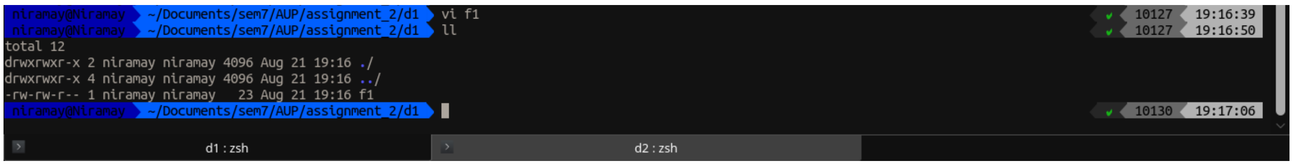
Assignment 2

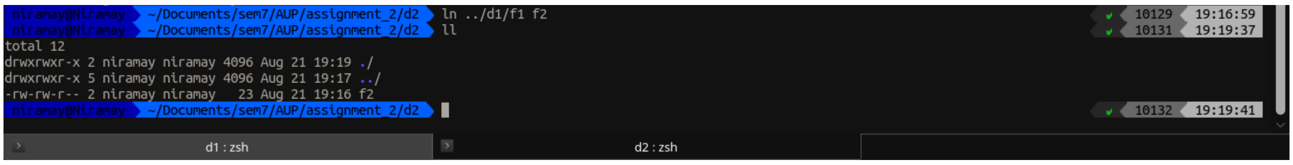
Niramay Vaidya 111605075
Srishti Shelke 111603056

1. Create a file named “f1” in directory “D1” and user 1 with appropriate access right to it. Create a link “f2” to “f1” in directory “D2” and give user 1 with appropriate access right to it. Demonstrate that both users can work on same file from different directories.



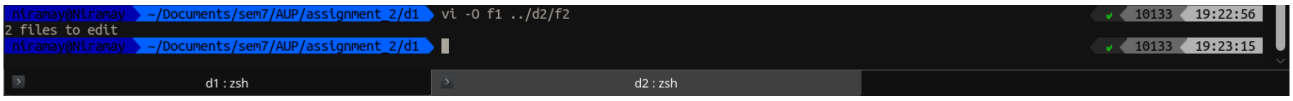
```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d1  vi f1          ✓  10127  19:16:39
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d1  ll             ✓  10127  19:16:50
total 12
drwxrwxr-x 2 niramay niramay 4096 Aug 21 19:16 ./
drwxrwxr-x 4 niramay niramay 4096 Aug 21 19:16 ../
-rw-rw-r-- 1 niramay niramay   23 Aug 21 19:16 f1
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d1                 ✓  10130  19:17:06
d1 : zsh                                        d2 : zsh
```

rw permissions for file f1 in directory d1 for owner, group and r permission for others (opened using vim)
Owner of the file is niramay



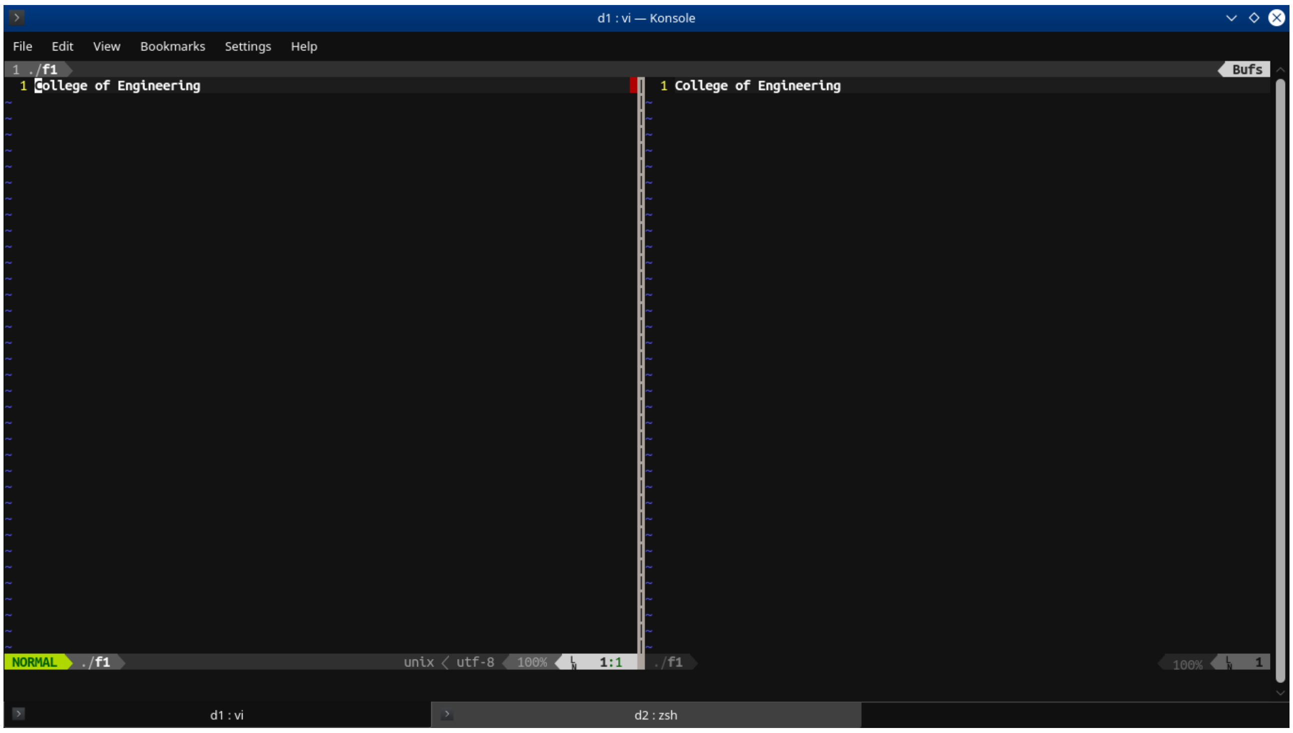
```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d2  ln ../d1/f1 f2   ✓  10129  19:16:59
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d2  ll               ✓  10131  19:19:37
total 12
drwxrwxr-x 2 niramay niramay 4096 Aug 21 19:19 ./
drwxrwxr-x 5 niramay niramay 4096 Aug 21 19:17 ../
-rw-rw-r-- 2 niramay niramay   23 Aug 21 19:16 f2
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d2                   ✓  10132  19:19:41
d1 : zsh                                        d2 : zsh
```

