/\*----------------------------------------------------------------------------

Meryl Mabin

CSE 410, section 001

proj01 - Find

---------------------------------------------------------------------------\*/

using namespace std;

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fnmatch.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <dirent.h>

#include <queue>

void readInput( queue<string> &options, queue<string> &targets,

int ArgCounter, char \*Arguments[], bool &recursion\_flag );

void Find( queue<string> &options, queue<string> &targets,

queue<string> t\_options, queue<string> t\_targets,

char \*Arguments[], bool recursion\_flag );

int main( int Argc, char \*Arg[] )

{

DIR\* direc;

queue<string> user\_options; // Queues for user input

queue<string> user\_targets;

queue<string> temp\_options;

queue<string> temp\_targets;

bool recursion\_flag = 0;

// If directory won't open, print error message and quit

if ( !(direc = opendir(Arg[1])) )

{

cout << "Invalid directory name.\n";

}

else

{

// Read user input - build the option and target queues

readInput( user\_options, user\_targets, Argc, Arg, recursion\_flag );

// Keep an extra copy of the options queue and targets queue

temp\_options = user\_options;

temp\_targets = user\_targets;

// If the queues are equal in size, Find

if ( (user\_options.size() == user\_targets.size()))

{

Find( user\_options, user\_targets, temp\_options, temp\_targets, Arg,

recursion\_flag );

} // end if (queue size check)

else

// User input isn't okay, so report error and break.

{

cout << "\n\*\*\* Invalid option-target combination entered. \*\*\*\n";

}

} // end directory input check

} // end main

/\*----------------------------------------------------------------------------

Name: readInput

Input: options queue, targets queue, arg counter, arg array

Purpose: Loop through user arguments and build queues to hold them

----------------------------------------------------------------------------\*/

void readInput( queue<string> &options, queue<string> &targets,

int ArgCounter, char \*Arguments[], bool &recursion\_flag )

{

string arg\_string;

recursion\_flag = 0;

// Loop through user arguments

for ( int I=2; I<ArgCounter; I++ )

{

arg\_string = Arguments[I];

// Is it some kind of option?

if ( arg\_string[0] == '-' )

{

// Is this a -R (recursive search) option?

if ( arg\_string == "-R" )

{

recursion\_flag = 1;

}

// Is this a different option? Then put it on the options stack.

else if ( arg\_string == "-name" )

{

options.push( arg\_string );

}

else if ( arg\_string == "-size" )

{

options.push( arg\_string );

}

else if ( arg\_string == "-uid" )

{

options.push( arg\_string );

}

else if ( arg\_string == "-gid" )

{

options.push( arg\_string );

}

else if ( arg\_string == "-atime" )

{

options.push( arg\_string );

}

else if ( arg\_string == "-mtime" )

{

options.push( arg\_string );

}

else if ( arg\_string == "-ctime" )

{

options.push( arg\_string );

}

else if ( arg\_string == "-perm" )

{

options.push( arg\_string );

}

} // end if (option tests)

else

// It failed the option test, so it must be a target.

{

targets.push( arg\_string );

}

} // end for (iterating over user args)

}

/\*----------------------------------------------------------------------------

Name: Find

Inputs: option and target queues, temp queues, arguments, flag, directory

Purpose: Search a directory for files matching the user's search query

----------------------------------------------------------------------------\*/

void Find( queue<string> &options, queue<string> &targets,

queue<string> t\_options, queue<string> t\_targets,

char \*Arguments[], bool recursion\_flag )

{

DIR\* directory = opendir(Arguments[1]);

dirent\* Reader; // For reading directories

string file\_name;

string holder;

const char \*file\_path;

string target;

string option;

struct stat buffer;

string flag = "";

int a\_flag = 0;

int found\_flag = 0;

string hours,

minutes,

seconds,

month,

day,

year,

target\_time,

file\_time,

s\_user\_perm,

s\_group\_perm,

s\_other\_perm;

unsigned int target\_int,

hundreds,

tens,

ones;

int perm,

file\_perm;

// If the queues are equal in size, continue

if ( (options.size() == targets.size()))

{

// Read directory until we run out of files

while ( (Reader = readdir(directory)) != NULL )

{

file\_name = Reader->d\_name;

holder = Arguments[1] + file\_name;

file\_path = holder.c\_str();

stat(file\_path, &buffer);

// For directories, change to that directory and search recursively

if ( (recursion\_flag != 0) && !(chdir(file\_path))

&& strcmp((file\_name).c\_str(), ".")

