**Jacob Rook**

**Data Structure and Algorithms II**

**Project 1**

**Functional Decomposition**

**User-defined data structures used as parameters in the functions**

/\*

\* @brief DirectoryFile Structure used for the Directories and files.

\*

\* @var Type The distinction between a directory and a file will be made by the Type variable. A directory will have a "D" assigned to Type and a file will have an "F" assigned to Type.

\*

\* @var DirName This variable will store the name of the directory or file.

\*

\* @var Parent This variable is a pointer to the directory that the current file is in.

\*

\* @var Children This variable is the head of the directory/file that lists all of the children of the directory.

@var Sibling This variable is the next sibling adjacent to the directory

\*/

**typedef** **struct** DirFile {

**char** Type;

**char** DirName[MAXTOKENLEN];

**struct** DirFile \*Parent;

**struct** DirFile \*Children;

**struct** DirFile \*Siblings;

}DirectoryFile;

DirectoryFile ROOT; //This is the root for all the files has scope in all file that include Functions.h.

DirectoryFile \*Curr; //Point to the current directory has scope in all file that include Functions.h

/\*

\* @brief DirPtr Stack structure to store the file names to be popped off

\*

\* @var DirectoryName Name of name to be output when popped

\*

\* @var Next Pointer to next member in the stack

\*

\*/

**typedef** **struct** NStack{

DirectoryFile \*DirPtr;

**struct** NStack \*Next;

}Stack;

Stack \*Top; //Pointer for the top of the stack structure for pwd function and the cp function because the cp function needs two stacks to function. Has Scope in all files that include Functions.h

Stack \*Top2; //Pointer for the top of the stack structure for whereis function and secondary stack for cp function. Has scope in all file that include Functions.h

**#define** MAXTOKENS 4

**#define** MAXTOKENLEN 100

**char** Tokens[MAXTOKENS][MAXTOKENLEN]; //Variable to store the Tokens of the function. Variabel has scope to all files that include Functions.h

**Files and Functions in the Program**

/\*

\* @file main.c This file is the main C file for a basic directory that

\* processes basic Unix/Linux commands. Main handles the file i/o part

\* of the project. It reads the command line and passes the commands to

\* the Functions file.

\*

\* @author Jacob Rook

\* @date 09/21/2017

\* @info Course COP 4534

\* Project 1

\*/

**Main.c**

/\*

\* @file Functions.c This file contains all of the code for the functions

\* to execute the Linux\Unix commands.

\*

\* @author Jacob Rook

\* @date 09/05/2017

\* @info Course COP 4534

\* Project 1

\*/

**Functions.c Functions.h**

/\*

\* @brief Tokenizer This function tokenizes a string of characters passed by the user. And, stores them in the scope variable Tokens 2d array

\*

\* @param String[] String of characters to be tokenized

\*

\* @param TokenChar[] Character array to identify the end of a token

\*

\* @return int Returns the number of tokens

\*

\*/

**int** **Tokenizer** (**char** String[], **const** **char** TokenChar[]);

/\*

\* @brief CommandOperator This function determines which Unix\Linux command needed to be called and calls the function for that command. Uses Tokens variable to determine the commands.

\*

\* @return void

\*/

**int** **CommandOperator**(**void**);

/\*

\* @brief ls Lists all files and directories in the current directory, indicating which (file or directory) it is

\*

\*/

**void** **ls**(**void**);

/\*

\* @brief mkdir Creates a new directory if it does not already exist. Uses Insert function to insert the new node into the directory tree.

\*

\*/

**void** **mkdir**(**void**);

/\*

\* @brief Insert Inserts a new directory or file in its current list of directories alphabetically

\*

\* @param NewNode New Node to be placed into its current directory

\*/

**void** **Insert**(DirectoryFile \*NewNode);

/\*

\* @brief cd Changes into the specified directory if there. Uses FindDirFile function to find where the directory is.

\*

\*/

**void** **cd**(**void**);

/\*

\* @brief FindDirFile Function finds a directory or file in the current directory

\*

\* @param DirFileName Name of directory/file to find

\*

\* @return DirectoryFile\* Returns pointer to found directory or file. Returns NULL if not found.

\*/

DirectoryFile \***FindDirFile**(**char** DirFileName[]);

/\*

\* @brief FindPrevDirFile Function finds the node to the left of passed node. If the node does not have a left node NULL is returned. Used in the mv function and rm function.

\*

\* @param DirFileName Name of directory/file to be found

\*/

DirectoryFile \***FindPrevDirFile**(**char** DirFileName[]);

/\*

\* @brief pwd Specifies the current directory as: <yourname>/root/nextdir/etc/ Uses the stack structures to print the path

\*

\*/

**void** **pwd**(**void**);

/\*

\* @brief addf Adds a file to the current directory. Uses the Insert function to insert in alphabetized order.

\*

\*/

**void** **addf**(**void**);

/\*

\* @brief mv Change the name of the file or directory to the new name

\*/

**void** **mv**(**void**);

/\*

\* @brief cp Copy file or folder to the new name uses Duplicate function to assist in function.

\*

\*/

**void** **cp**(**void**);

/\*

\* @brief Duplicate Copies info from Original to Copy

\*

\* @param Copy Place to be copied to

\*

\* @param Original Original directory to be copied

\*/

**void** **Duplicate**(DirectoryFile \*Copy, DirectoryFile \*Original);

/\*

\* @brief Locate and remove the file or directory uses del function to recursively scan through the directory tree.

\*

\*/

**void** **rm**(**void**);

/\*

\* @brief del Secondary delete function to recursively delete nodes.

\*

\* @param Head Head node to be deleted

\*/

**void** **del**(DirectoryFile \*Head);

/\*

\* @brief bye Deallocates all allocated memory and ends session.

\*

\*/

**void** **bye**(**void**);

/\*

\* @brief whereis Show path to first occurrence of file or directory if it exists

\*/

**void** **whereis**(**void**);

/\*

\* @brief Recursive function for the whereis function

\*/

**void** **whereisrecur**(**void**);

/\*

\* @brief Push Push function for the NameStack stack

\*

\*

\* @param Directory Directory to be pushed onto the first stack (Top)

\*/

**void** **Push**(DirectoryFile \*Directory);

/\*

\* @brief Pop Pop function for the Top stack

\*

\*/

DirectoryFile \***Pop**(**void**);

/\*

\* @brief isEmpty Function to determine if the Top stack is empty or not

\*

\*/

**int** **isEmpty**(**void**);

/\*

\* @brief Push2 Push function for the Top2 stack for whereis

\*

\* @param Directory Directory to be pushed onto the Top2 stack

\*/

**void** **Push2**(DirectoryFile \*Directory);

/\*

\* @brief Pop2 Pop function for Top2 stack for whereis

\*

\*/

DirectoryFile \***Pop2**(**void**);

/\*

\* @brief isEmpty2 Function to determine if the Top2 stack for whereis is empty

\*

\*/

**int** **isEmpty2**(**void**);