f2 created in d2 as hard link to f1 in d1 using ln command
rw permissions for f2 for owner, group and r permission for others
Owner of the file is niramay



```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d1  vi -O f1 ../d2/f2   ✓  10133  19:22:56
2 files to edit
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d1                      ✓  10133  19:23:15
d1 : zsh                                        d2 : zsh
```

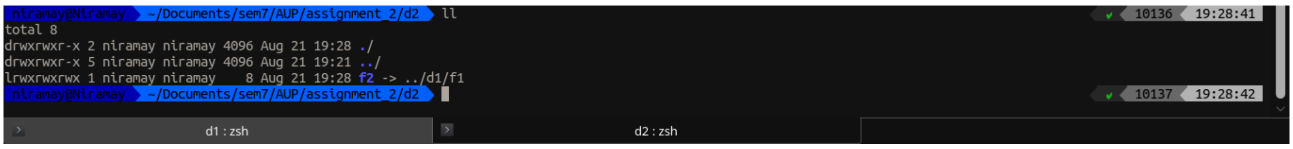
Opening both files simultaneously using vi -O f1 ../d2/f2
College of Engineering was written in f1 previously
Same text visible when f2 is opened



```
1 College of Engineering          1 College of Engineering
./f1   unix < utf-8 100% 1:1      ./f1   100% 1
```

If any change is made to f2 while f1 is open for editing too, the change in f2 is immediately reflected side by side in f1.

f2 created in d2 as symbolic link to f1 in d1 using ln command
rwx permissions for f2 for owner, group and others
Owner of the file is niramay

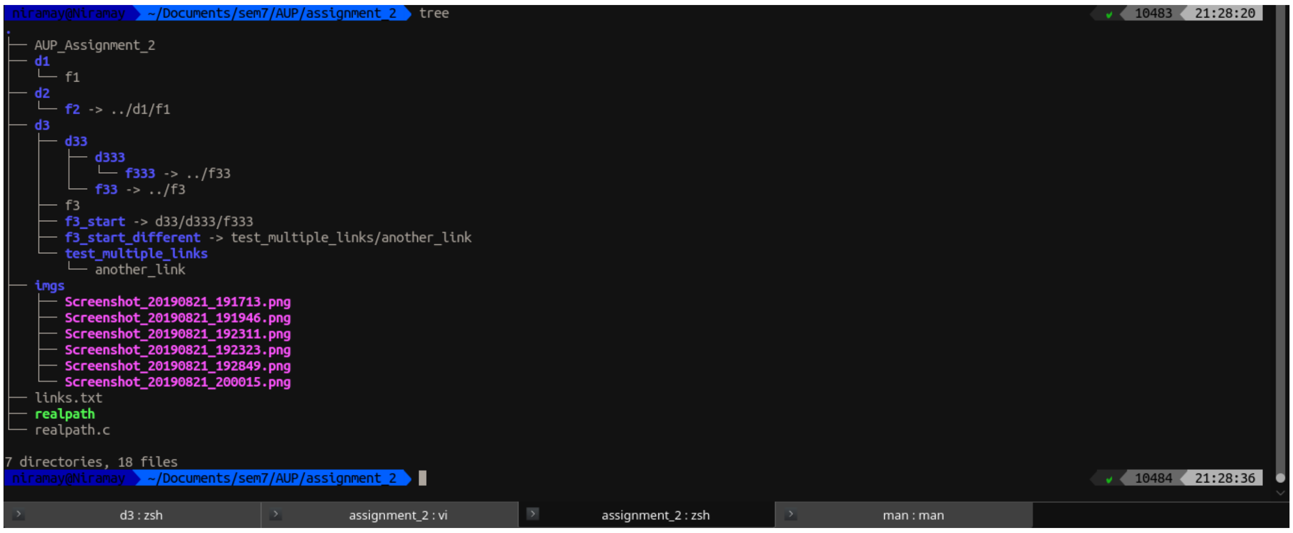


```
niramay@Niramay ~/Documents/sem7/AUP/assignment_2/d2 ll                   10136 19:28:41
total 8
drwxrwxr-x 2 niramay niramay 4096 Aug 21 19:28 ./
drwxrwxr-x 5 niramay niramay 4096 Aug 21 19:21 ../
lrwxrwxrwx 1 niramay niramay    8 Aug 21 19:28 f2 -> ../d1/f1
niramay@Niramay ~/Documents/sem7/AUP/assignment_2/d2                      10137 19:28:42
```

Same behaviour is observed even in case of creating a symbolic link i.e. both files can be opened exactly as above simultaneously for editing and changes in one are immediately reflected in the other.