&& strcmp((file\_name).c\_str(), "..") )

{

directory = opendir(file\_path);

Find( options, targets, t\_options, t\_targets, Arguments,

recursion\_flag );

}

else

// While the target and option queues are not empty, check each option

// against its corresponding target (for each file).

{

while ( (options.size() > 0) && (targets.size() > 0) )

{

// Store the top elements of each queue

option = options.front();

target = targets.front();

// Is this a -name option search?

if ( option == "-name" )

{

if ( !fnmatch(target.c\_str(), file\_name.c\_str(), a\_flag) )

{

// flag = file\_path;

flag = file\_name;

}

else

{

flag = "";

break;

}

options.pop();

targets.pop();

continue;

} // end -name option

// Is this a -size option search?

else if ( option == "-size" )

{

if ( atoi(target.c\_str()) == buffer.st\_size )

{

// flag = file\_path;

flag = file\_name;

}

else

{

flag = "";

break;

}

options.pop();

targets.pop();

continue;

} // end -size option

// Is this a -uid option search?

else if ( option == "-uid" )

{

target\_int = atoi(target.c\_str());

if ( target\_int == buffer.st\_uid )

{

// flag = file\_path;

flag = file\_name;

}

else

{

flag = "";

break;

}

options.pop();

targets.pop();

continue;

} // end -uid option

// Is this a -gid option search?

else if ( option == "-gid" )

{

target\_int = atoi(target.c\_str());

if ( target\_int == buffer.st\_gid )

{

flag = file\_name;

// flag = file\_path;

}

else

{

flag = "";

break;

}

options.pop();

targets.pop();

continue;

} // end -gid option

// Is this a -atime option search?

else if ( option == "-atime" )

{

// cout << ctime(&buffer.st\_atime) << "\n";

if ( !fnmatch("\*:\*:\*-\*/\*/\*", target.c\_str(), 0)

&& strlen(target.c\_str()) == 17 )

{

// Parse the string for hours, minutes, etc.

hours = target.substr(0, 2);

minutes = target.substr(3, 2);

seconds = target.substr(6, 2);

month = target.substr(9, 2);

day = target.substr(12, 2);

year = target.substr(15, 2);

// Turn the numerical month into its name abbreviation

if ( month == "01" )

{

month = "Jan";

}

else if ( month == "02" )

{

month = "Feb";

}

else if ( month == "03" )

{

month = "Mar";

}

else if ( month == "04" )

{

month = "Apr";

}

else if ( month == "05" )

{

month = "May";

}

else if ( month == "06" )

{

month = "Jun";

}

else if ( month == "07" )

{

month = "Jul";

}

else if ( month == "08" )

{

month = "Aug";

}

else if ( month == "09" )

{

month = "Sep";

}

else if ( month == "10" )

{

month = "Oct";

}

else if ( month == "11" )

{

month = "Nov";

}

else if ( month == "12" )

{

month = "Dec";

}

else

{

cout << "\*\*\* Invalid date entered. \*\*\*\n";

exit(1);

}

target\_time = month + " " + day + " " + hours + ":" + minutes

+ ":" + seconds + " 20" + year;

// Turn the file time into a pattern string

file\_time = ctime(&buffer.st\_atime);

file\_time = file\_time.substr(4, 20);

file\_time = "\*" + file\_time;

// flag = file\_path;

flag = file\_name;

}

else

{

cout << "\*\*\* Invalid date entered. \*\*\*\n";

exit(1);

}

if ( !fnmatch( (file\_time.c\_str()), target\_time.c\_str(), 0) )

{

// flag = file\_path;

flag = file\_name;

}

else

{

flag = "";

}

options.pop();

targets.pop();

continue;

} // end -atime option

// Is this a -mtime option search?

else if ( option == "-mtime" )

{

if ( !fnmatch("\*:\*:\*-\*/\*/\*", target.c\_str(), 0)

&& strlen(target.c\_str()) == 17 )

{

// Parse the string for hours, minutes, etc.

hours = target.substr(0, 2);

minutes = target.substr(3, 2);

seconds = target.substr(6, 2);

month = target.substr(9, 2);

day = target.substr(12, 2);

year = target.substr(15, 2);

// Turn the numerical month into its name abbreviation

if ( month == "01" )

{

month = "Jan";

}

else if ( month == "02" )

{

month = "Feb";

}

else if ( month == "03" )

{

month = "Mar";

}

else if ( month == "04" )

{

month = "Apr";

}

else if ( month == "05" )

{

month = "May";

}

else if ( month == "06" )

{

month = "Jun";

