

Electricity Board Management System

This project helps operator (working at electricity office) to maintain records of user and request made by them. Operator will take user details and also take request details from user.

(A request may be for installing an electricity connection at newly created house, factory or Apartment.)

Operator can add user, add request, view any user and request made by him, edit request and view request. Functions have been made in separate files to keep project organised and readable. This project implements important features of C like I/O stuffs, Loop, branching, Pointers, Structures, Memory management and Functions. To run this project, create following files in a directory and run following command.

Commands

```
cd <path_to_directory>
gcc main.c application.c user.c request.c helpers.c -o application
chmod +x application
./application
```



Note gcc command will compile all files given by command line arguments, link them and create executable file specified by -o flag.

To Do

1. Make "Edit User" feature (similar to edit request), so that operator can change name or contact no. of any user.
2. Add new request status called "COMPLETED", which indicates successful completion of request.

Main-c

```

/*****
*****

Electricity board Management system
Compiled on : Linux (GCC)

*****
*****/

/* main.c */

#include <stdio.h>
#include "application.h"

int main(int argc, const char * argv[]) {

```

```
    /* Entry point of Application */

    start_application();
}
```

Application-h

```
/* Application.h */

/* We are using #ifndef macro to prevent multiple inclusion of header files */

#ifndef application_h
#define application_h

/* Maximum length of user id and request id */

#define ID_LENGTH 10

/* Maximum length of string used in this project */
/* User name, comment, address etc. */

#define STR_MAX_LENGTH 256

/* All application and user interface related functions. */

/* String which will be used to separate data of each line in given file. */

extern const char * DELIMITER;

/* Start application. */

extern void start_application(void);

/* Show welcome screen of application. */

extern void welcome(void);

/* Show menu for operator with various options. */

extern void render_menu(void);

/* Add new user who want to request for electricity. */

extern void add_user_page(void);

/* Operator can view specific user and requests done by that user. */

extern void view_user_page(void);
```

```

/* Operator can add new request which is requested by specific user. */

extern void add_request_page(void);

/* In future, if operator wants to edit request (i.e. change status or enter comments). */

extern void edit_request_page(void);

/* If operator wants to view specific request. */

extern void view_request_page(void);

/*
    Close application after operator completes his work
    input : status of application.
    input : message to display on the screen in case of erroneous exit.
*/

extern void close_application(int exit_status, const char * message);

/*
    Function which set cursor at X, Y on display
    input : X Coordinate of screen.
    input : Y Coordinate of screen.
*/

extern void gotoxy(int x, int y);

/*
    Function which display message and additional info on screen
    input : message to be displayed.
    input : additional information along with message.
*/

extern void show_dialogue_box(const char * message, char * info);

#endif /* application_h */

```

Application-c

```

/* application.c */

#include "application.h"
#include "user.h"
#include "request.h"
#include "helpers.h"

#include <stdio.h>
#include <stdlib.h>