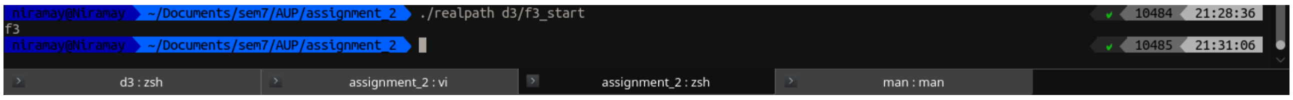
2. A function realpath() resolves all symbolic links in path and returns the *ultimate* target. Write a program to simulate realpath() to list the ultimate target of the only filename that are symbolic links in a directory. The program takes one optional argument, which is the name of a directory to be searched for the links. When no argument is specified, the search is conducted in the current working directory. Display appropriate error messages. Demonstrate by creating symbolic links A->B->C etc.

tree to display the symbolic links created

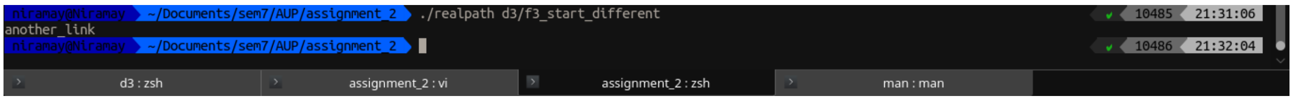


```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2  tree                    ✔  10483  21:28:20
.
├── AUP_Assignment_2
├── d1
│   └── f1
├── d2
│   └── f2 -> ../d1/f1
├── d3
│   ├── d33
│   │   ├── d333
│   │   │   └── f333 -> ../f33
│   │   └── f33 -> ../f3
│   ├── f3
│   ├── f3_start -> d33/d333/f333
│   ├── f3_start_different -> test_multiple_links/another_link
│   └── test_multiple_links
│       └── another_link
├── imgs
│   ├── Screenshot_20190821_191713.png
│   ├── Screenshot_20190821_191946.png
│   ├── Screenshot_20190821_192311.png
│   ├── Screenshot_20190821_192323.png
│   ├── Screenshot_20190821_192849.png
│   └── Screenshot_20190821_200015.png
├── links.txt
├── realpath
└── realpath.c

7 directories, 18 files
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2                          ✔  10484  21:28:36
```

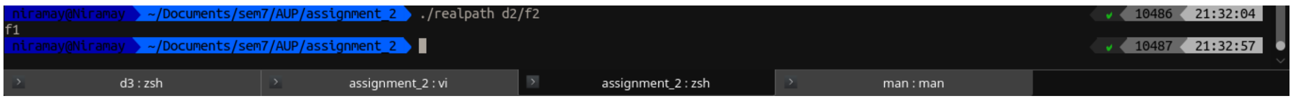
realpath run with parameters-



```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2  ./realpath d3/f3_start   ✔  10484  21:28:36
f3
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2                          ✔  10485  21:31:06
```

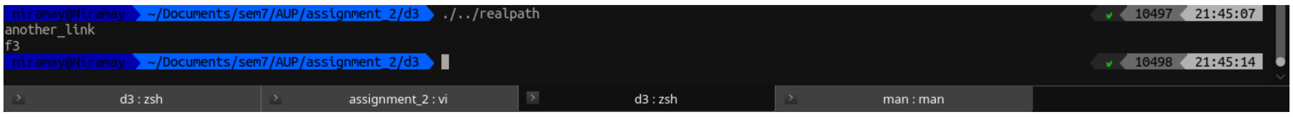


```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2  ./realpath d3/f3_start_different   ✔  10485  21:31:06
another_link
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2                          ✔  10486  21:32:04
```

