileW(hFind, &findFileData) != 0);

FindClose(hFind);

*// Create a new directory called "bin" in the current code file location*

wchar\_t binDirectory[FILENAME\_MAX];

\_snwprintf(binDirectory, FILENAME\_MAX, L"%ls**\\**bin", path);

CreateDirectoryW(binDirectory, NULL);

*// Create and log to a text file in the "bin" directory*

wchar\_t logFileName[FILENAME\_MAX];

int suffix = 1;

wchar\_t suffixString[10];

\_snwprintf(suffixString, 10, L"%d", suffix);

\_snwprintf(logFileName, FILENAME\_MAX, L"%ls**\\**log\_%ls.txt", path, suffixString);

FILE \*logFile = \_wfopen(logFileName, L"w");

if (logFile)

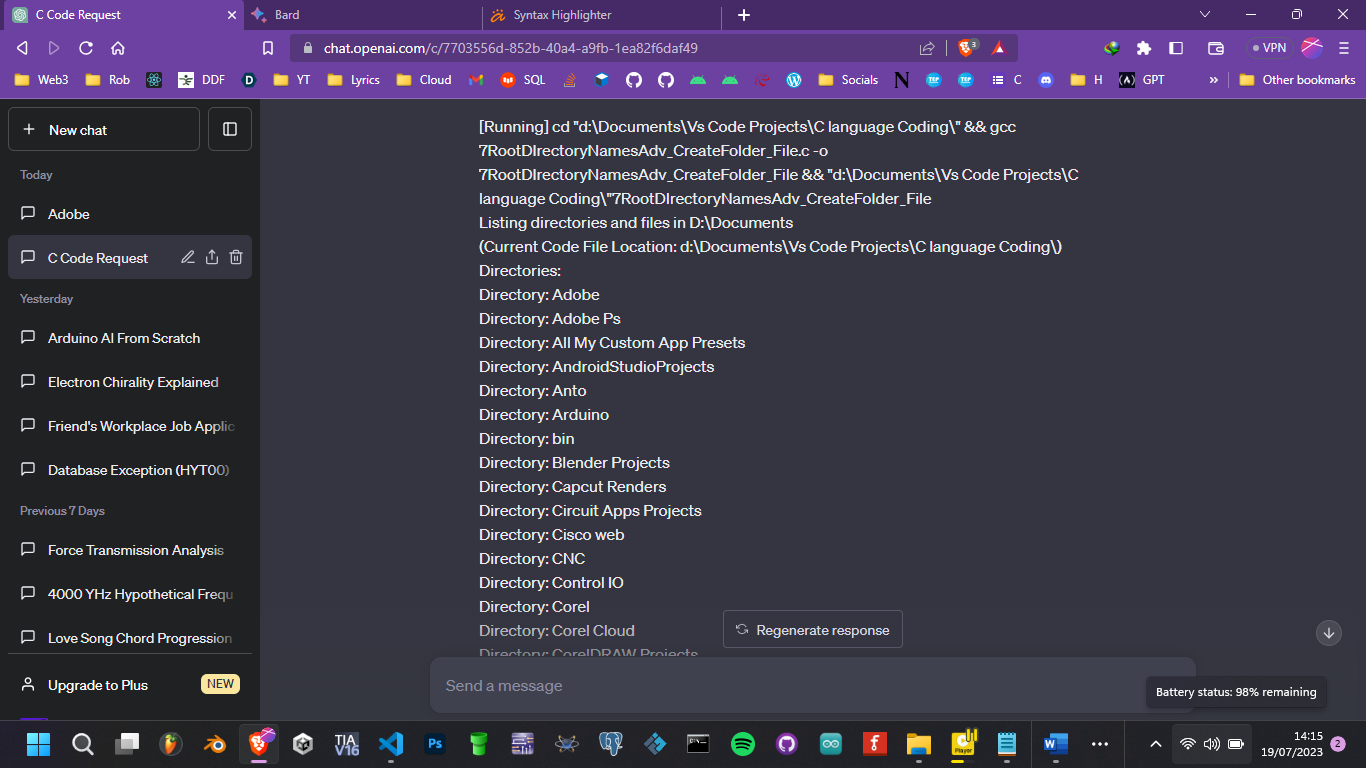
{

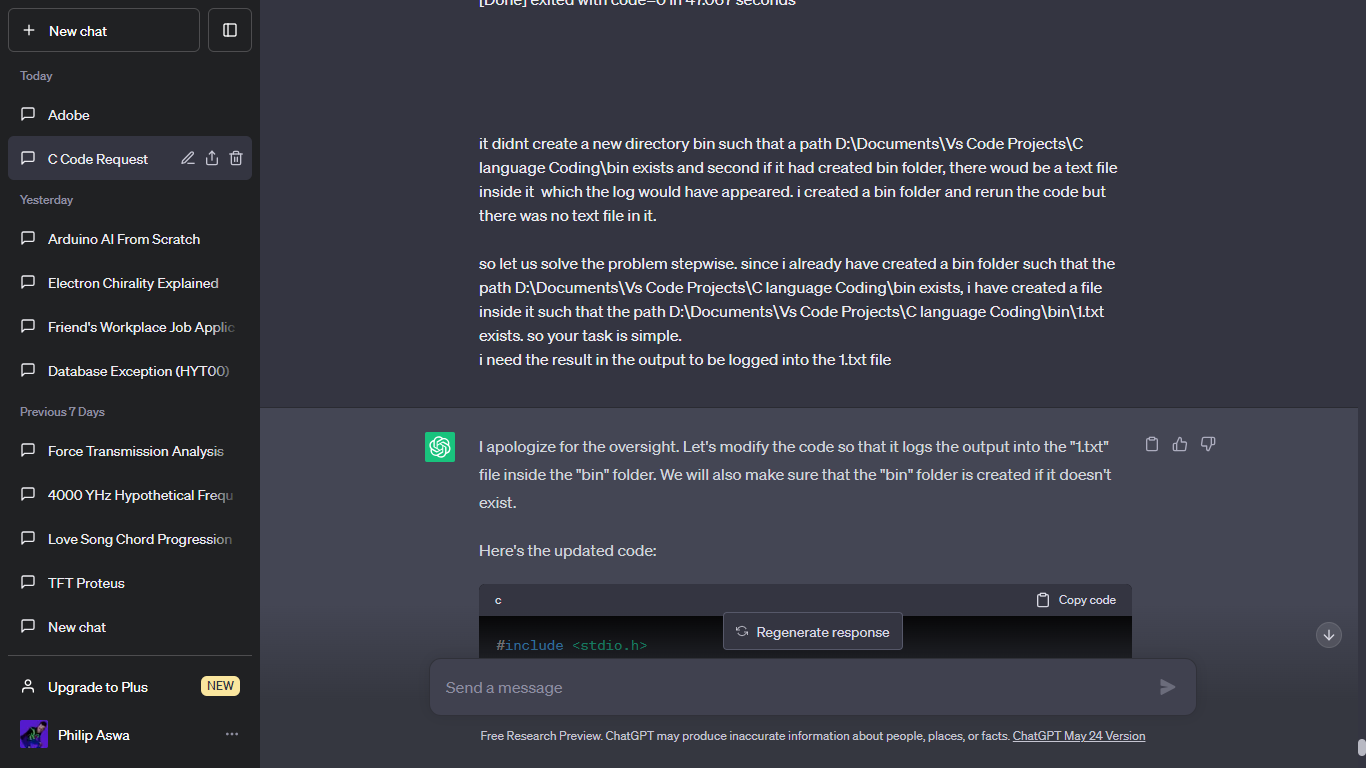
fwprintf(logFile, L"This is a sample log entry.**\n**");

fclose(logFile);

}

}





**The following code was able to write into D:\Documents\bin\1.txt. the text** This is a sample log entry

#include <stdio.h>

#include <windows.h>

#include <string.h>

void listDirectoriesAndFilesInRoot(const wchar\_t \*path);

int main()

{

wchar\_t currentCodeFilePath[FILENAME\_MAX];

GetModuleFileNameW(NULL, currentCodeFilePath, FILENAME\_MAX);

const wchar\_t \*lastBackslash = wcsrchr(currentCodeFilePath, L'**\\**');

currentCodeFilePath[lastBackslash - currentCodeFilePath + 1] = L'**\0**';

wprintf(L"Listing directories and files in D:**\\**Documents**\n**(Current Code File Location: %ls)**\n**", currentCodeFilePath);

listDirectoriesAndFilesInRoot(L"D:**\\**Documents");

return 0;

}

void listDirectoriesAndFilesInRoot(const wchar\_t \*path)

{

WIN32\_FIND\_DATAW findFileData;

HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

if (hFind == INVALID\_HANDLE\_VALUE)

