Advanced System Programming: COMP COM8567 | Section 3

300 Ouellette Ave

Windsor, Ontario, Canada N9A 7B4

Assignment 1

Author:

Abhirup Ranjan	**110091866**

Submitted To:

Professor
Dr. Prashanth Ranga

Table of Contents

1. 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>

```
//@Author: Abhirup Ranjan(110091866)
// Section: 3
// COMP8567

// Took help from Google, StackOverflow and other technical websites for designing the code.

//It gives the compiler instructions to include definitions for a few extra functions that are included in the X/Open and POSIX standards.
#define _XOPEN_SOURCE 500

  #include <stdio.h>
  #include <string.h>
  #include <ftw.h>
  #include <errno.h>
  #include <unistd.h>
  #include <stdlib.h>

  char fileExt[5][5]; // Allowing max 6 extensions with each length upto 6 character
  char sourcePath[500];// Maximum 500 length character word file can be made
  char destPath[500];// Maximum 500 lenght character word file can be made

  // Declaring three varibales been used latter
  int insideFolder = 0;
  int countExt = 0;

  // Check for the extension of a file
  int validateExtension(const char *sourcePath)
  {
      int i;
      // "strrchr" FUNCTION is used to find the last occurrence of the period character ('.') in the path string.
      // If char is not found, it returns NULL or a pointer to the char that was found.
      char *a = strrchr(sourcePath, '.');
      if (a == NULL)
```

```c
            return 0;
        for (i=0; i<countExt; i++)
        {
            if ((strcmp(a+1, fileExt[i]) == 0))
                return 1;
        }
        return 0;
    }

    int copyContent(const char *path, const struct stat *st, int type, struct FTW *ftwbuffer)
    {
        char temp_path[500];
        // Formats a string by concatenating path, followed by a substring of path starting from the position
        // after the src string, and stores the result in the temp_path string.
        sprintf(temp_path, "%s%s", destPath, path + strlen(sourcePath));
        int rmv;
        // FTW_D represents a directory type.
        if (type == FTW_D) // IF it would be a directory then this part would be executed
        {
            if (insideFolder == 0) //Jump from the root folder
            {
                insideFolder++;
            }
            else //handling if the directory doesnot exits
            {
                rmv = mkdir(temp_path, 0777);
                if (rmv == -1 && errno != EEXIST)
                    printf("Some error in function name: copyContent where if (rmv == -1 && errno != EEXIST)");
            }
        }

        // FTW_F represents regular file type.
        if (type == FTW_F)
            // IF this is statisfied COPY all files
            if (countExt == 0)
            {
```

```c
            // Link function creates a new hard link between the files specified by path and temp_path.
            rmv = link(path, temp_path);
            if (rmv == -1)
                printf("Some error in function name: copyContent where if (countExt == 0)");
        }
        else
        {
            // Only copyies file for where extension is not matched
            if (!validateExtension(path))
            {
                rmv = link(path, temp_path);
                if (rmv == -1)
                    printf("Some error in function name: copyContent where if (!validateExtension(path))");
            }
        }
    return 0;
}

// Create a Traget folder if doesnot exits
int creatFolder(const char *destPath)
{
    struct stat info; // used for retrieving information about files and directories.
    // st_mode stores file type & permissions.
    // S_ISDIR returns true if given mode represents a directory.
    if (!(stat(destPath, &info) == 0 && S_ISDIR(info.st_mode)))
    {
        int status = mkdir(destPath, 0777);
        if (status == 0)
            return 1;
        else
            return 0;
    }
    return 1;
}

// Function to copy files and directories
int copyDirectory(const char *sourcePath, const char *destPath)
```

```c
{
    // 5 represents the maximum number of file descriptors that the nftw function can open simultaneously.
    // FTW_PHYS physical walk of the file system
    return nftw(sourcePath, copyContent, 5, FTW_PHYS);
}

// Function to move files and directories
int moveDirectory(const char *sourcePath, const char *destPath)
{
    int varMov1;
    varMov1 = copyDirectory(sourcePath, destPath); // Copy function call
    if (varMov1 == -1)
        return varMov1;

    // REMOVING THE FILES AND DIRECTORY AFTER MOVING
    // Invokes nftw function to recursively traverse directory tree starting from the directory specified by src.
    // The remove function will be called on each file or directory encountered during the traversal,
    // and the traversal will be performed in a depth-first manner while
    // treating symbolic links as regular files or directories.

    varMov1 = nftw(sourcePath, remove, 5, FTW_DEPTH | FTW_PHYS);
    if (varMov1 == -1)
        printf("Some error in function name: moveDirectory where if (varMov1 == -1)");
    return varMov1;
}

//MAIN METHOD STARTS HERE

int main(int argCount, char *argVar[])
{
    if (argCount < 4)
    {
        // This will instruct the user with correct command which is required to be entered by user
        printf("Use SYNTEXT AS BELOW:\n%s Source_DirPath Destination_DirPath -cp or -mv {extensions which are to be excluded}\n", argVar[0]);
        return 1;
    }
    struct stat st;
    strcpy(sourcePath, argVar[1]);
    strcpy(destPath, argVar[2]);

    // Storing source and target directories paths in another variable
    // strcpy copies the contents of one string to another for this case the value from array is stored to these strings src & target
    // Error pops when source path is not found
    if (!(stat(sourcePath, &st) == 0 && S_ISDIR(st.st_mode)))
    {
        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]);
        return 1;
    }

    // This LOGIC WILL CREATE THE FOLDER IN CASE DOESNOT EXITS
    creatFolder(destPath);

     // Get the desired extensions
    if (argCount > 4)
    {
        // UPTO 6 EXTENSION CAN BE PROVIDED AS A LIST
        for (int i=4; i<argCount && i-4 < 6; i++)
        {
            strcpy(fileExt[i-4], argVar[i]);
            countExt++;
        }
    }

    // CHECK IF THE ACTION REQUIRED IS COPY OR MOVE
    // strcmp function used to compare 2 strings.
    if (strcmp(argVar[3], "-cp") == 0)
        return copyDirectory(sourcePath, destPath);
    if (strcmp(argVar[3], "-mv") == 0)
        return moveDirectory(sourcePath, destPath);
    else
    {
        // IF IN CASE NEITHER -CP NOR -MV IS PASSED BY USER HENCE HANDLE EXECPTION HERE
        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]);
        return 1;
    }
}
```