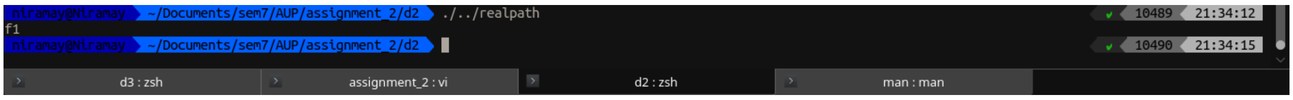


```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2  ./realpath d2/f2         ✔  10486  21:32:04
f1
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2                          ✔  10487  21:32:57
```

realpath run without parameters-

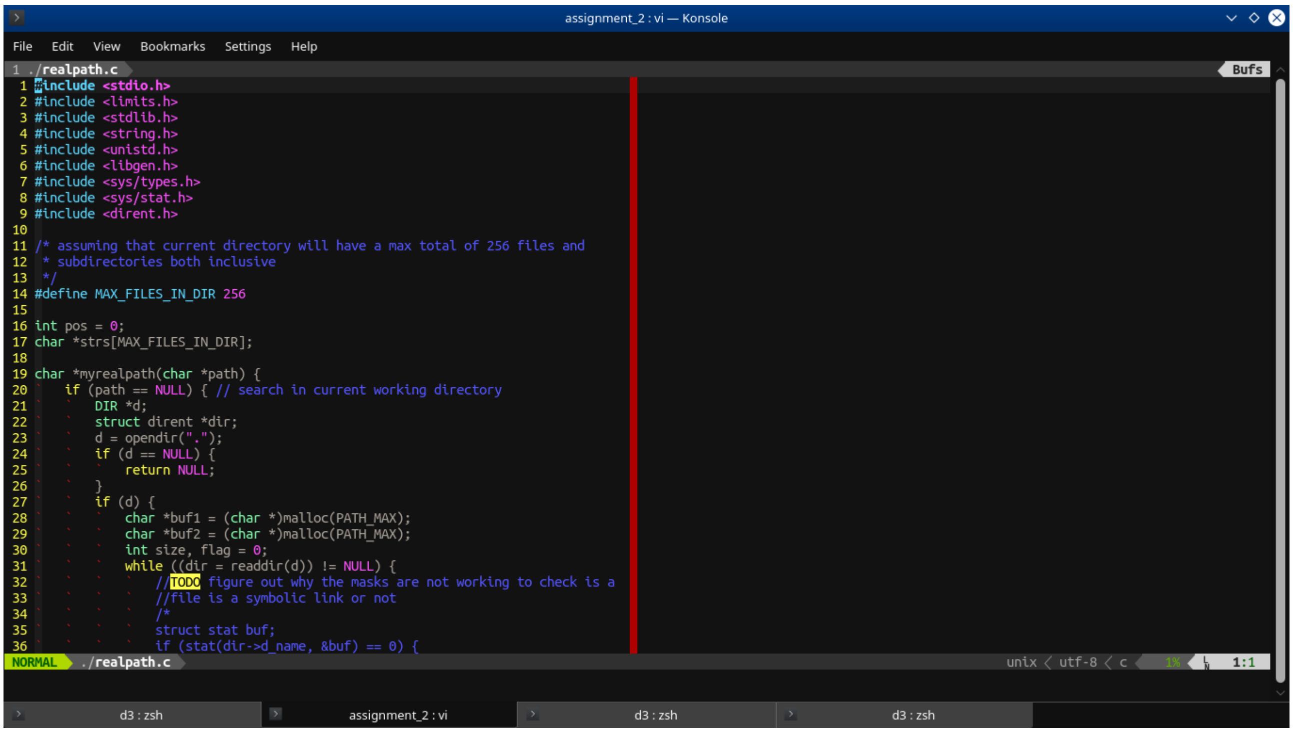


```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d3  ./../realpath        ✔  10497  21:45:07
another_link
f3
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d3                       ✔  10498  21:45:14
```

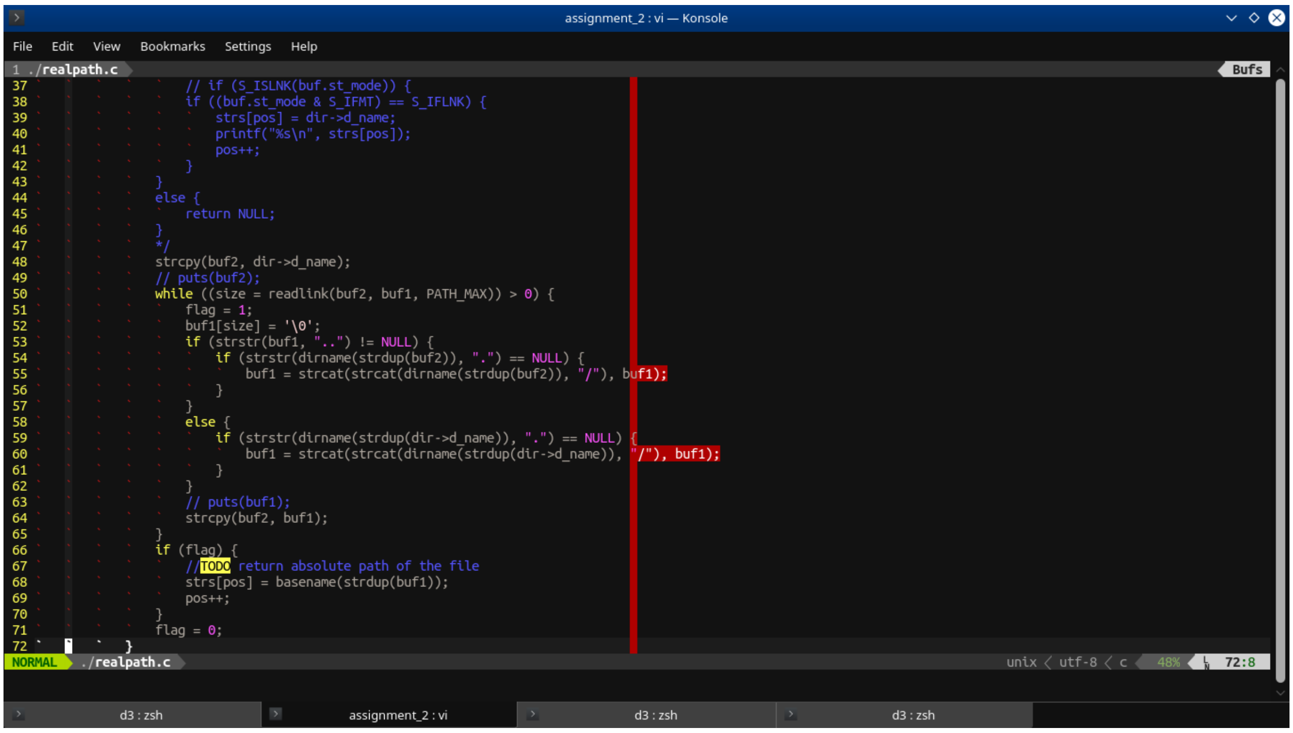


```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d2  ./../realpath        ✔  10489  21:34:12
f1
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2/d2                       ✔  10490  21:34:15
```