```

```
#include <string.h>

const char * DELIMITER = ";";

void start_application(){

    /* Open user file in append mode and set 'user_fp' points to file's stream. */
    user_fp = fopen(user_file_name, "a+");

    /* Open request file in append mode and set 'request_fp' points to file's stream. */
    request_fp = fopen(request_file_name, "a+");

    /* If pointer cannot be set then warn operator about it and quit program. */
    if(user_fp == NULL || request_fp == NULL){
        printf("Something went wrong, while opening database file.\n");
        printf("Please make sure that file %s resides in current directory\n", user_file_name);
        getchar();
        return;
    }

    /* Fetch all users and setup array of all users */
    users = read_all_users_from_file(user_fp);

    /* Fetch all requests and setup array of all requests */
    requests = read_all_requests_from_file(request_fp);

    /* Show welcome page */
    welcome();
}

void welcome(){

    system("clear");

    gotoxy(25, 5);
    printf("Welcome To Electricity Board\n");
    gotoxy(27, 23);
    printf("Press Any Key to continue");

    /* Wait until user press some key. */
    getchar();

    render_menu();
}

void render_menu(){
```

```
unsigned char choice = 0;
char user_count_label[STR_MAX_LENGTH];

system("clear");

gotoxy(25, 5);
printf("Select Any option from below");
gotoxy(25, 7);
printf("1. Add new user");
gotoxy(25, 8);
printf("2. View user");
gotoxy(25, 9);
printf("3. Add new connection request");
gotoxy(25, 10);
printf("4. Edit existing request");
gotoxy(25, 11);
printf("5. View Request");
gotoxy(25, 12);
printf("6. exit");

printf(user_count_label, "Total Users : %lu and Total Requests : %lu", number_of_users,
number_of_requests);
gotoxy(0, 24);
printf("%s", user_count_label);

gotoxy(30, 15);
printf("Enter your choice : ");

gotoxy(50, 15);
scanf("%c", &choice);
flush_buffer();

switch (choice) {
    case '1':
        add_user_page();
        break;
    case '2':
        view_user_page();
        break;
    case '3':
        add_request_page();
        break;
    case '4':
        edit_request_page();
        break;
    case '5':
        view_request_page();
        break;
```

```
        case '6':
            close_application(EXIT_SUCCESS, "");
        default:
            close_application(EXIT_SUCCESS, "");
            break;
    }
}

void add_user_page(){

    /* Requests Id which will be stored in file. */
    unsigned long user_id = last_user_id + 1;

    /* User name. */
    char user_name[STR_MAX_LENGTH];

    /* User contact no. */
    char user_contact[ID_LENGTH + 2];

    /* User Id string, which will be shown in success dialogue box. */
    char user_id_string[STR_MAX_LENGTH];

    /* This variable will store user's data. */
    User user;

    /* User information in string, so it can be stored into file easily. */
    char * user_info_string = NULL;

    /* All labels which will be displayed on screen. */
    const char * enter_detail_label = "Enter details of user (All fields are required) ";
    const char * user_name_label = "Enter user name : ";
    const char * user_contact_label = "Enter user Contact : ";

    system("clear");

    gotoxy(0, 1);
    printf("%s", enter_detail_label);

    gotoxy(0, 3);
    printf("%s", user_name_label);

    gotoxy(0, 4);
    printf("%s", user_contact_label);

    /* Prepare input field for user name on row = 3, column = length(user_name_label). */
    gotoxy((int)strlen(user_name_label), 3);
```

```
scanf("%[^\\n]s", user_name);
flush_buffer();

/* Prepare input field for user contact on row = 4, column = length(user_contact_label). */

gotoxy((int)strlen(user_contact_label), 4);
scanf("%s", user_contact);
flush_buffer();

user.user_id = user_id;
user.user_name = user_name;
user.user_contact = user_contact;

/* Delimited string representation of 'user', which will be directly stored in file. */
/* 'user_info_string' is pointing to heap memory, so release it after work is done. */

user_info_string = get_user_string(&user);

/* Write new user at end of file. */

fseek(user_fp, 0, SEEK_END);
fputs(user_info_string, user_fp);
fputs("\\n", user_fp);

/* Free memory created by 'get_user_string' */
free(user_info_string);

/* free memory occupied by array of users and requests */

release_requests();
release_users();

/* Refresh all users */
users = read_all_users_from_file(user_fp);

/* Refresh all requests */
requests = read_all_requests_from_file(request_fp);

/* Prepare success dialogue */
sprintf(user_id_string, "User Id : %lu", user_id);
dialogue("User added successfully", user_id_string);
render_menu();
}

void view_user_page(){

/* User Id which operator wants to search. */
unsigned long user_id = 0;
```

```
/* User details.*/
User * user = NULL;

/* All Request made by above user */
Request ** user_requests = NULL;

/* Total count of all requests made by above user */
int user_requests_count = 0;

/* Line offset will be helpful to print information at desired location. */
int line_offset = 1;

system("clear");

printf("Enter user id : ");
scanf("%lu", &user_id);
flush_buffer();

/* Get user details based on 'user_id' */
user = get_user(user_id);

/* If no such user found, then show error */
if(user == NULL){
    system("clear");
    show_dialogue_box("No such user exists.", "");
    render_menu();
    return;
}

system("clear");

/* Show user details and requests made by him */
printf("User Details\n");

printf("\nUser Id : %lu", user->user_id);
printf("\nUser Name : %s", user->user_name);
printf("\nUser Contact : %s", user->user_contact);

/* 'user_requests' points to memory in heap, so release memory after work is done */
user_requests = get_request_of_user(user_id, &user_requests_count);

gotoxy(41,1);

if(user_requests == NULL){
    printf("Request Details\n");
    gotoxy(41, 3);
    printf("No requests made by this user");
}
```

```

        gotoxy(0, 24);
        printf("Press any key to return to main menu");

        /* Wait for user to press some key. */
        getchar();
        render_menu();
        return;
    }

    /* Show request details on right side */
    printf("Request Details");

    for (int i = 0 ; i < user_requests_count; i++) {
        line_offset += 2;
        gotoxy(41,line_offset++);
        printf("Request Id : %lu", user_requests[i]->request_id);
        gotoxy(41,line_offset++);
        printf("Request Status : %s", get_request_status_string(user_requests[i]->status));
        gotoxy(41,line_offset++);
        printf("Comments : %s", user_requests[i]->comment);
        gotoxy(41,line_offset++);
        printf("Address : %s", user_requests[i]->address);
    }

    /* releasing heap memory pointed to by 'user_requests' */
    free(user_requests);

    gotoxy(0, 24);
    printf("Press any key to return to main menu");

    /* Wait for user to press some key. */
    getchar();
    render_menu();
}

void view_request_page(){

    /* Request ID which operator wants to search. */
    unsigned long request_id = 0;

    /* Request details. */
    Request * request = NULL;

    system("clear");

    printf("Enter request id : ");
    scanf("%lu", &request_id);