{

wprintf(L"Error finding directories and files in: %ls**\n**", path);

return;

}

*// Collect directories and non-directory files separately*

wprintf(L"Directories:**\n**");

do

{

if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)

{

wprintf(L"Directory: %ls**\n**", findFileData.cFileName);

}

}

} while (FindNextFileW(hFind, &findFileData) != 0);

*// Close the handle and reopen it to reset the search*

FindClose(hFind);

hFind = FindFirstFileW((wchar\_t \*)L"D:**\\**Documents**\\**\*", &findFileData);

*// Collect non-directory files*

wprintf(L"Files:**\n**");

do

{

if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))

{

wprintf(L"File: %ls**\n**", findFileData.cFileName);

}

} while (FindNextFileW(hFind, &findFileData) != 0);

FindClose(hFind);

*// Create a new directory called "bin" in the current code file location*

wchar\_t binDirectory[FILENAME\_MAX];

\_snwprintf(binDirectory, FILENAME\_MAX, L"%ls**\\**bin", path);

CreateDirectoryW(binDirectory, NULL);

*// Create and log to a text file in the "bin" directory*

wchar\_t logFileName[FILENAME\_MAX];

\_snwprintf(logFileName, FILENAME\_MAX, L"%ls**\\**bin**\\**1.txt", path);

FILE \*logFile = \_wfopen(logFileName, L"w");

if (logFile)

{

fwprintf(logFile, L"This is a sample log entry.**\n**");

fclose(logFile);

wprintf(L"Successfully logged to %ls.**\n**", logFileName);

}

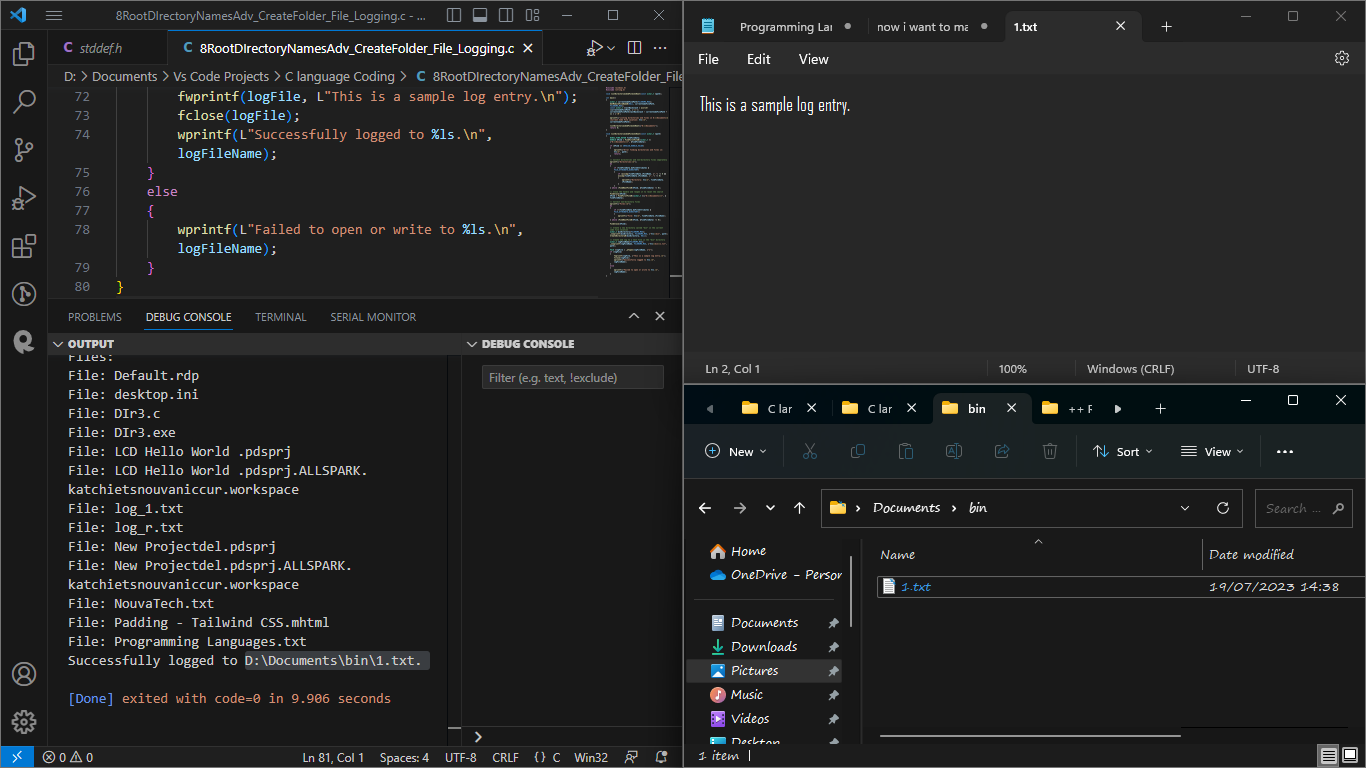
else

{

wprintf(L"Failed to open or write to %ls.**\n**", logFileName);

}

}

****

**The code below is a symbol of advancement. It was able to:**

1. **Create a new folder in the path D:\Documents in case it was non existent**
2. **Create a new file 1.txt incase it was non existent**
3. **Log in data into the text file.**
4. **Incase there was data in the log file it was able to append additional data into the log file.**

#include <stdio.h>

#include <windows.h>

#include <string.h>

void listDirectoriesAndFilesInRoot(const wchar\_t \*path);

int main()

{

    wchar\_t currentCodeFilePath[FILENAME\_MAX];

    GetModuleFileNameW(NULL, currentCodeFilePath, FILENAME\_MAX);

    const wchar\_t \*lastBackslash = wcsrchr(currentCodeFilePath, L'\\');

    currentCodeFilePath[lastBackslash - currentCodeFilePath + 1] = L'\0';

    wprintf(L"Listing directories and files in D:\\Documents\n(Current Code File Location: %ls)\n", currentCodeFilePath);

    listDirectoriesAndFilesInRoot(L"D:\\Documents");

    return 0;

}

void listDirectoriesAndFilesInRoot(const wchar\_t \*path)

{

    WIN32\_FIND\_DATAW findFileData;

    HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:\\Documents\\\*", &findFileData);

    if (hFind == INVALID\_HANDLE\_VALUE)

    {

        wprintf(L"Error finding directories and files in: %ls\n", path);

        return;

    }

    // ... (Code to list directories and files remains unchanged)

    // Create the "bin" directory if it doesn't exist

    wchar\_t binDirectory[FILENAME\_MAX];

    \_snwprintf(binDirectory, FILENAME\_MAX, L"%ls\\bin", path);

    CreateDirectoryW(binDirectory, NULL);

    // Create and log to a text file in the "bin" directory

    wchar\_t logFileName[FILENAME\_MAX];

    \_snwprintf(logFileName, FILENAME\_MAX, L"%ls\\bin\\1.txt", path);

    FILE \*logFile = \_wfopen(logFileName, L"a"); // Open in "append" mode instead of "write" mode

    if (logFile)

    {

        fwprintf(logFile, L"This is a sample log entry.\n");

        fwprintf(logFile, L"Additional log entry 1.\n");

        fwprintf(logFile, L"Additional log entry 2.\n");

        fclose(logFile);

        wprintf(L"Successfully logged to %ls.\n", logFileName);

        // Read and print the contents of the log file to the console

        wprintf(L"\nLogged Contents:\n");

        FILE \*readLogFile = \_wfopen(logFileName, L"r");

        if (readLogFile)

        {

            wchar\_t buffer[512];

            while (fgetws(buffer, 512, readLogFile))

            {

                wprintf(L"%ls", buffer);

            }

            fclose(readLogFile);

        }

        else

        {

            wprintf(L"Failed to read %ls.\n", logFileName);