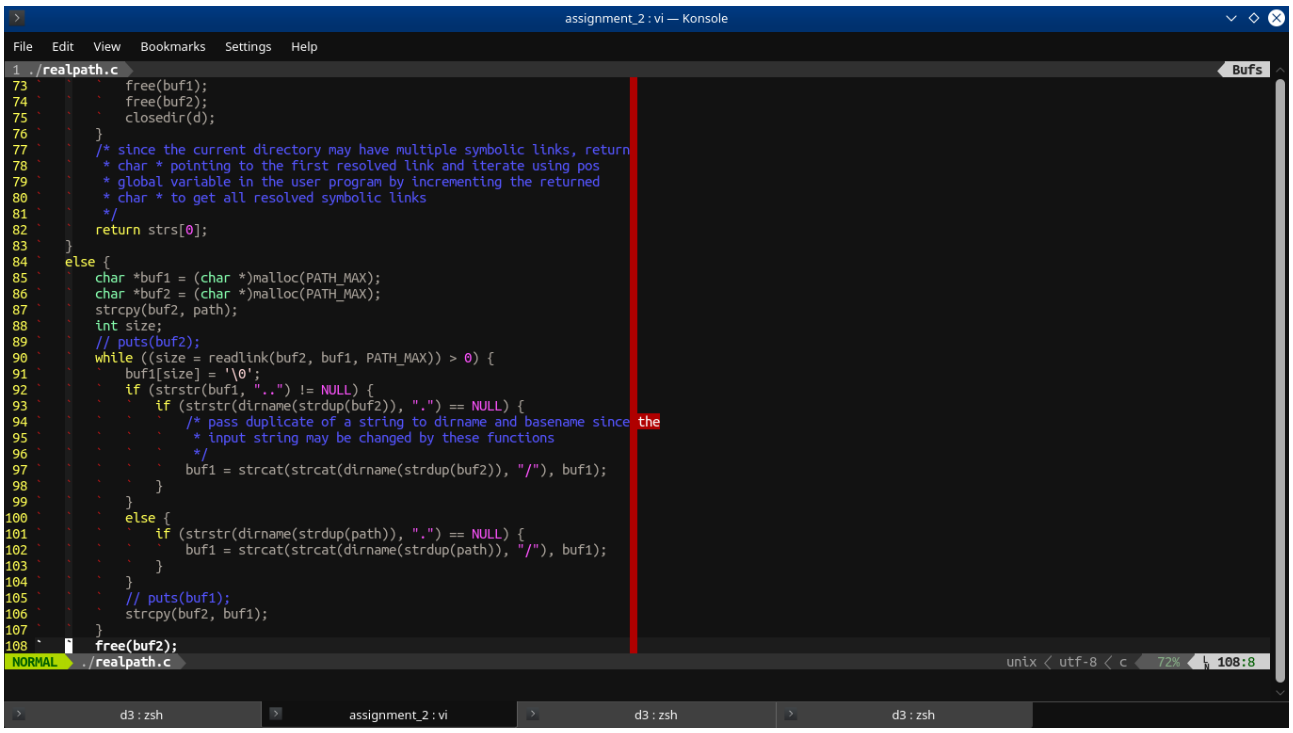
realpath program-



```c
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <libgen.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>

/* assuming that current directory will have a max total of 256 files and
 * subdirectories both inclusive
 */
#define MAX_FILES_IN_DIR 256

int pos = 0;
char *strs[MAX_FILES_IN_DIR];

char *myrealpath(char *path) {
    if (path == NULL) { // search in current working directory
        DIR *d;
        struct dirent *dir;
        d = opendir(".");
        if (d == NULL) {
            return NULL;
        }
        if (d) {
            char *buf1 = (char *)malloc(PATH_MAX);
            char *buf2 = (char *)malloc(PATH_MAX);
            int size, flag = 0;
            while ((dir = readdir(d)) != NULL) {
                //TODO figure out why the masks are not working to check is a
                //file is a symbolic link or not
                /*
                struct stat buf;
                if (stat(dir->d_name, &buf) == 0) {
```

