LAB 4

Secure coding lab

Ananthanarayanan S

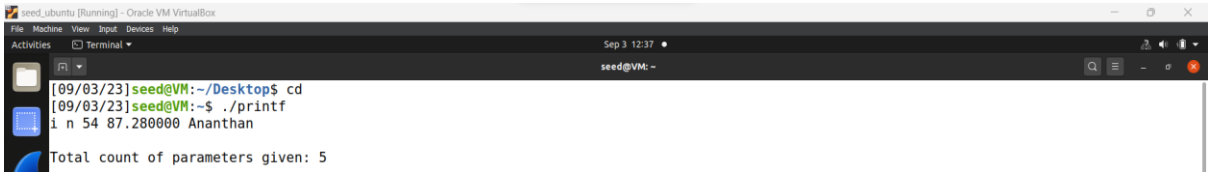
CB.EN.P2CYS23007

1.Write your own version of printf named myprintfunction().a.It should be able to accept various types of parameterssuch as char, int, double,etc.b.Bonus : The function should be able to accept different parameter count. The firstparameter says the count of parameters, followed by actual parameters.

```
#include <stdio.h>
#include <stdarg.h>

// Custom printf function that takes a format string and variable number of
arguments
void myprintf(const char *format, ...)
{
    va_list args;
    va_start(args, format); // Initialize the argument list

    int paramCount = 0; // Counter for the total number of parameters

    while (*format)
    {
        if (*format == '%') // Check for the format specifier
        {
            format++; // Move to the character after '%'

            switch (*format)
            {
                case 'c': // Character format specifier
                    paramCount++;
                    putchar(va_arg(args, int)); // Print a character from
the argument list
                    break;
                case 'd': // Integer format specifier
                    paramCount++;
                    printf("%d", va_arg(args, int)); // Print an integer
from the argument list
                    break;
                case 'f': // Floating-point format specifier
                    paramCount++;
                    printf("%f", va_arg(args, double)); // Print a double
from the argument list
                    break;
                case 's': // String format specifier
```

```
                    paramCount++;
                    fputs(va_arg(args, const char*), stdout); // Print a
string from the argument list
                    break;
                default: // If the format specifier is not recognized, print
the character itself
                    putchar(*format);
                    break;
            }
        }
        else
        {
            putchar(*format); // If not a format specifier, print the
character as is
        }

        format++; // Move to the next character in the format string
    }

    va_end(args); // Clean up the argument list

    printf("\nTotal count of parameters given: %d\n\n", paramCount); //
Print the total parameter count
}

int main()
{
    // Call the custom myprintf function with a format string and arguments
    myprintf("%c %c %d %f %s\n", 'i', 'n', 54, 87.28, "Ananthan");

    return 0;
}
```