        }

    }

    else

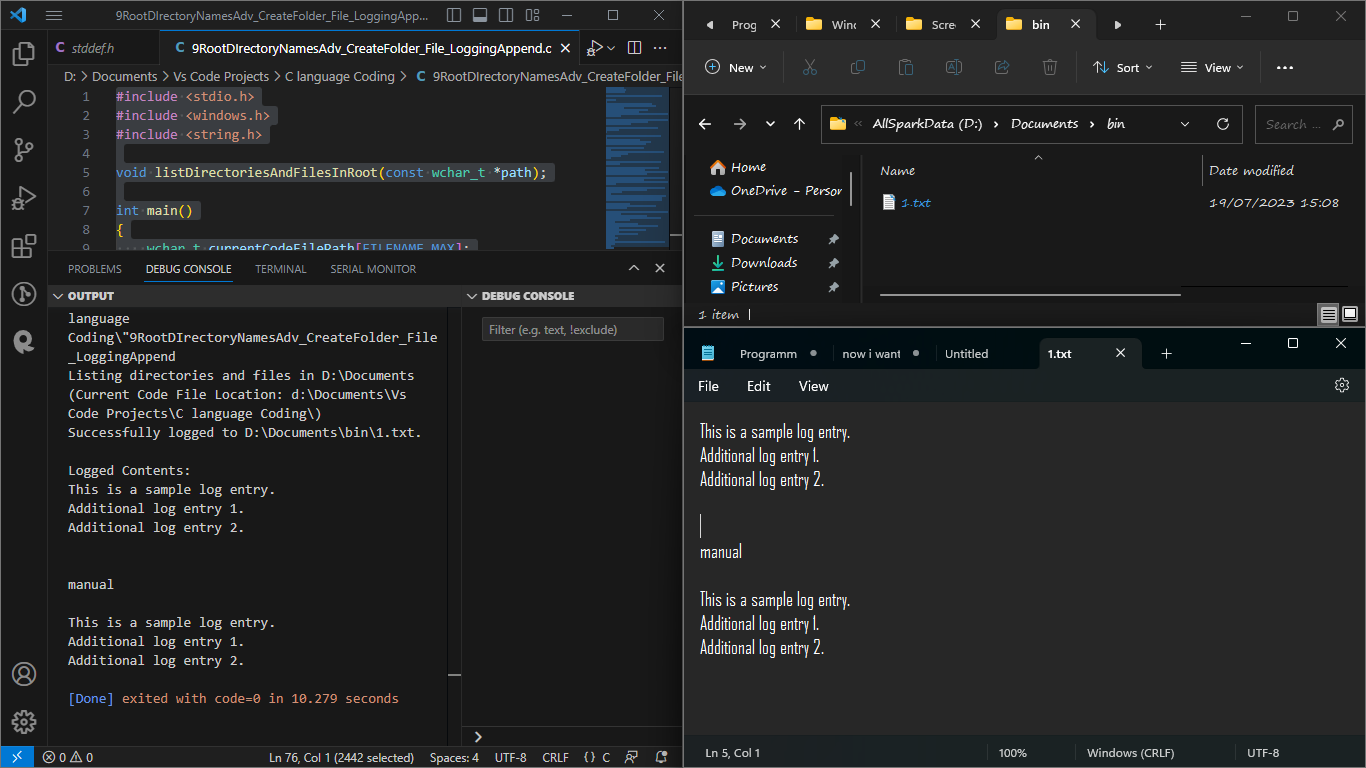
    {

        wprintf(L"Failed to open or write to %ls.\n", logFileName);

    }

    // ... (The rest of the code remains unchanged)