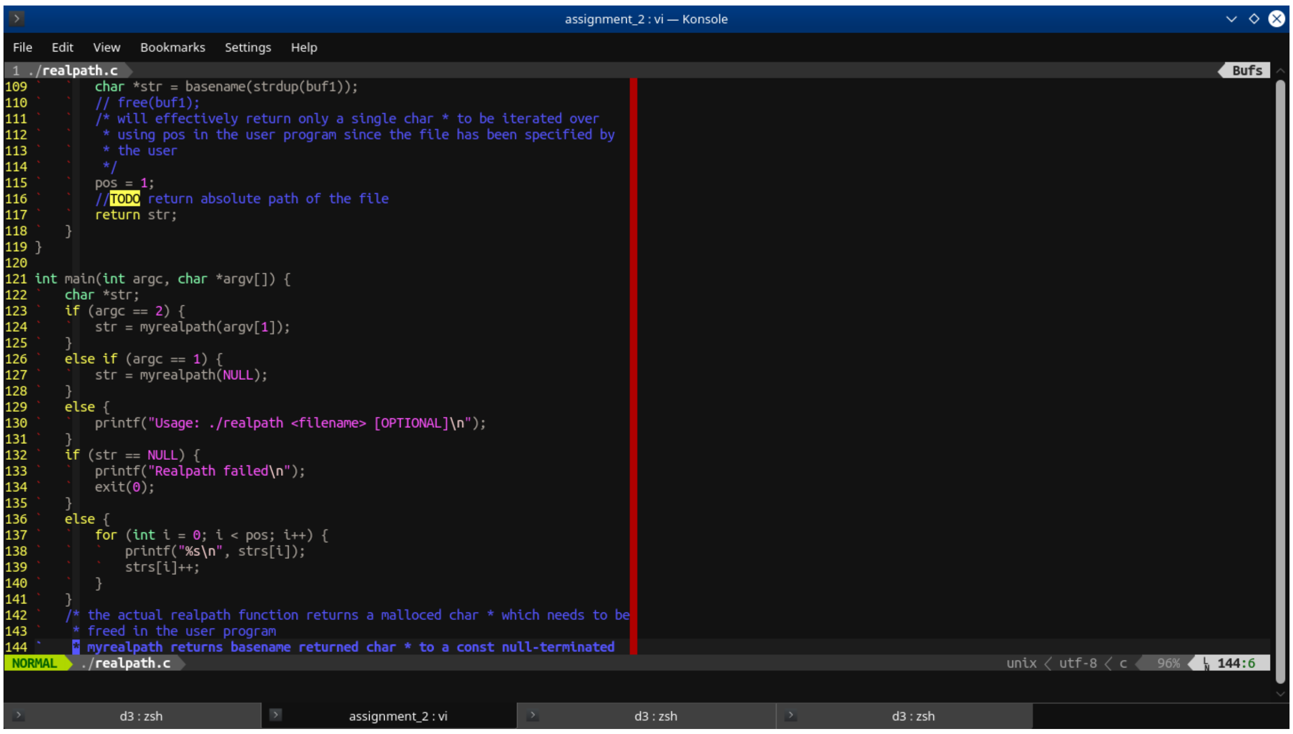


```c
                    // if (S_ISLNK(buf.st_mode)) {
                    if ((buf.st_mode & S_IFMT) == S_IFLNK) {
                        strs[pos] = dir->d_name;
                        printf("%s\n", strs[pos]);
                        pos++;
                    }
                }
                else {
                    return NULL;
                }
                */
                strcpy(buf2, dir->d_name);
                // puts(buf2);
                while ((size = readlink(buf2, buf1, PATH_MAX)) > 0) {
                    flag = 1;
                    buf1[size] = '\0';
                    if (strstr(buf1, "..") != NULL) {
                        if (strstr(dirname(strdup(buf2)), ".") == NULL) {
                            buf1 = strcat(strcat(dirname(strdup(buf2)), "/"), buf1);
                        }
                    }
                    else {
                        if (strstr(dirname(strdup(dir->d_name)), ".") == NULL) {
                            buf1 = strcat(strcat(dirname(strdup(dir->d_name)), "/"), buf1);
                        }
                    }
                    // puts(buf1);
                    strcpy(buf2, buf1);
                }
                if (flag) {
                    //TODO return absolute path of the file
                    strs[pos] = basename(strdup(buf1));
                    pos++;
                }
                flag = 0;
            }
```