Output

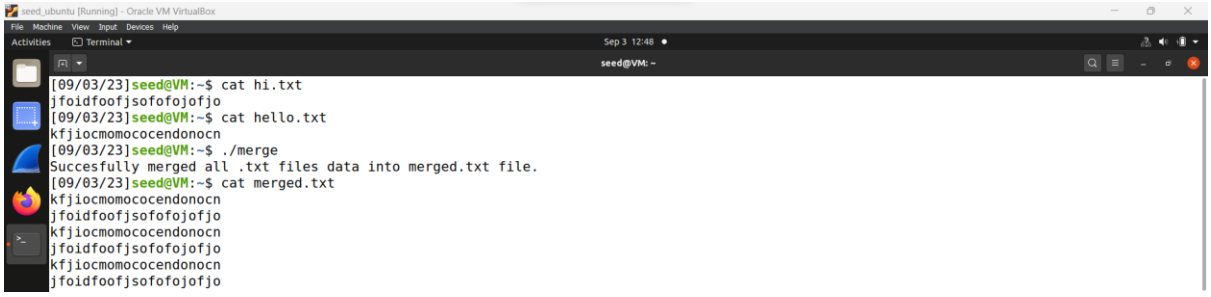


2.Write a program to read all txt files (that is files that ends with .txt) in the currentdirectory and merge them all to one text file and return a file descriptor for the newfile.

```
#include <stdio.h>
#include <dirent.h>
#include <string.h>

int main(void)
{
    FILE *ip, *op;       // Declare file pointers for input (ip) and output
(op)
    char ch;             // Declare a character variable ch
    char *txt = ".txt";  // Store the extension ".txt" in a string variable

    struct dirent *de;   // Declare a structure for directory entry
    DIR *dir = opendir(".");  // Open the current directory and get a
directory pointer

    if (dir == NULL)
    {
        printf("Can't open the current directory."); // Print an error
message if the directory couldn't be opened
        return 0;
    }

    while ((de = readdir(dir)) != NULL)  // Iterate over each entry in the
directory
    {
        char *filename = de->d_name;   // Get the name of the current
directory entry
        char *ext = strrchr(filename, '.');  // Find the last occurrence of
'.' in the filename

        if (!(ext == NULL || ext == filename))  // Check if a file has an
extension
        {
            if (strcmp(ext, txt) == 0)  // Check if the extension is ".txt"
            {
                op = fopen("merged.txt", "a+");  // Open or create
"merged.txt" for appending
                ip = fopen(filename, "r");       // Open the current .txt
file for reading

                while (1)
                {
                    ch = fgetc(ip);  // Read a character from the input file
                    if (ch == EOF)
                        break;  // Break the loop when the end of the file
is reached
```

```
                    putc(ch, op);   // Write the character to the output
file
                }

                fclose(ip);  // Close the input file
                fclose(op);  // Close the output file
            }
        }
    }

    closedir(dir);  // Close the directory

    printf("Successfully merged all .txt files data into merged.txt
file.\n");  // Print a success message
    return 0;
}
```

output



3.Write a program that will categorize all files in the current folder based on their file type. That is all .txt files in one folder called txt, all .bmp files in another folder called bmp etc. The argument to the program is a folder name

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>

int main(void)
{
    DIR *crdir;
    char *p1, *p2, ext[100][100], c, filename[50], path[100];

    for (int i = 0; i < 100; i++)
        strcpy(ext[i], "0");

    struct dirent *dir;
    crdir = opendir("."); // Open the current directory
```

```
    if (crdir)
    {
        while ((dir = readdir(crdir)) != NULL) // Iterate over each entry in
the current directory
        {
            p1 = strtok(dir->d_name, ".");  // Tokenize the filename using
'.'
            p2 = strtok(NULL, "."); // Get the extension of the filename

            if (p2 != NULL)
            {
                if (strcmp(ext[p2[0] - 97], "0") == 0)  // Check if the
extension is not yet recorded
                    strcpy(ext[p2[0] - 97], p2);  // Record the extension

                strcpy(filename, p1);  // Copy the base filename without
extension
                strcat(filename, ".");  // Add a period ('.') to the
filename
                strcat(filename, p2);  // Add the extension to the filename

                mkdir(p2, 0755);  // Create a directory with the extension
name

                strcpy(path, p2);  // Copy the extension to the path
                strcat(path, "/");  // Add a slash to the path
                strcat(path, filename);  // Add the filename to the path

                FILE *fp1 = fopen(path, "w");  // Open a file for writing in
the directory
                FILE *fp2 = fopen(filename, "r");  // Open the source file
for reading

                while ((c = fgetc(fp2)) != EOF)  // Read characters from the
source file
                    fputc(c, fp1);  // Write characters to the new file in
the directory

                fclose(fp1); // Close the destination file
                fclose(fp2); // Close the source file
            }
        }
        closedir(crdir);  // Close the current directory
    }
    return 0;
}
```

Output

```
┌──(kali㉿kali)-[~/Desktop]
└─$ ls
c            css_ssh  exec.c  f.C   fork.c  group.c  hello.c    hi.txt  ioj.txt   oi.bmp  system
css_exercise ex       f       fork  group   hello    hello.txt  hu.txt  makefile  okey    systemcalls.c