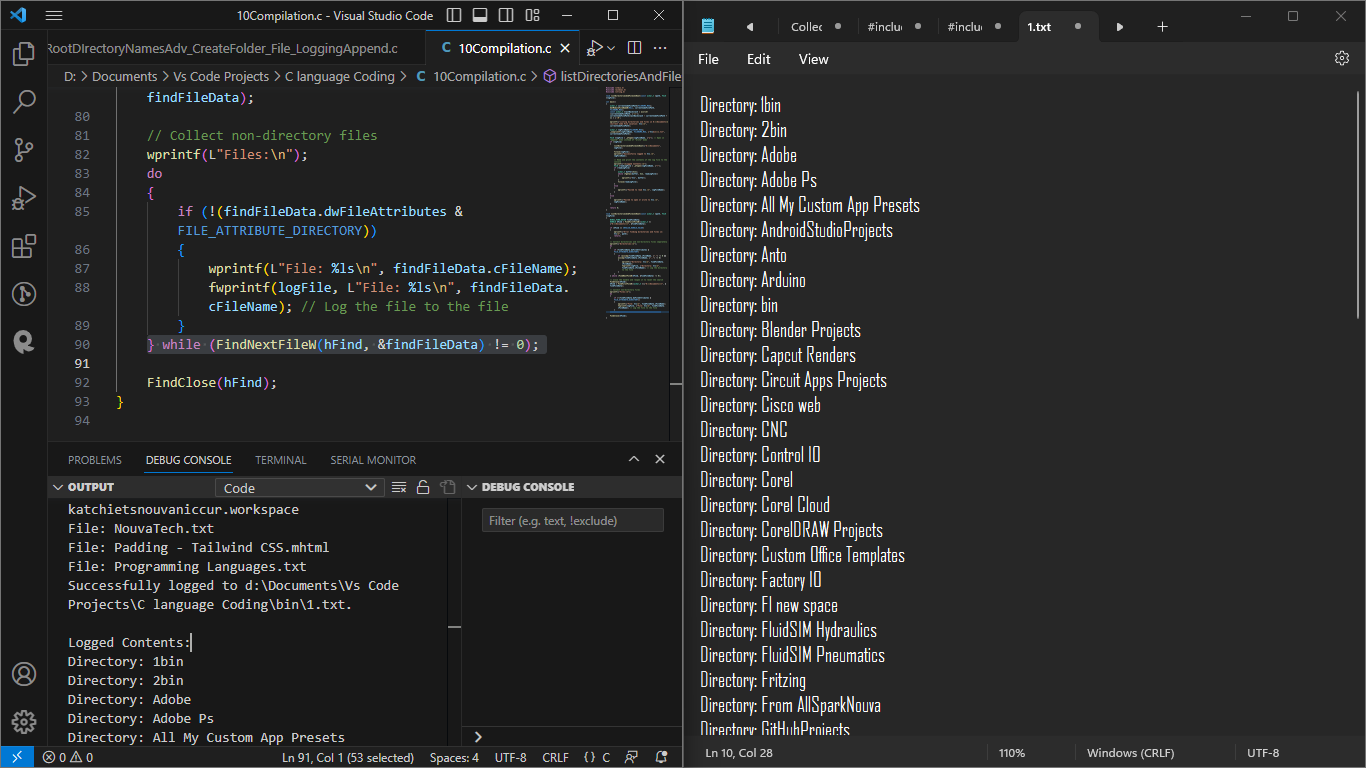
}



## Advanced Desktop Application

**The code below is a symbol of Super advancement. It was able to:**

1. **Check if the bin folder exists in the root of the installation folder**
2. **If not, it will create it before attempting to open the log file (This should prevent the "Failed to open or write to" error from occurring when the folder does not exist.)**
3. **Create a new file 1.txt in case it was nonexistent.**
4. **Write the root contents of path D:\Documents into the log file in installation folder\bin\1.txt**
5. **If the log file already exists, it will append the data to the already existing contents.**
6. **I successfully copied it to another drive and could do step 1 through to 5 without problem**
7. #include <stdio.h>
8. #include <windows.h>
9. #include <string.h>
10. // Function prototype declaration
11. void listDirectoriesAndFilesInRoot(const wchar\_t \*path, FILE \*logFile);
12. int main()
13. {
14. // Variable to store the path of the current code file
15. wchar\_t currentCodeFilePath[FILENAME\_MAX];
16. // Get the path of the current executable (the code file)
17. GetModuleFileNameW(NULL, currentCodeFilePath, FILENAME\_MAX);
18. // Extract the directory path by removing the file name from the path
19. const wchar\_t \*lastBackslash = wcsrchr(currentCodeFilePath, L'\\');
20. currentCodeFilePath[lastBackslash - currentCodeFilePath + 1] = L'\0';
21. // Print the current directory and the code file's location
22. wprintf(L"Listing directories and files in D:\\Documents\n(Current Code File Location: %ls)\n", currentCodeFilePath);
23. // Variable to store the log file name
24. wchar\_t logFileName[FILENAME\_MAX];
25. // Create the log file name using the current directory path
26. \_snwprintf(logFileName, FILENAME\_MAX, L"%lsbin\\1.txt", currentCodeFilePath);
27. // Check if the 'bin' folder exists, if not, create it
28. wchar\_t binFolderPath[FILENAME\_MAX];
29. \_snwprintf(binFolderPath, FILENAME\_MAX, L"%lsbin", currentCodeFilePath);
30. if (!CreateDirectoryW(binFolderPath, NULL))
31. {
32. DWORD error = GetLastError();
33. if (error != ERROR\_ALREADY\_EXISTS)
34. {
35. // Print an error message if the folder creation fails
36. wprintf(L"Failed to create 'bin' folder: %ls\n", binFolderPath);
37. return 1; // Return with an error code (1) to indicate failure
38. }
39. }
40. // Open the log file in "append" mode (add data to the existing file)
41. FILE \*logFile = \_wfopen(logFileName, L"a");
42. if (logFile)
43. {
44. // Call the function to list directories and files in the specified path
45. listDirectoriesAndFilesInRoot(L"D:\\Documents", logFile);
46. // Close the log file after writing the data
47. fclose(logFile);
48. // Print a success message with the log file name
49. wprintf(L"Successfully logged to %ls.\n", logFileName);
50. // Read and print the contents of the log file to the console
51. wprintf(L"\nLogged Contents:\n");
52. FILE \*readLogFile = \_wfopen(logFileName, L"r");
53. if (readLogFile)
54. {
55. wchar\_t buffer[512];
56. while (fgetws(buffer, 512, readLogFile))
57. {
58. wprintf(L"%ls", buffer);
59. }
60. fclose(readLogFile);
61. }
62. else
63. {
64. // Print an error message if reading the log file fails
65. wprintf(L"Failed to read %ls.\n", logFileName);
66. }
67. }
68. else
69. {
70. // Print an error message if opening or writing to the log file fails
71. wprintf(L"Failed to open or write to %ls.\n", logFileName);
72. }
73. return 0; // Return with a success code (0) to indicate successful execution
74. }
75. // Function definition to list directories and files in the specified path
76. void listDirectoriesAndFilesInRoot(const wchar\_t \*path, FILE \*logFile)
77. {
78. // Variables for handling file enumeration in the specified path
79. WIN32\_FIND\_DATAW findFileData;
80. HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:\\Documents\\\*", &findFileData);
81. // Check if file enumeration is successful or not
82. if (hFind == INVALID\_HANDLE\_VALUE)
83. {
84. // Print an error message if enumeration fails
85. wprintf(L"Error finding directories and files in: %ls\n", path);
86. return; // Return from the function
87. }
88. // Collect directories and non-directory files separately
89. wprintf(L"Directories:\n");
90. do
91. {
92. // Check if the current item is a directory
93. if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)
94. {
95. // Exclude '.' and '..' directories from the listing
96. if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)
97. {
98. // Print the directory name to the console
99. wprintf(L"Directory: %ls\n", findFileData.cFileName);
100. // Log the directory name to the log file
101. fwprintf(logFile, L"Directory: %ls\n", findFileData.cFileName);
102. }
103. }
104. } while (FindNextFileW(hFind, &findFileData) != 0);
105. // Close the handle and reopen it to reset the search
106. FindClose(hFind);
107. hFind = FindFirstFileW((wchar\_t \*)L"D:\\Documents\\\*", &findFileData);
108. // Collect non-directory files
109. wprintf(L"Files:\n");
110. do
111. {
112. // Check if the current item is not a directory (i.e., a file)
113. if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))
114. {
115. // Print the file name to the console
116. wprintf(L"File: %ls\n", findFileData.cFileName);
117. // Log the file name to the log file
118. fwprintf(logFile, L"File: %ls\n", findFileData.cFileName);
119. }
120. } while (FindNextFileW(hFind, &findFileData) != 0);
121. // Close the handle to release resources
122. FindClose(hFind);
123. }



To change the path you want to look into (e.g., D:\dukto instead of D:\Documents), you need to modify two parts of the code:

1. In the **main()** function:

1 int main()

1. {
2. // ...
3. // Replace the path "D:\\Documents" with "D:\\dukto"
4. listDirectoriesAndFilesInRoot(L"D:\\dukto", logFile);
5. // ...
6. }

In the **listDirectoriesAndFilesInRoot** function:

1. void listDirectoriesAndFilesInRoot(const wchar\_t \*path, FILE \*logFile)
2. {
3. // ...
4. // Replace the path "D:\\Documents\\\*" with the new path "D:\\dukto\\\*"
5. HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:\\dukto\\\*", &findFileData);
6. // ...
7. }

After making these changes, the program will list directories and files in the specified path (D:\dukto) instead of D:\Documents.

Remember to ensure that the specified path (D:\dukto) exists and contains the files and directories you want to list; otherwise, the program might not produce the expected results.

**I wanted to access the default folders. Now suppose that normally in every computer, by default there is a downloads, document, videos, pictures, music and desktop folders. In most operating systems, especially Windows, macOS, and Linux distributions, there are standard default folders commonly found in the user's home directory. These folders are created by the system to organize specific types of files and are commonly known as "special folders" or "user folders."**

**The standard default folders you mentioned are commonly found in many systems:**

1. **Downloads: This folder is intended to store downloaded files from the internet or other sources.**
2. **Documents: It's used to store user documents and other text-based files.**
3. **Videos: This folder is meant to hold video files.**
4. **Pictures: It's used to store image files and photos.**
5. **Music: This folder is meant to hold audio files, like music tracks.**
6. **Desktop: It's used to store files and shortcuts that are displayed on the user's desktop.**

**The actual names of these folders might vary slightly depending on the operating system and its language settings. For example, on some systems, "Documents" might be called "My Documents" or "Documents" in the native language.**

**These folders are intended to help users organize their files conveniently, and most applications typically save and access user-specific files from these standard folders to promote a consistent user experience.**

**It's important to note that while these default folders are commonly present in most systems, users can still create, rename, or delete folders to suit their preferences or organizational needs. Additionally, the availability and names of these folders can differ in various operating systems and versions.**

To access the standard default folders without using hardcoded paths, you can use platform-specific functions that provide the paths to these special folders programmatically. In Windows, you can use the SHGetFolderPath function to get the paths to special folders. On macOS, you can use NSSearchPathForDirectoriesInDomains, and on Linux, you can use XDG Base Directory Specification.

Here's how you can modify the code to access the "Downloads" folder on Windows using SHGetFolderPath

Added the #include <shlobj.h> header to include the necessary declarations for using the SHGetFolderPath function.

Modified the main() function to use SHGetFolderPath to get the path of the "Downloads" folder:

The SHGetFolderPath function retrieves the path of a special folder identified by a CSIDL (Constant Special Item ID List) value. In this case, CSIDL\_PERSONAL is used, which represents the "My Documents" folder on Windows, where the "Downloads" folder is often located.

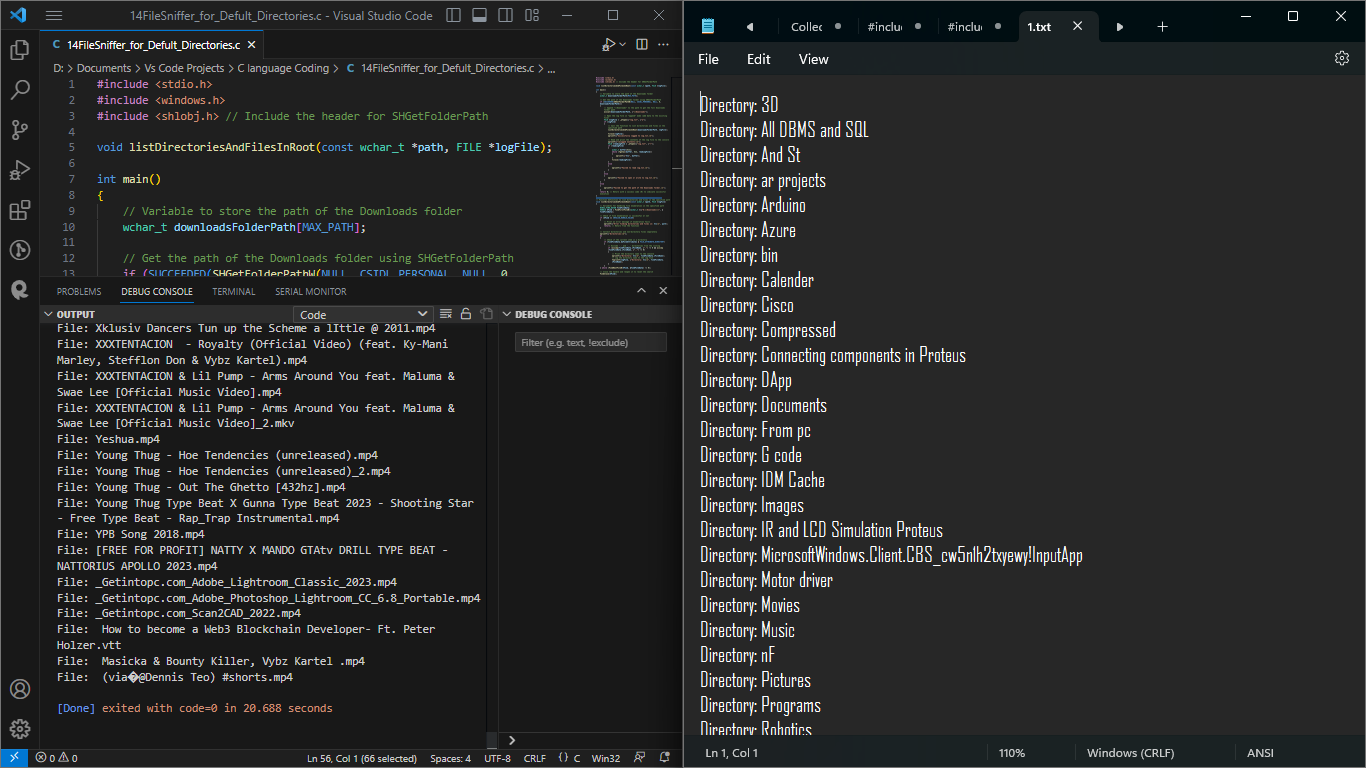
After retrieving the path using SHGetFolderPath, the code appends "\Downloads" to the path to form the full path to the "Downloads" folder. The rest of the code then uses this dynamically obtained path to list directories and files in the "Downloads" folder.

