**Assignment 1**

Author:

|  |  |
| --- | --- |
| **Abhirup Ranjan** | **110091866** |

Submitted To:

|  |
| --- |
| **Professor** |
| **Dr. Prashanth Ranga** |

**Table of Contents**

[1. Code: 1](#_Toc137206197)

# **Code**

Execution Syntax: **gcc -o ncpmvdir ncpmvdir.c**

Execution Syntax: **./ncpmvdir Folder1 dest -cp txt pdf pptx**

**ncpmvdir [ source\_dir] [destination\_dir] [options] <extension list>**

1. //@Author: Abhirup Ranjan(110091866)
2. // Section: 3
3. // COMP8567
4. // Took help from Google, StackOverflow and other technical websites for designing the code.
5. //It gives the compiler instructions to include definitions for a few extra functions that are included in the X/Open and POSIX standards.
6. #define \_XOPEN\_SOURCE 500
7. #include <stdio.h>
8. #include <string.h>
9. #include <ftw.h>
10. #include <errno.h>
11. #include <unistd.h>
12. #include <stdlib.h>
13. char fileExt[5][5]; // Allowing max 6 extensions with each length upto 6 character
14. char sourcePath[500];// Maximum 500 length character word file can be made
15. char destPath[500];// Maximum 500 lenght character word file can be made
16. // Declaring three varibales been used latter
17. int insideFolder = 0;
18. int countExt = 0;
19. // Check for the extension of a file
20. int validateExtension(const char \*sourcePath)
21. {
22. int i;
23. // "strrchr" FUNCTION is used to find the last occurrence of the period character ('.') in the path string.
24. // If char is not found, it returns NULL or a pointer to the char that was found.
25. char \*a = strrchr(sourcePath, '.');
26. if (a == NULL)
27. return 0;
28. for (i=0; i<countExt; i++)
29. {
30. if ((strcmp(a+1, fileExt[i]) == 0))
31. return 1;
32. }
33. return 0;
34. }
35. int copyContent(const char \*path, const struct stat \*st, int type, struct FTW \*ftwbuffer)
36. {
37. char temp\_path[500];
38. // Formats a string by concatenating path, followed by a substring of path starting from the position
39. // after the src string, and stores the result in the temp\_path string.
40. sprintf(temp\_path, "%s%s", destPath, path + strlen(sourcePath));
41. int rmv;
42. // FTW\_D represents a directory type.
43. if (type == FTW\_D) // IF it would be a directory then this part would be executed
44. {
45. if (insideFolder == 0) //Jump from the root folder
46. {
47. insideFolder++;
48. }
49. else //handling if the directory doesnot exits
50. {
51. rmv = mkdir(temp\_path, 0777);
52. if (rmv == -1 && errno != EEXIST)
53. printf("Some error in function name: copyContent where if (rmv == -1 && errno != EEXIST)");
54. }
55. }
56. // FTW\_F represents regular file type.
57. if (type == FTW\_F)
58. // IF this is statisfied COPY all files
59. if (countExt == 0)
60. {
61. // Link function creates a new hard link between the files specified by path and temp\_path.
62. rmv = link(path, temp\_path);
63. if (rmv == -1)
64. printf("Some error in function name: copyContent where if (countExt == 0)");
65. }
66. else
67. {
68. // Only copyies file for where extension is not matched
69. if (!validateExtension(path))
70. {
71. rmv = link(path, temp\_path);
72. if (rmv == -1)
73. printf("Some error in function name: copyContent where if (!validateExtension(path))");
74. }
75. }
76. return 0;
77. }
78. // Create a Traget folder if doesnot exits
79. int creatFolder(const char \*destPath)
80. {
81. struct stat info; // used for retrieving information about files and directories.
82. // st\_mode stores file type & permissions.
83. // S\_ISDIR returns true if given mode represents a directory.
84. if (!(stat(destPath, &info) == 0 && S\_ISDIR(info.st\_mode)))
85. {
86. int status = mkdir(destPath, 0777);
87. if (status == 0)
88. return 1;
89. else
90. return 0;
91. }
92. return 1;
93. }
94. // Function to copy files and directories
95. int copyDirectory(const char \*sourcePath, const char \*destPath)
96. {
97. // 5 represents the maximum number of file descriptors that the nftw function can open simultaneously.
98. // FTW\_PHYS physical walk of the file system
99. return nftw(sourcePath, copyContent, 5, FTW\_PHYS);
100. }
101. // Function to move files and directories
102. int moveDirectory(const char \*sourcePath, const char \*destPath)
103. {
104. int varMov1;
105. varMov1 = copyDirectory(sourcePath, destPath); // Copy function call
106. if (varMov1 == -1)
107. return varMov1;
108. // REMOVING THE FILES AND DIRECTORY AFTER MOVING
109. // Invokes nftw function to recursively traverse directory tree starting from the directory specified by src.
110. // The remove function will be called on each file or directory encountered during the traversal,
111. // and the traversal will be performed in a depth-first manner while
112. // treating symbolic links as regular files or directories.
113. varMov1 = nftw(sourcePath, remove, 5, FTW\_DEPTH | FTW\_PHYS);
114. if (varMov1 == -1)
115. printf("Some error in function name: moveDirectory where if (varMov1 == -1)");
116. return varMov1;
117. }
118. //MAIN METHOD STARTS HERE
119. int main(int argCount, char \*argVar[])
120. {
121. if (argCount < 4)
122. {
123. // This will instruct the user with correct command which is required to be entered by user
124. printf("Use SYNTEXT AS BELOW:\n%s Source\_DirPath Destination\_DirPath -cp or -mv {extensions which are to be excluded}\n", argVar[0]);
125. return 1;
126. }
127. struct stat st;
128. strcpy(sourcePath, argVar[1]);
129. strcpy(destPath, argVar[2]);
130. // Storing source and target directories paths in another variable
131. // strcpy copies the contents of one string to another for this case the value from array is stored to these strings src & target
132. // Error pops when source path is not found
133. if (!(stat(sourcePath, &st) == 0 && S\_ISDIR(st.st\_mode)))
134. {
135. printf("Use SYNTEXT AS BELOW:\n%s Source\_DirPath Destination\_DirPath -cp or -mv {extensions which are to be excluded}\nAlso Make sure that source Directory should exits in the path epecified!!\n", argVar[0]);
136. return 1;
137. }
138. // This LOGIC WILL CREATE THE FOLDER IN CASE DOESNOT EXITS
139. creatFolder(destPath);
140. // Get the desired extensions
141. if (argCount > 4)
142. {
143. // UPTO 6 EXTENSION CAN BE PROVIDED AS A LIST
144. for (int i=4; i<argCount && i-4 < 6; i++)
145. {
146. strcpy(fileExt[i-4], argVar[i]);
147. countExt++;
148. }
149. }
150. // CHECK IF THE ACTION REQUIRED IS COPY OR MOVE
151. // strcmp function used to compare 2 strings.
152. if (strcmp(argVar[3], "-cp") == 0)
153. return copyDirectory(sourcePath, destPath);
154. if (strcmp(argVar[3], "-mv") == 0)
155. return moveDirectory(sourcePath, destPath);
156. else
157. {
158. // IF IN CASE NEITHER -CP NOR -MV IS PASSED BY USER HENCE HANDLE EXECPTION HERE
159. printf("Use SYNTEXT AS BELOW:\n%s Source\_DirPath Destination\_DirPath -cp or -mv {extensions which are to be excluded}\nEither use command -cp for copy or -mv for move other inputs are not accepted!!\n", argVar[0]);
160. return 1;
161. }
162. }