}

else if ( month == "07" )

{

month = "Jul";

}

else if ( month == "08" )

{

month = "Aug";

}

else if ( month == "09" )

{

month = "Sep";

}

else if ( month == "10" )

{

month = "Oct";

}

else if ( month == "11" )

{

month = "Nov";

}

else if ( month == "12" )

{

month = "Dec";

}

else

{

cout << "\*\*\* Invalid date entered. \*\*\*\n";

exit(1);

}

target\_time = month + " " + day + " " + hours + ":" + minutes

+ ":" + seconds + " 20" + year;

// Turn the file time into a pattern string

file\_time = ctime(&buffer.st\_mtime);

file\_time = file\_time.substr(4, 20);

file\_time = "\*" + file\_time;

flag = file\_name;

}

else

{

cout << "\*\*\* Invalid date entered. \*\*\*\n";

exit(1);

}

if ( !fnmatch( (file\_time.c\_str()), target\_time.c\_str(), 0) )

{

flag = file\_name;

}

else

{

flag = "";

}

options.pop();

targets.pop();

continue;

} // end -mtime option

// Is this a -ctime option search?

else if ( option == "-ctime" )

{

if ( !fnmatch("\*:\*:\*-\*/\*/\*", target.c\_str(), 0)

&& strlen(target.c\_str()) == 17 )

{

// Parse the string for hours, minutes, etc.

hours = target.substr(0, 2);

minutes = target.substr(3, 2);

seconds = target.substr(6, 2);

month = target.substr(9, 2);

day = target.substr(12, 2);

year = target.substr(15, 2);

// Turn the numerical month into its name abbreviation

if ( month == "01" )

{

month = "Jan";

}

else if ( month == "02" )

{

month = "Feb";

}

else if ( month == "03" )

{

month = "Mar";

}

else if ( month == "04" )

{

month = "Apr";

}

else if ( month == "05" )

{

month = "May";

}

else if ( month == "06" )

{

month = "Jun";

}

else if ( month == "07" )

{

month = "Jul";

}

else if ( month == "08" )

{

month = "Aug";

}

else if ( month == "09" )

{

month = "Sep";

}

else if ( month == "10" )

{

month = "Oct";

}

else if ( month == "11" )

{

month = "Nov";

}

else if ( month == "12" )

{

month = "Dec";

}

else

{

cout << "\*\*\* Invalid date entered. \*\*\*\n";

exit(1);

}

target\_time = month + " " + day + " " + hours + ":" + minutes

+ ":" + seconds + " 20" + year;

// Turn the file time into a pattern string

file\_time = ctime(&buffer.st\_ctime);

file\_time = file\_time.substr(4, 20);

file\_time = "\*" + file\_time;

//flag = file\_path;

flag = file\_name;

}

else

{

cout << "\*\*\* Invalid date entered. \*\*\*\n";

exit(1);

}

if ( !fnmatch( (file\_time.c\_str()), target\_time.c\_str(), 0) )

{

flag = file\_name;

}

else

{

flag = "";

}

options.pop();

targets.pop();

continue;

} // end -ctime option

else if ( option == "-perm" )

{

// Make sure the permissions option is properly formatted

if ( (strlen(target.c\_str()) != 3)

|| target[0] < '0' || target[0] > '7'

|| target[1] < '0' || target[1] > '7'

|| target[2] < '0' || target[2] > '7')

{

cout << "\*\*\* Invalid permissions input. \*\*\*\n";

exit(1);

}

else

{

perm = atoi(target.c\_str());

file\_perm = buffer.st\_mode;

hundreds = perm/100;

tens = perm%100/10;

ones = perm%10;

if ( (((buffer.st\_mode & S\_IRWXU) >> 6) == hundreds)

&& (((buffer.st\_mode & S\_IRWXG) >> 3) == tens)

&& (((buffer.st\_mode & S\_IRWXO) >> 0) == ones) )

{

flag = file\_name;

}

else

{

flag = "";

}

}

options.pop();

targets.pop();

} // end -perm option

} // end while (check options/targets for one file; queues now empty)

// Reset the queues for the next file

options = t\_options;

targets = t\_targets;

// If the search worked for something, print it out

if ( !(flag == "") )

{

cout << flag << "\n";

found\_flag = 1;

}

} // end if (file or directory)

}// end while (read directory)

if ( found\_flag == 0 )

{

cout << "\*\*\* No matches found. \*\*\*\n";

}

} // end if (queue size check)

else

{

cout << "\*\*\* Invalid user input. \*\*\*\n";

}

}