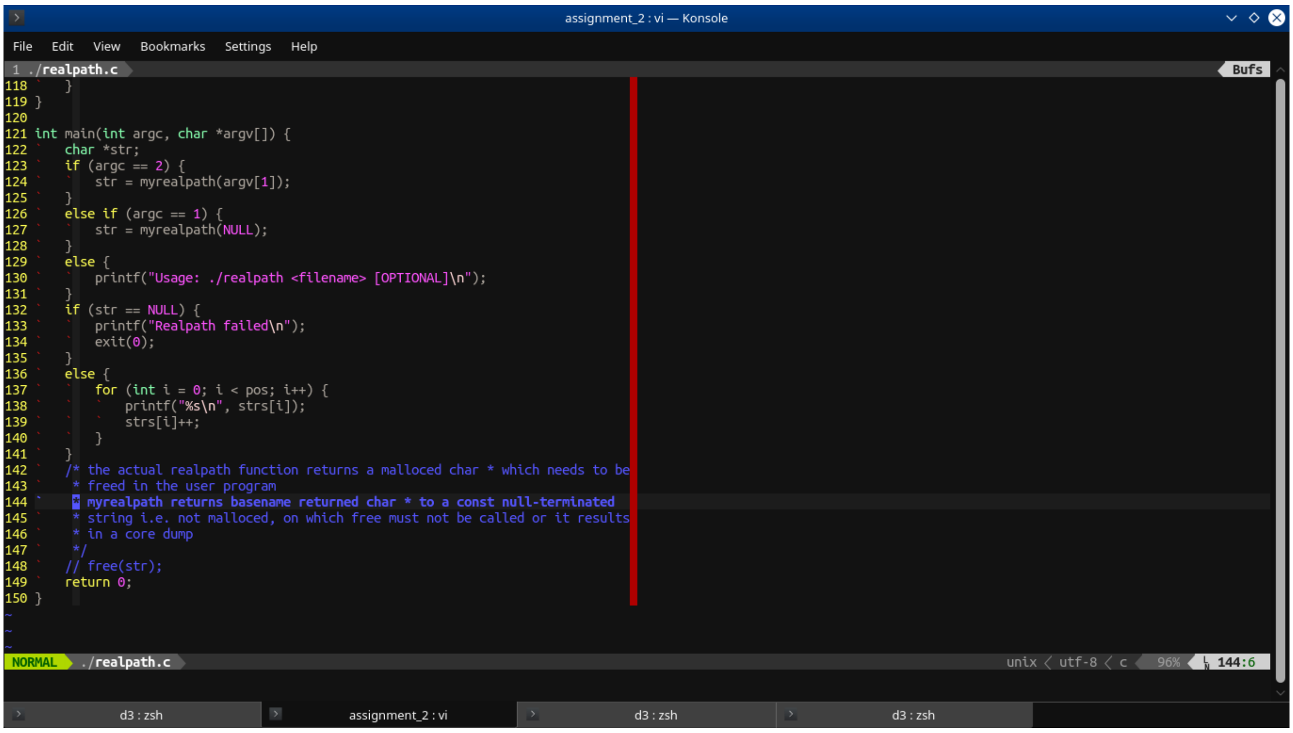
```c
            free(buf1);
            free(buf2);
            closedir(d);
        }
        /* since the current directory may have multiple symbolic links, return
         * char * pointing to the first resolved link and iterate using pos
         * global variable in the user program by incrementing the returned
         * char * to get all resolved symbolic links
         */
        return strs[0];
    }
    else {
        char *buf1 = (char *)malloc(PATH_MAX);
        char *buf2 = (char *)malloc(PATH_MAX);
        strcpy(buf2, path);
        int size;
        // puts(buf2);
        while ((size = readlink(buf2, buf1, PATH_MAX)) > 0) {
            buf1[size] = '\0';
            if (strstr(buf1, "..") != NULL) {
                if (strstr(dirname(strdup(buf2)), ".") == NULL) {
                    /* pass duplicate of a string to dirname and basename since the
                     * input string may be changed by these functions
                     */
                    buf1 = strcat(strcat(dirname(strdup(buf2)), "/"), buf1);
                }
            }
            else {
                if (strstr(dirname(strdup(path)), ".") == NULL) {
                    buf1 = strcat(strcat(dirname(strdup(path)), "/"), buf1);
                }
            }
            // puts(buf1);
            strcpy(buf2, buf1);
        }
        free(buf2);
```

```
NORMAL  ./realpath.c                                unix < utf-8 < c  72%  108:8
```



```c
        char *str = basename(strdup(buf1));
        // free(buf1);
        /* will effectively return only a single char * to be iterated over
         * using pos in the user program since the file has been specified by
         * the user
         */
        pos = 1;
        //TODO return absolute path of the file
        return str;
    }
}

int main(int argc, char *argv[]) {
    char *str;
    if (argc == 2) {
        str = myrealpath(argv[1]);
    }
    else if (argc == 1) {
        str = myrealpath(NULL);
    }
    else {
        printf("Usage: ./realpath <filename> [OPTIONAL]\n");
    }
    if (str == NULL) {
        printf("Realpath failed\n");
        exit(0);
    }
    else {
        for (int i = 0; i < pos; i++) {
            printf("%s\n", strs[i]);
            strs[i]++;
        }
    }
    /* the actual realpath function returns a malloced char * which needs to be
     * freed in the user program
     * myrealpath returns basename returned char * to a const null-terminated
```

```
NORMAL  ./realpath.c                                unix < utf-8 < c  96%  144:6
```