Using this approach allows the code to access the "Downloads" folder without relying on hardcoded paths, making it more flexible and portable across different systems and users.

// ... Your existing function implementation remains unchanged ...

**From code number 57**

1. #include <stdio.h>
2. #include <windows.h>
3. #include <shlobj.h> // Include the header for SHGetFolderPath
4. void listDirectoriesAndFilesInRoot(const wchar\_t \*path, FILE \*logFile);
5. int main()
6. {
7. // Variable to store the path of the Downloads folder
8. wchar\_t downloadsFolderPath[MAX\_PATH];
10. // Get the path of the Downloads folder using SHGetFolderPath
11. if (SUCCEEDED(SHGetFolderPathW(NULL, CSIDL\_PERSONAL, NULL, 0, downloadsFolderPath)))
12. {
13. // Append "\\Downloads" to the path to get the full Downloads folder path
14. wcscat(downloadsFolderPath, L"\\Downloads");
15. // Open the log file in "append" mode (add data to the existing file)
16. FILE \*logFile = \_wfopen(L"log.txt", L"a");
17. if (logFile)
18. {
19. // Call the function to list directories and files in the specified path
20. listDirectoriesAndFilesInRoot(downloadsFolderPath, logFile);
21. fclose(logFile);
22. wprintf(L"Successfully logged to log.txt.\n");
23. // Read and print the contents of the log file to the console
24. wprintf(L"\nLogged Contents:\n");
25. FILE \*readLogFile = \_wfopen(L"log.txt", L"r");
26. if (readLogFile)
27. {
28. wchar\_t buffer[512];
29. while (fgetws(buffer, 512, readLogFile))
30. {
31. wprintf(L"%ls", buffer);
32. }
33. fclose(readLogFile);
34. }
35. else
36. {
37. wprintf(L"Failed to read log.txt.\n");
38. }
39. }
40. else
41. {
42. wprintf(L"Failed to open or write to log.txt.\n");
43. }
44. }
45. else
46. {
47. wprintf(L"Failed to get the path of the Downloads folder.\n");
48. }
49. return 0; // Return with a success code (0) to indicate successful execution
50. }
51. // ... Your existing function implementation remains unchanged ...
52. // Function definition to list directories and files in the specified path
53. void listDirectoriesAndFilesInRoot(const wchar\_t \*path, FILE \*logFile)
54. {
55. // Variables for handling file enumeration in the specified path
56. WIN32\_FIND\_DATAW findFileData;
57. HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:\\Downloads\\\*", &findFileData);
58. // Check if file enumeration is successful or not
59. if (hFind == INVALID\_HANDLE\_VALUE)
60. {
61. // Print an error message if enumeration fails
62. wprintf(L"Error finding directories and files in: %ls\n", path);
63. return; // Return from the function
64. }
65. // Collect directories and non-directory files separately
66. wprintf(L"Directories:\n");
67. do
68. {
69. // Check if the current item is a directory
70. if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)
71. {
72. // Exclude '.' and '..' directories from the listing
73. if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)
74. {
75. // Print the directory name to the console
76. wprintf(L"Directory: %ls\n", findFileData.cFileName);
77. // Log the directory name to the log file
78. fwprintf(logFile, L"Directory: %ls\n", findFileData.cFileName);
79. }
80. }
81. } while (FindNextFileW(hFind, &findFileData) != 0);
82. // Close the handle and reopen it to reset the search
83. FindClose(hFind);
84. hFind = FindFirstFileW((wchar\_t \*)L"D:\\Downloads\\\*", &findFileData);
85. // Collect non-directory files
86. wprintf(L"Files:\n");
87. do
88. {
89. // Check if the current item is not a directory (i.e., a file)
90. if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))
91. {
92. // Print the file name to the console
93. wprintf(L"File: %ls\n", findFileData.cFileName);
94. // Log the file name to the log file
95. fwprintf(logFile, L"File: %ls\n", findFileData.cFileName);
96. }
97. } while (FindNextFileW(hFind, &findFileData) != 0);
98. // Close the handle to release resources
99. FindClose(hFind);
100. }



**The following code is a template of default folders iteration**

#include <stdio.h>

#include <windows.h>

#include <shlobj.h> // Include the header for SHGetFolderPath

void processFolder(const wchar\_t \*folderName);

int main()

{

    // Define an array of folder names

    const wchar\_t \*defaultFolders[] = {

        L"Downloads",

        L"Documents",

        L"Videos",

        L"Pictures",

        L"Music",

        L"Desktop"

    };

    // Iterate through the array and process each folder

    for (int i = 0; i < sizeof(defaultFolders) / sizeof(defaultFolders[0]); i++)

    {

        processFolder(defaultFolders[i]);

    }

    return 0;

}

// Function to process the contents of a folder

void processFolder(const wchar\_t \*folderName)

{

    // Variable to store the path of the folder

    wchar\_t folderPath[MAX\_PATH];

    // Get the path of the specified folder using SHGetFolderPath

    if (SUCCEEDED(SHGetFolderPathW(NULL, CSIDL\_PERSONAL, NULL, 0, folderPath)))

    {

        // Append the folder name to the path to get the full folder path

        wcscat(folderPath, L"\\");

        wcscat(folderPath, folderName);

        // Perform operations on the folder here...

        // For example, you can list directories and files in the folder, or do any other processing.

        // For demonstration purposes, let's just print the folder path to the console

        wprintf(L"Folder: %ls\n", folderPath);

    }

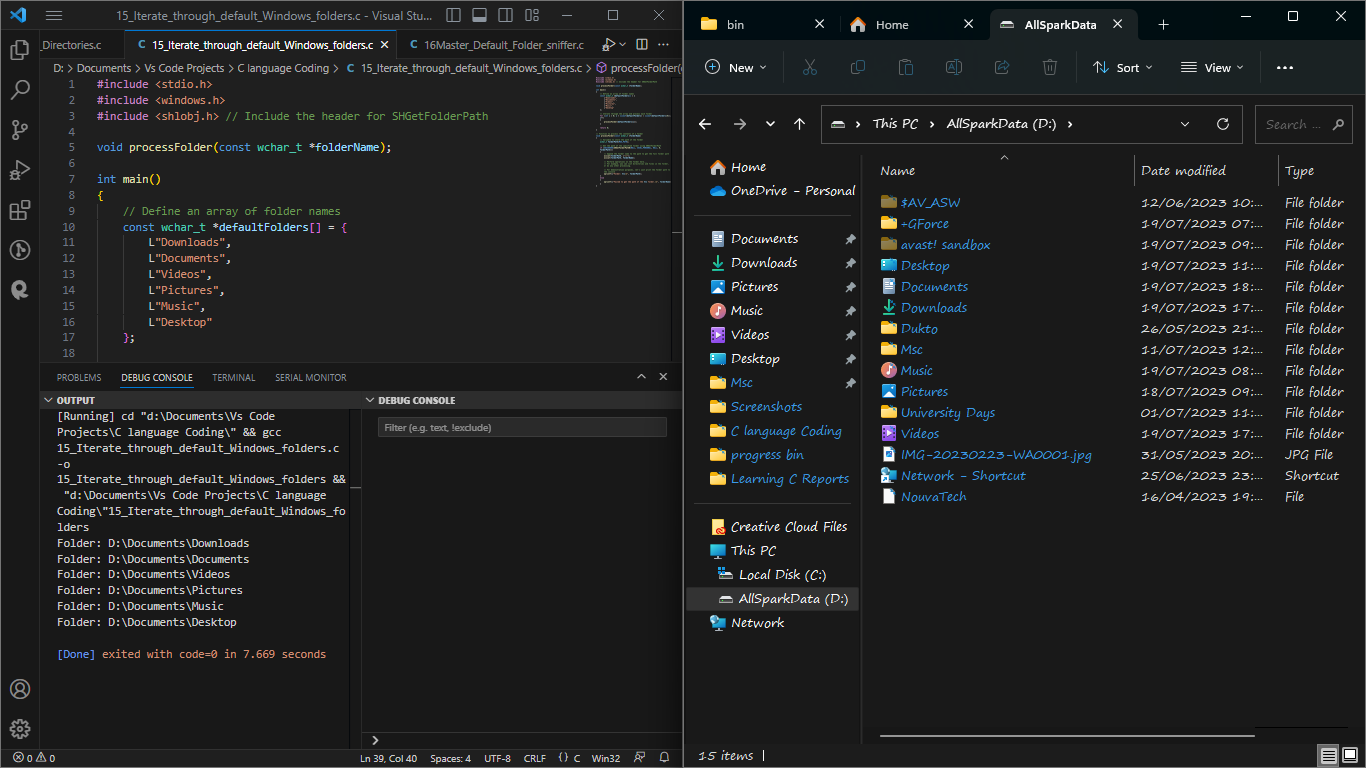
    else

    {

        wprintf(L"Failed to get the path of the %ls folder.\n", folderName);