┌──(kali㉿kali)-[~/Desktop]
└─$ ./group

┌──(kali㉿kali)-[~/Desktop]
└─$ ls
bmp  C            css_ssh  exec.c  f.C   fork.c  group.c  hello.c    hi.txt  ioj.txt   oi.bmp  system        txt
c    css_exercise ex       f       fork  group   hello    hello.txt  hu.txt  makefile  okey    systemcalls.c

┌──(kali㉿kali)-[~/Desktop]
└─$
```

Strace:

Strace is mainly used for following functions

1. Debugging Programs: helps for troubleshooting issues by showing how a program interacts with the system.

2. Troubleshooting Programs: memory leaks

3. Intercepting system calls by a process: It traces all system calls issued by a program along with their return codes.

4. Recording system calls by a process: It returns the name of each system call along with its argument.

5. Process Monitoring: It allows to find out how a program is interacting with the OS.

```
└─$ strace -d ./group
strace: ptrace_setoptions = 0x51
strace: PTRACE_GET_SYSCALL_INFO works
strace: new tcb for pid 17017, active tcbs:1
strace: [wait(0x80137f) = 17017] WIFSTOPPED,sig=SIGSTOP,EVENT_STOP (128)
strace: next_event: queued pid 17017
strace: next_event: dequeued pid 17017
strace: pid 17017 has TCB_STARTUP, initializing it
strace: [wait(0x80057f) = 17017] WIFSTOPPED,sig=SIGTRAP,EVENT_STOP (128)
strace: next_event: queued pid 17017
strace: next_event: dequeued pid 17017
strace: [wait(0x00127f) = 17017] WIFSTOPPED,sig=SIGCONT
strace: next_event: queued pid 17017
strace: next_event: dequeued pid 17017
strace: [wait(0x00857f) = 17017] WIFSTOPPED,sig=133
strace: next_event: queued pid 17017
strace: next_event: dequeued pid 17017
sstrace: [wait(0x00857f) = 17017] WIFSTOPPED,sig=133
strace: next_event: queued pid 17017
strace: next_event: dequeued pid 17017
strace: [wait(0x00857f) = 17017] WIFSTOPPED,sig=133
strace: next_event: queued pid 17017
strace: next_event: dequeued pid 17017
trace: seccomp filter disabled
strace: [wait(0x00857f) = 17017] WIFSTOPPED,sig=133
strace: next_event: queued pid 17017
strace: next_event: dequeued pid 17017
strace: [wait(0x00857f) = 17017] WIFSTOPPED,sig=133
strace: next_event: queued pid 17017
strace: next_event: dequeued pid 17017
execve("./group", ["./group"], 0x7fffc85f9168 /* 54 vars */strace: [wait(0x04057f) = 17017] WIFSTOPPED,sig=SIGTRAP,EVENT_EXEC (4)
```

strace -d : print debugging output