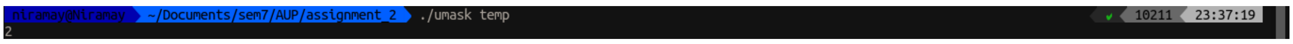


```
1 ./realpath.c
    }
}

int main(int argc, char *argv[]) {
    char *str;
    if (argc == 2) {
        str = myrealpath(argv[1]);
    }
    else if (argc == 1) {
        str = myrealpath(NULL);
    }
    else {
        printf("Usage: ./realpath <filename> [OPTIONAL]\n");
    }
    if (str == NULL) {
        printf("Realpath failed\n");
        exit(0);
    }
    else {
        for (int i = 0; i < pos; i++) {
            printf("%s\n", strs[i]);
            strs[i]++;
        }
    }
    /* the actual realpath function returns a malloced char * which needs to be
     * freed in the user program
     * myrealpath returns basename returned char * to a const null-terminated
     * string i.e. not malloced, on which free must not be called or it results
     * in a core dump
     */
    // free(str);
    return 0;
}
~
~
~
NORMAL  ./realpath.c                                    unix < utf-8 < c  96%  144:6
```

3. Create a shared directory for usage with a purpose that any user (not super user) can create new files in this directory, but only the owner can delete his own files and everyone else can read all files. Demonstrate the functionality.

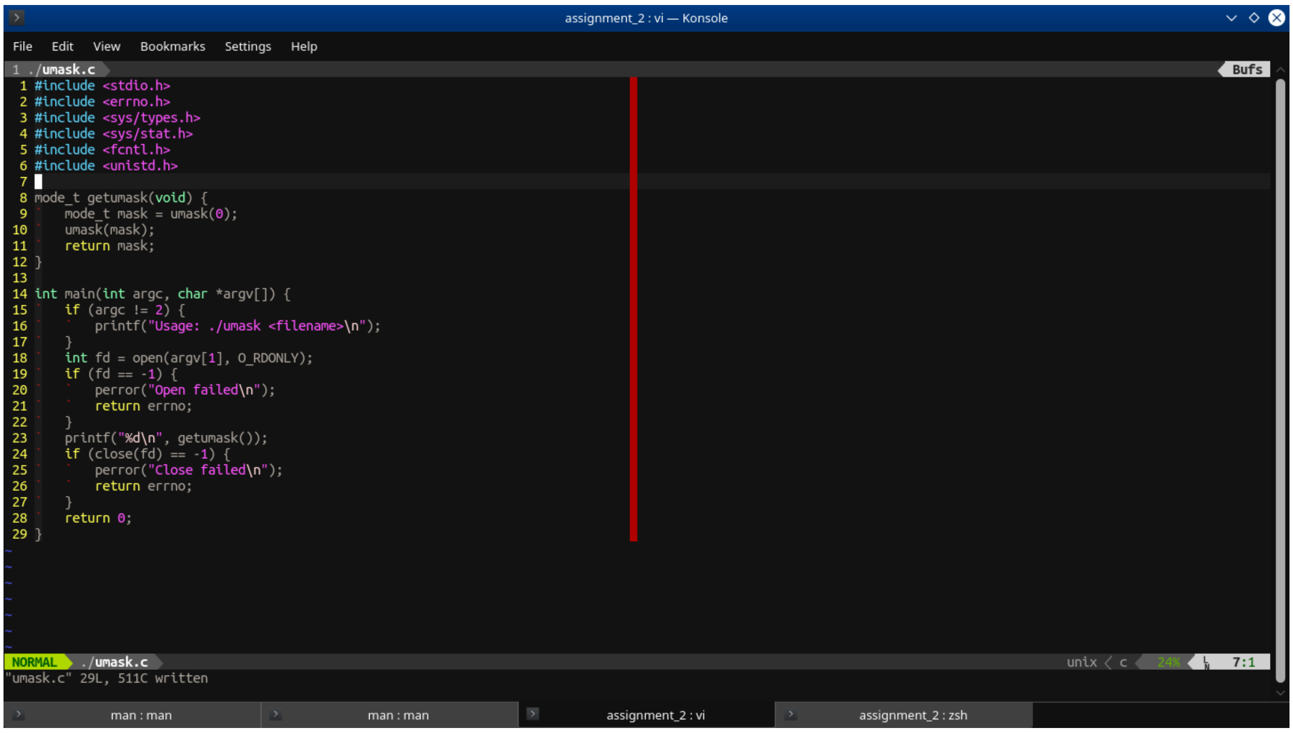
4. umask() always sets the process umask and, at the same time, returns a copy of the old umask. How can we obtain a copy of the current process umask while leaving it unchanged? Write a program to demonstrate.

umask output-



```
niramay@Niramay  ~/Documents/sem7/AUP/assignment_2  ./umask temp          ✔  10211  23:37:19
2
```

umask program-



```
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

mode_t getumask(void) {
    mode_t mask = umask(0);
    umask(mask);
    return mask;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: ./umask <filename>\n");
    }
    int fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        perror("Open failed\n");
        return errno;
    }
    printf("%d\n", getumask());
    if (close(fd) == -1) {
        perror("Close failed\n");
        return errno;
    }
    return 0;
}
~
~
~
~
~
~
~
NORMAL  ./umask.c                                                    unix < c  24%  7:1
"umask.c" 29L, 511C written
```