    }

}



**The above code had the following limitations:**

**The code you provided seems to be a C program that lists directories and files in specific folders on your computer and logs the results to a file. The folders it lists are the default ones like Downloads, Documents, Videos, Pictures, Music, and Desktop. However, there are a few issues in the code that need to be addressed.**

**Hardcoded Path: The code currently lists the directories and files in the "D:\Documents" folder. To make it more flexible and work on any system, you can replace the hardcoded path with a dynamic path using the SHGetFolderPath function.**

**Folder Enumeration: The listDirectoriesAndFilesInRoot function currently uses the "D:\Documents" folder directly for enumeration. Instead, you should use the path parameter that's passed to the function.**

**Incorrect Folder Enumeration: The listDirectoriesAndFilesInRoot function doesn't properly use the provided path for enumeration. It uses a hardcoded path instead. To fix this, you need to replace (wchar\_t \*)L"D:\\Documents\\\*" with (wchar\_t \*)path in both places.**

**Incorrect Log Folder Path: In the processDefaultFolders function, when calling listDirectoriesAndFilesInRoot, it passes the folder names as paths directly. Instead, it should use the dynamic path obtained from SHGetFolderPath for each default folder.**

**With these modifications, the code should now correctly list the directories and files in the default folders and log the results to the specified log file. Additionally, it uses the SHGetFolderPath function to dynamically obtain the path of the "Documents" folder, making the code more adaptable to different systems**

**The listDirectoriesAndFilesInRoot function is not correctly using the provided path for enumeration, and it's always using the "D:\Documents" folder for listing, which leads to the incorrect log.**

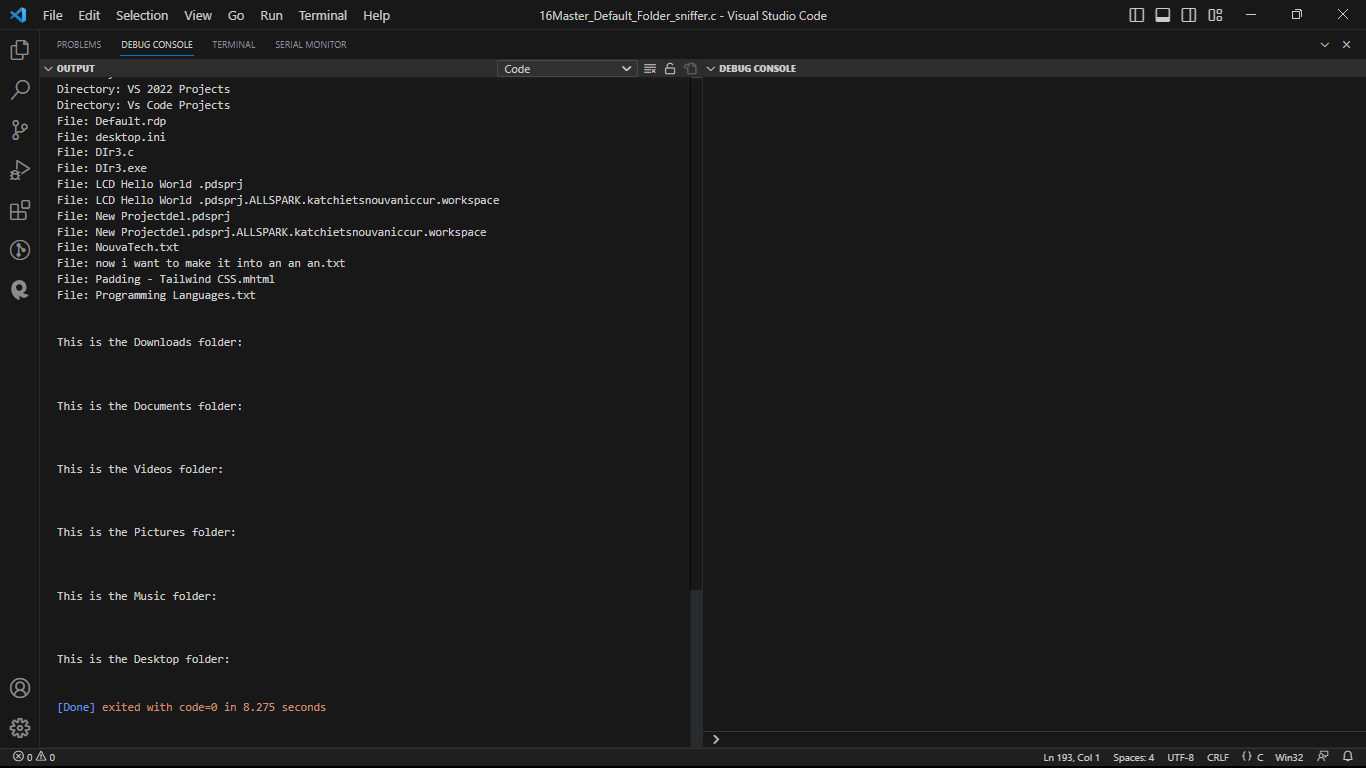
**To fix this, we need to replace the hardcoded path (wchar\_t \*)L"D:\\Documents\\\*" with the path parameter in the listDirectoriesAndFilesInRoot function.**

**still an issue with the code when it comes to listing the contents of the default folders like Downloads, Documents, Videos, etc. The code is not correctly listing the contents of those folders, and it shows "Error finding directories and files" for each of them.**

**The problem lies in the processDefaultFolders function, where we are not correctly passing the paths of the default folders to the listDirectoriesAndFilesInRoot function. We need to provide the correct paths to the listDirectoriesAndFilesInRoot function.**

**To fix this, we need to use the SHGetFolderPath function to obtain the paths of the default folders dynamically**

**With this modification, the processDefaultFolders function should correctly list the contents of each default folder and log them to the file.**



**t there's still an issue with the code when listing the contents of the default folders. The issue lies in the way we are using the SHGetFolderPath function to obtain the paths of the default folders.**

**Instead of using SHGetFolderPath for each default folder, we can directly access the default folders using the corresponding CSIDLs (constants). We don't need to use SHGetFolderPath for these default folders because they have predefined CSIDL values.**

**Let's modify the processDefaultFolders function to directly access the default folders using their CSIDL values:**

**To include the Downloads folder in the output, you can add the CSIDL value for the Downloads folder (CSIDL\_PERSONAL) to the defaultFolderCSIDLs array in the processDefaultFolders function.**