```
┌──(kali㉿kali)-[~/Desktop]
└─$ strace --seccomp-bpf ./group
strace: --seccomp-bpf cannot be used without -f/--follow-forks, disabling
execve("./group", ["./group"], 0x7ffd04ecebe8 /* 54 vars */) = 0
brk(NULL)                               = 0x563fc2765000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f17e70cd000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=86318, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 86318, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f17e70b7000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220x\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1926256, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 1974096, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f17e6ed5000
mmap(0x7f17e6efb000, 1396736, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7f17e6efb000
mmap(0x7f17e7050000, 344064, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17b000) = 0x7f17e7050000
mmap(0x7f17e70a4000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1cf000) = 0x7f17e70a4000
mmap(0x7f17e70aa000, 53072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f17e70aa000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f17e6ed2000
arch_prctl(ARCH_SET_FS, 0x7f17e6ed2740) = 0
set_tid_address(0x7f17e6ed2a10)         = 18558
set_robust_list(0x7f17e6ed2a20, 24)     = 0
rseq(0x7f17e6ed3060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f17e70a4000, 16384, PROT_READ) = 0
mprotect(0x563fc218f000, 4096, PROT_READ) = 0
mprotect(0x7f17e70ff000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
```

Seccomp-BPF (Seccomp: Secure Computing Mode with Berkeley Packet Filter) is a Linux kernel feature that is used to restrict the system calls that a process can make.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ strace -i ./group
[00007f148975067] execve("./group", ["./group"], 0x7fffb7f4d938 /* 54 vars */) = 0
[00007f353a6ae157] brk(NULL)                               = 0x56426d2f4000
[00007f353a6af1b3] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f353a68d000
[00007f353a6aeef7] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
[00007f353a6af02d] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[00007f353a6aee3a] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=86318, ...}, AT_EMPTY_PATH) = 0
[00007f353a6af1b3] mmap(NULL, 86318, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f353a677000
[00007f353a6aef17] close(3)                                = 0
[00007f353a6af02d] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
[00007f353a6af084] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220x\2\0\0\0\0\0"..., 832) = 832
[00007f353a6af0aa] pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
[00007f353a6aee3a] newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1926256, ...}, AT_EMPTY_PATH) = 0
[00007f353a6af0aa] pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
[00007f353a6af1b3] mmap(NULL, 1974096, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f353a495000
[00007f353a6af1b3] mmap(0x7f353a4bb000, 1396736, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7f353a4bb000
[00007f353a6af1b3] mmap(0x7f353a610000, 344064, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17b000) = 0x7f353a610000
[00007f353a6af1b3] mmap(0x7f353a664000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1cf000) = 0x7f353a664000
[00007f353a6af1b3] mmap(0x7f353a66a000, 53072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f353a66a000
[00007f353a6aef17] close(3)                                = 0
[00007f353a6af1b3] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f353a492000
[00007f353a6a9c95] arch_prctl(ARCH_SET_FS, 0x7f353a492740) = 0
[00007f353a6a117a] set_tid_address(0x7f353a492a10) = 19327
[00007f353a6a11d6] set_robust_list(0x7f353a492a20, 24) = 0
[00007f353a6a120d] rseq(0x7f353a493060, 0x20, 0, 0x53053053) = 0
[00007f353a6af217] mprotect(0x7f353a664000, 16384, PROT_READ) = 0
[00007f353a6af217] mprotect(0x56426d0a9000, 4096, PROT_READ) = 0
[00007f353a6af217] mprotect(0x7f353a6bf000, 8192, PROT_READ) = 0
[00007f353a591f70] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
```

print instruction pointer at time of sysytem call

```
┌──(kali㉿kali)-[~/Desktop]
└─$ strace -xx ./group
execve("\x2e\x2f\x67\x72\x6f\x75\x70", ["\x2e\x2f\x67\x72\x6f\x75\x70"], 0x7ffd7ae82598 /* 54 vars */) = 0
brk(NULL)                               = 0x55a810437000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fda410c7000
access("\x2f\x65\x74\x63\x2f\x6c\x64\x2e\x73\x6f\x2e\x70\x72\x65\x6c\x6f\x61\x64", R_OK) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "\x2f\x65\x74\x63\x2f\x6c\x64\x2e\x73\x6f\x2e\x63\x61\x63\x68\x65", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=86318, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 86318, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fda410b1000
close(3)                                = 0
openat(AT_FDCWD, "\x2f\x6c\x69\x62\x2f\x78\x38\x36\x5f\x36\x34\x2d\x6c\x69\x6e\x75\x78\x2d\x67\x6e\x75\x2f\x6c\x69\x62\x63\x2e\x73\x6f\x2e\x36",
O_RDONLY|O_CLOEXEC) = 3
read(3, "\x7f\x45\x4c\x46\x02\x01\x01\x03\x00\x00\x00\x00\x00\x00\x00\x00\x03\x00\x3e\x00\x01\x00\x00\x00\x90\x78\x02\x00\x00\x00\x00\x00"..., 83
2) = 832
pread64(3, "\x06\x00\x00\x00\x04\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00"...,
 784, 64) = 784
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1926256, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\x06\x00\x00\x00\x04\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00"...,
 784, 64) = 784
mmap(NULL, 1974096, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fda40ecf000
mmap(0x7fda40ef5000, 1396736, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7fda40ef5000
mmap(0x7fda4104a000, 344064, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17b000) = 0x7fda4104a000
mmap(0x7fda4109e000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1cf000) = 0x7fda4109e000
mmap(0x7fda410a4000, 53072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fda410a4000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fda40ecc000
arch_prctl(ARCH_SET_FS, 0x7fda40ecc740) = 0
set_tid_address(0x7fda40ecca10)         = 21097
set_robust_list(0x7fda40ecca20, 24)     = 0
rseq(0x7fda40ecd060, 0x20, 0, 0x53053053) = 0
mprotect(0x7fda4109e000, 16384, PROT_READ) = 0
```