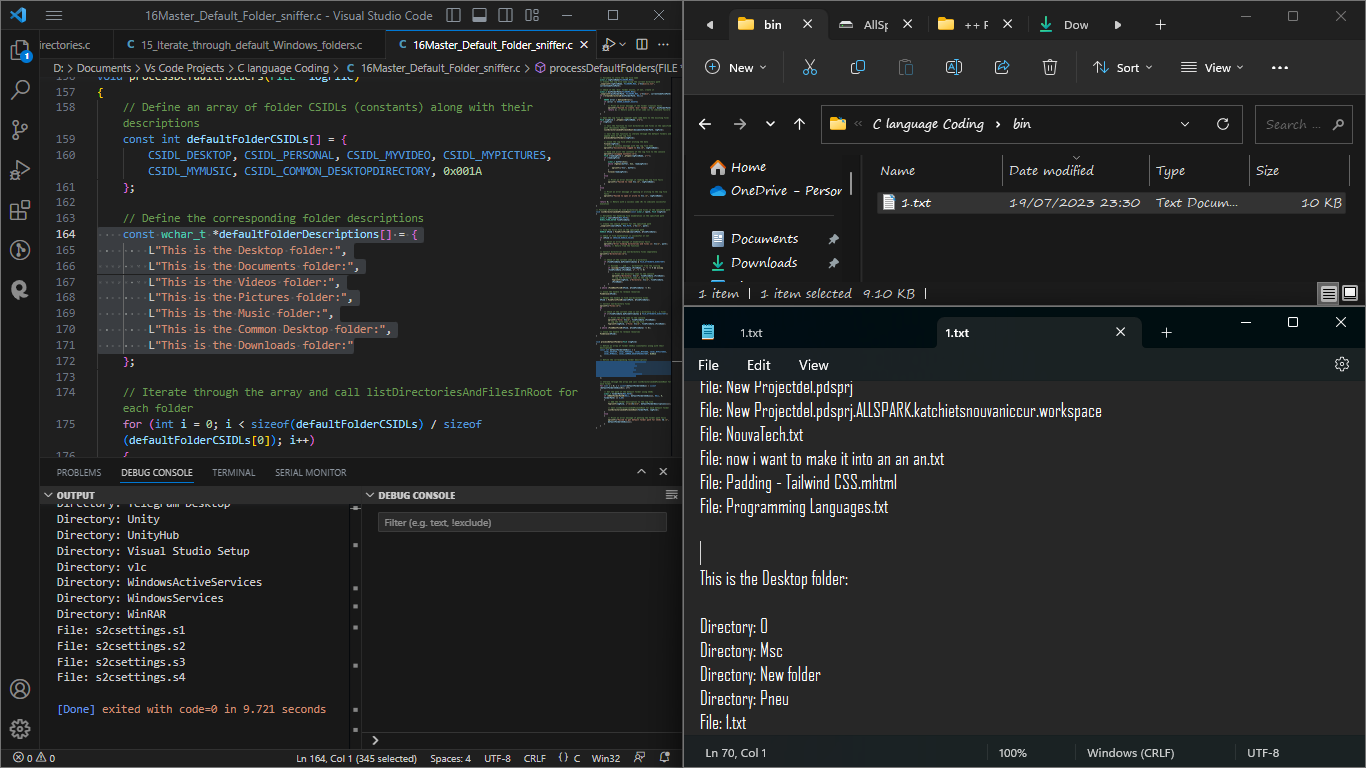
**The following code printed for all the default folders except for the downloads folder:  
It displayed for :**

1. **Desktop folder**
2. **Documents folder**
3. **Videos folder**
4. **Pictures folder**
5. **Music folder**
6. **Common Desktop folder**

**It didn’t display correctly for:**

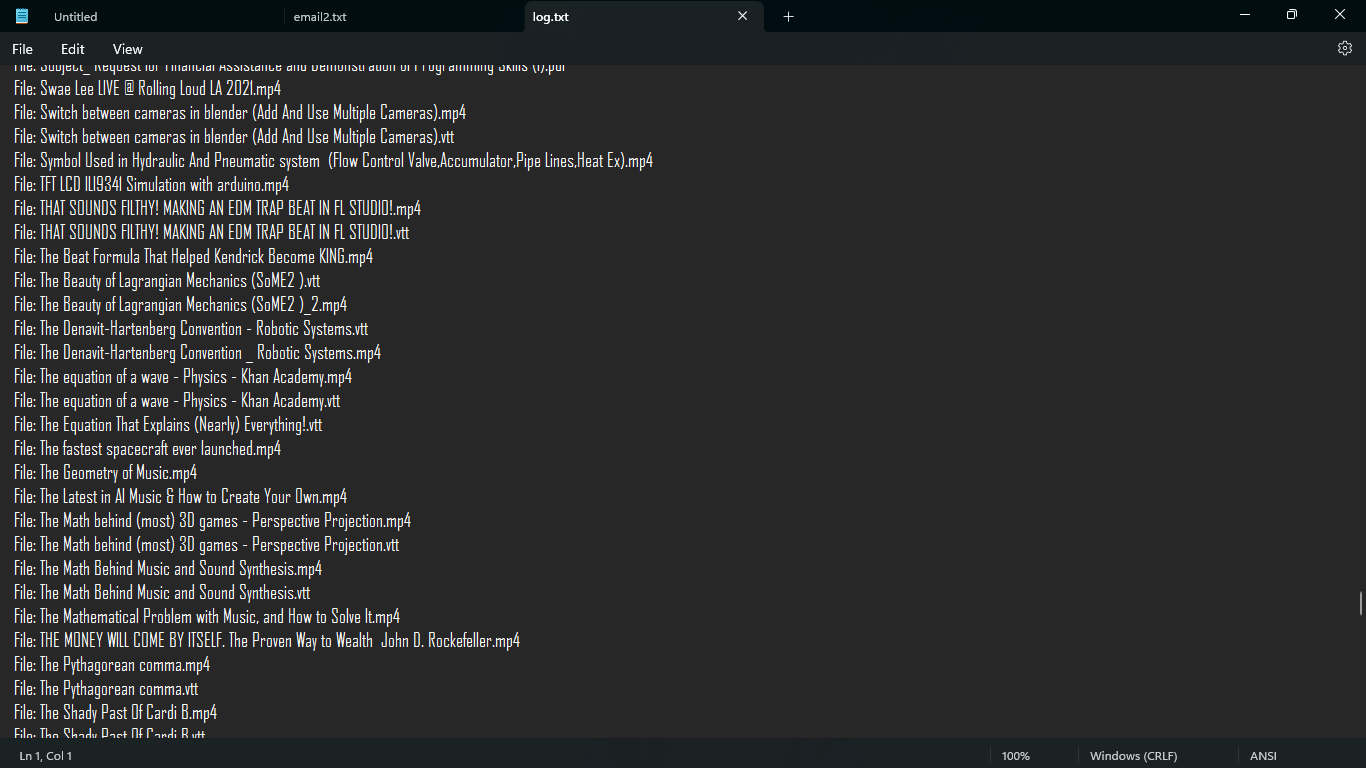
**1. Downloads folder**

1. #include <stdio.h>
2. #include <windows.h>
3. #include <shlobj.h> // Include the header for SHGetFolderPath
4. // Function prototypes
5. void listDirectoriesAndFilesInRoot(const wchar\_t \*path, FILE \*logFile);
6. void processDefaultFolders(FILE \*logFile);
7. int main()
8. {
9. // Variable to store the path of the current code file
10. wchar\_t currentCodeFilePath[FILENAME\_MAX];
11. // Get the path of the current executable (the code file)
12. GetModuleFileNameW(NULL, currentCodeFilePath, FILENAME\_MAX);
13. // Extract the directory path by removing the file name from the path
14. const wchar\_t \*lastBackslash = wcsrchr(currentCodeFilePath, L'\\');
15. currentCodeFilePath[lastBackslash - currentCodeFilePath + 1] = L'\0';
16. // Get the path of the 'Documents' folder using SHGetFolderPath
17. wchar\_t documentsFolderPath[FILENAME\_MAX];
18. if (SHGetFolderPathW(NULL, CSIDL\_PERSONAL, NULL, 0, documentsFolderPath) != S\_OK)
19. {
20. // Print an error message if getting the folder path fails
21. wprintf(L"Failed to get 'Documents' folder path.\n");
22. return 1; // Return with an error code (1) to indicate failure
23. }
24. // Print the current directory and the code file's location
25. wprintf(L"Listing directories and files in %ls\n(Current Code File Location: %ls)\n", documentsFolderPath, currentCodeFilePath);
26. // Variable to store the log file name
27. wchar\_t logFileName[FILENAME\_MAX];
28. // Create the log file name using the current directory path
29. \_snwprintf(logFileName, FILENAME\_MAX, L"%lsbin\\1.txt", currentCodeFilePath);
30. // Check if the 'bin' folder exists, if not, create it
31. wchar\_t binFolderPath[FILENAME\_MAX];
32. \_snwprintf(binFolderPath, FILENAME\_MAX, L"%lsbin", currentCodeFilePath);
33. if (!CreateDirectoryW(binFolderPath, NULL))
34. {
35. DWORD error = GetLastError();
36. if (error != ERROR\_ALREADY\_EXISTS)
37. {
38. // Print an error message if the folder creation fails
39. wprintf(L"Failed to create 'bin' folder: %ls\n", binFolderPath);
40. return 1; // Return with an error code (1) to indicate failure
41. }
42. }
43. // Open the log file in "append" mode (add data to the existing file)
44. FILE \*logFile = \_wfopen(logFileName, L"a");
45. if (logFile)
46. {
47. // Call the function to list directories and files in the specified path (Documents folder)
48. listDirectoriesAndFilesInRoot(documentsFolderPath, logFile);
49. // Call the new function to iterate through the default folders and append data to the log file
50. processDefaultFolders(logFile);
51. // Close the log file after writing the data
52. fclose(logFile);
53. // Print a success message with the log file name
54. wprintf(L"Successfully logged to %ls.\n", logFileName);
55. // Read and print the contents of the log file to the console
56. wprintf(L"\nLogged Contents:\n");
57. FILE \*readLogFile = \_wfopen(logFileName, L"r");
58. if (readLogFile)
59. {
60. wchar\_t buffer[512];
61. while (fgetws(buffer, 512, readLogFile))
62. {
63. wprintf(L"%ls", buffer);
64. }
65. fclose(readLogFile);
66. }
67. else
68. {
69. // Print an error message if reading the log file fails
70. wprintf(L"Failed to read %ls.\n", logFileName);
71. }
72. }
73. else
74. {
75. // Print an error message if opening or writing to the log file fails
76. wprintf(L"Failed to open or write to %ls.\n", logFileName);
77. }
78. return 0; // Return with a success code (0) to indicate successful execution
79. }
80. // Function definition to list directories and files in the specified path
81. void listDirectoriesAndFilesInRoot(const wchar\_t \*path, FILE \*logFile)
82. {
83. // Variables for handling file enumeration in the specified path
84. wchar\_t searchPath[MAX\_PATH];
85. WIN32\_FIND\_DATAW findFileData;
86. // Create the search pattern for the specified path
87. \_snwprintf(searchPath, MAX\_PATH, L"%s\\\*", path);
88. // Find the first file in the specified path
89. HANDLE hFind = FindFirstFileW(searchPath, &findFileData);
90. // Check if file enumeration is successful or not
91. if (hFind == INVALID\_HANDLE\_VALUE)
92. {
93. // Print an error message if enumeration fails
94. wprintf(L"Error finding directories and files in: %ls\n", path);
95. return; // Return from the function
96. }
97. // Collect directories and non-directory files separately
98. wprintf(L"Directories:\n");
99. do
100. {
101. // Check if the current item is a directory
102. if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)
103. {
104. // Exclude '.' and '..' directories from the listing
105. if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)
106. {
107. // Print the directory name to the console
108. wprintf(L"Directory: %ls\n", findFileData.cFileName);
109. // Log the directory name to the log file
110. fwprintf(logFile, L"Directory: %ls\n", findFileData.cFileName);
111. }
112. }
113. } while (FindNextFileW(hFind, &findFileData) != 0);
114. // Close the handle to release resources
115. FindClose(hFind);
116. // Reopen the handle to find non-directory files
117. hFind = FindFirstFileW(searchPath, &findFileData);
118. // Collect non-directory files
119. wprintf(L"Files:\n");
120. do
121. {
122. // Check if the current item is not a directory (i.e., a file)
123. if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))
124. {
125. // Print the file name to the console
126. wprintf(L"File: %ls\n", findFileData.cFileName);
127. // Log the file name to the log file
128. fwprintf(logFile, L"File: %ls\n", findFileData.cFileName);
129. }
130. } while (FindNextFileW(hFind, &findFileData) != 0);
131. // Close the handle to release resources
132. FindClose(hFind);
133. }
134. void processDefaultFolders(FILE \*logFile)
135. {
136. // Define an array of folder CSIDLs (constants) along with their descriptions
137. const int defaultFolderCSIDLs[] = {
138. CSIDL\_DESKTOP, CSIDL\_PERSONAL, CSIDL\_MYVIDEO, CSIDL\_MYPICTURES, CSIDL\_MYMUSIC, CSIDL\_COMMON\_DESKTOPDIRECTORY, 0x001A
139. };
140. // Define the corresponding folder descriptions
141. const wchar\_t \*defaultFolderDescriptions[] = {
142. L"This is the Desktop folder:",
143. L"This is the Documents folder:",
144. L"This is the Videos folder:",
145. L"This is the Pictures folder:",
146. L"This is the Music folder:",
147. L"This is the Common Desktop folder:",
148. L"This is the Downloads folder:"
149. };
150. // Iterate through the array and call listDirectoriesAndFilesInRoot for each folder
151. for (int i = 0; i < sizeof(defaultFolderCSIDLs) / sizeof(defaultFolderCSIDLs[0]); i++)
152. {
153. // Get the path of the default folder using CSIDL
154. wchar\_t folderPath[MAX\_PATH];
155. if (SHGetFolderPathW(NULL, defaultFolderCSIDLs[i], NULL, 0, folderPath) == S\_OK)
156. {
157. // Add the folder description to the log file
158. fwprintf(logFile, L"\n\n%s\n\n", defaultFolderDescriptions[i]);
159. // Call listDirectoriesAndFilesInRoot for each default folder
160. listDirectoriesAndFilesInRoot(folderPath, logFile);
161. }
162. else
163. {
164. // Print an error message if getting the folder path fails
165. wprintf(L"Failed to get default folder path for CSIDL %d.\n", defaultFolderCSIDLs[i]);
166. }
167. }
168. }



**To be able to get and lo the contents the download folder, I had to hard Code the path D:\Downloads**

1. #include <stdio.h>
2. #include <windows.h>
3. #include <string.h>
4. void listDirectoriesAndFilesInRoot(const wchar\_t \*path, FILE \*logFile);
5. int main()
6. {
7. wchar\_t currentCodeFilePath[FILENAME\_MAX];
8. GetModuleFileNameW(NULL, currentCodeFilePath, FILENAME\_MAX);
9. const wchar\_t \*lastBackslash = wcsrchr(currentCodeFilePath, L'\\');
10. currentCodeFilePath[lastBackslash - currentCodeFilePath + 1] = L'\0';
11. wprintf(L"Listing directories and files in D:\\Documents\n(Current Code File Location: %ls)\n", currentCodeFilePath);
12. wchar\_t logFileName[FILENAME\_MAX];
13. \_snwprintf(logFileName, FILENAME\_MAX, L"%lsbin\\1.txt", currentCodeFilePath);
14. // Check if the 'bin' folder exists, if not, create it
15. wchar\_t binFolderPath[FILENAME\_MAX];
16. \_snwprintf(binFolderPath, FILENAME\_MAX, L"%lsbin", currentCodeFilePath);
17. if (!CreateDirectoryW(binFolderPath, NULL))
18. {
19. DWORD error = GetLastError();
20. if (error != ERROR\_ALREADY\_EXISTS)
21. {
22. wprintf(L"Failed to create 'bin' folder: %ls\n", binFolderPath);
23. return 1;
24. }
25. }
26. FILE \*logFile = \_wfopen(logFileName, L"a"); // Open in "append" mode instead of "write" mode
27. if (logFile)
28. {
29. listDirectoriesAndFilesInRoot(L"D:\\Downloads", logFile);
30. fclose(logFile);
31. wprintf(L"Successfully logged to %ls.\n", logFileName);
32. // Read and print the contents of the log file to the console
33. wprintf(L"\nLogged Contents:\n");
34. FILE \*readLogFile = \_wfopen(logFileName, L"r");
35. if (readLogFile)
36. {
37. wchar\_t buffer[512];
38. while (fgetws(buffer, 512, readLogFile))
39. {
40. wprintf(L"%ls", buffer);
41. }
42. fclose(readLogFile);
43. }
44. else
45. {
46. wprintf(L"Failed to read %ls.\n", logFileName);
47. }
48. }
49. else
50. {
51. wprintf(L"Failed to open or write to %ls.\n", logFileName);
52. }
53. return 0;
54. }
55. void listDirectoriesAndFilesInRoot(const wchar\_t \*path, FILE \*logFile)
56. {
57. WIN32\_FIND\_DATAW findFileData;
58. HANDLE hFind = FindFirstFileW((wchar\_t \*)L"D:\\Downloads\\\*", &findFileData);
59. if (hFind == INVALID\_HANDLE\_VALUE)
60. {
61. wprintf(L"Error finding directories and files in: %ls\n", path);
62. return;
63. }
64. // Collect directories and non-directory files separately
65. wprintf(L"Directories:\n");
66. do
67. {
68. if (findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)
69. {
70. if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0)
71. {
72. wprintf(L"Directory: %ls\n", findFileData.cFileName);
73. fwprintf(logFile, L"Directory: %ls\n", findFileData.cFileName); // Log the directory to the file
74. }
75. }
76. } while (FindNextFileW(hFind, &findFileData) != 0);
77. // Close the handle and reopen it to reset the search
78. FindClose(hFind);
79. hFind = FindFirstFileW((wchar\_t \*)L"D:\\Downloads\\\*", &findFileData);
80. // Collect non-directory files
81. wprintf(L"Files:\n");
82. do
83. {
84. if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))
85. {
86. wprintf(L"File: %ls\n", findFileData.cFileName);
87. fwprintf(logFile, L"File: %ls\n", findFileData.cFileName); // Log the file to the file
88. }
89. } while (FindNextFileW(hFind, &findFileData) != 0);
90. FindClose(hFind);
91. }

  
f