the -xx option in strace is used to provide detailed information about the memory locations and data associated with system call arguments

4. Given a directory, write a program that will find all files with the same name in the directory and its sub directories. Show their name, which folder they are in and what day they were created. Expand the program to remove all duplicate copies based on user input. That is, ask the user if each one of the files is to be kept or deleted. Based on user input, perform the appropriate action.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>

#define MAX 1000

void find_files(char *basePath, char *filename, int *count, char
paths[MAX][MAX]);
void remove_duplicates(char paths[MAX][MAX], int count);

int main()
{
    char filename[MAX];
    char basePath[MAX];
    char paths[MAX][MAX];
    int count = 0;

    printf("Enter the directory path: ");
    scanf("%s", basePath);

    printf("Enter the filename to search for: ");
    scanf("%s", filename);

    find_files(basePath, filename, &count, paths); // Call the find_files
function to search for files

    if (count == 0)
        printf("No files found with the name '%s'\\n", filename); // Print a
message if no files were found
    else
        remove_duplicates(paths, count); // Call the remove_duplicates
function to manage duplicate files

    return 0;
}

void find_files(char *basePath, char *filename, int *count, char
paths[MAX][MAX])
```

```c
{
    char path[MAX];
    struct dirent *dp;
    struct stat buffer;
    DIR *dir = opendir(basePath); // Open the directory specified by
basePath

    if (!dir)
        return;

    while ((dp = readdir(dir)) != NULL) // Iterate over directory entries
    {
        if (strcmp(dp->d_name, ".") != 0 && strcmp(dp->d_name, "..") != 0)
        {
            strcpy(path, basePath); // Construct the full path to the file
or directory
            strcat(path, "/");
            strcat(path, dp->d_name);

            if (stat(path, &buffer) == 0 && S_ISDIR(buffer.st_mode)) //
Check if it's a directory
                find_files(path, filename, count, paths); // Recursively
search inside subdirectories
            else if (strcmp(dp->d_name, filename) == 0) // Check if the
filename matches the target
            {
                printf("File found: %s\\n", path); // Print the path of the
found file
                printf("Folder: %s\\n", basePath); // Print the folder
containing the file
                printf("Creation time: %s\\n", ctime(&buffer.st_ctime)); //
Print the creation time
                strcpy(paths[*count], path); // Store the path in the paths
array
                (*count)++; // Increment the count of found files
            }
        }
    }

    closedir(dir); // Close the directory
}

void remove_duplicates(char paths[MAX][MAX], int count)
{
    char ch;
    int i;

    for (i = 0; i < count; i++)
```

```c
    {
        printf("\\nDo you want to keep or delete file '%s'? (k/d): ",
paths[i]); // Ask the user to keep or delete
        scanf(" %c", &ch);

        if (ch == 'd' || ch == 'D') // If the user chooses to delete
        {
            if (remove(paths[i]) == 0) // Remove the file and check if it
was successful
                printf("File '%s' deleted successfully.\\n", paths[i]); //
Print a success message
            else
                printf("Unable to delete file '%s'.\\n", paths[i]); // Print
an error message if deletion fails
        }
        else
            printf("File '%s' kept.\\n", paths[i]); // Print a message
indicating the file was kept
    }
}
```

Output