```

```
flush_buffer();

/* Get request details based on 'request_id' */
request = get_request(request_id);

/* If no such request found, then show error */
if(request == NULL){
    system("clear");
    show_dialogue_box("No such request exists.", "Please contact administration.");
    render_menu();
    return;
}

system("clear");

/* Show Request details */
printf("Request Details\n");

printf("\nRequest Id : %lu", request->request_id);
printf("\nRequested By : %s (id : %lu)", request->requested_by->user_name,
    request->requested_by->user_id);
printf("\nRequest Status : %s", get_request_status_string(request->status));
printf("\nAddress : %s", request->address);
printf("\nComments : %s", request->comment);

gotoxy(0, 24);

printf("Press any key to return to main menu");

/* Wait for user to press some key. */
getchar();
render_menu();
}

void add_request_page(){

    /* Requests Id which will be stored in file. */
    unsigned long request_id = last_request_id + 1;

    /* User id of user who made request */
    unsigned long user_id = 0;

    /* Comment, if operator wants to enter one. */
    char comment[STR_MAX_LENGTH];

    /* Address of resource, where electricity lines to be installed. */
    char address[STR_MAX_LENGTH];
```

```

/* Request Id string, which will be shown in success dialogue box. */
char request_id_string[STR_MAX_LENGTH];

/* User details. */
User * user = NULL;

/* Request details. */
Request request;

/* Request information in string, so it can be stored into file easily. */
char * request_info_string = NULL;

/* All labels which will be displayed on screen. */
const char * enter_detail_label = "Enter details of Request made by user ";
const char * user_id_label = "Enter user id : ";
const char * request_comment_label = "Enter any comment (optional) : ";
const char * request_address_label = "Enter address : ";

system("clear");

gotoxy(0, 1);
printf("%s", enter_detail_label);

gotoxy(0, 3);
printf("%s", user_id_label);

gotoxy(0, 4);
printf("%s", request_comment_label);

gotoxy(0, 5);
printf("%s", request_address_label);

/* Prepare input field for user name on row = 3, column = length(user_name_label). */
gotoxy((int)strlen(user_id_label), 3);
scanf("%lu", &user_id);
flush_buffer();

/* Prepare input field for user contact on row = 4, column = length(user_contact_label). */
gotoxy((int)strlen(request_comment_label), 4);
/* Using fgets because string can be empty. */
fgets(comment, STR_MAX_LENGTH, stdin);

/* If user press just enter. (i.e. empty data). */
if(strcmp(comment, "\n") == 0){
    strcpy(comment, "");
}

```

```
    }else{
        /* If operator has entered any string, it will contain '\n' as last character. */
        /* So replace that character with NULL */
        comment[strlen(comment) - 1] = '\0';
    }

    /* Prepare input field for user contact on row = 5, column = length(user_contact_label). */

    gotoxy((int)strlen(request_address_label), 5);

    scanf("%[^\n]s", address);
    flush_buffer();

    user = get_user(user_id);

    /* If cannot find user corresponding to 'user_id' then simply redirect operator to main
    menu. */

    if(user == NULL){
        system("clear");
        show_dialogue_box("No such user exists.", "Please contact administration.");
        render_menu();
    }

    request.request_id = request_id;
    request.requested_by = user;
    request.status = PENDING;
    request.comment = comment;
    request.address = address;

    /* Delimited string representation of 'request', which will be stored in file directly. */
    /* 'request_info_string' is pointing to heap memory, so release it after work is done */

    request_info_string = get_request_string(&request);

    /* Write new request to file. */

    fseek(request_fp, 0, SEEK_END);
    fputs(request_info_string, request_fp);
    fputs("\n", request_fp);

    /* Free memory created by 'get_user_string' */

    free(request_info_string);

    /* free memory occupied by array of users and requests */
    release_requests();
    release_users();
```

```

/* Refresh all users */
users = read_all_users_from_file(user_fp);

/* Refresh all requests */
requests = read_all_requests_from_file(request_fp);

/* Prepare success dialogue */
sprintf(request_id_string, "Request Id : %lu", request_id);
show_dialogue_box("Request added successfully.", request_id_string);
render_menu();
}

void edit_request_page(){

/* Request id for request which user want to edit. */
unsigned long request_id = 0;

/* New status for request. Defaults to 'PENDING'. */
request_status new_status = PENDING;

/* New comments which user will enter. */
char new_comment[STR_MAX_LENGTH];

/* Request details. */
Request * request = NULL;

/* Labels which will be displayed on screen. */

char * enter_new_status_label = "Enter request status : ";
char * enter_new_status_suggestion_label = "(1 -> In progress, 2 -> Rejected)";
char * enter_new_comment_label = "Enter any comments : ";

system("clear");
printf("Enter Request id : ");
scanf("%lu", &request_id);
flush_buffer();

/* Fetch request from 'requests' array. */
request = get_request(request_id);

/* If cannot find request corresponding to 'requests_id' then simply redirect operator to
main menu. */
if(request == NULL){
    system("clear");
    show_dialogue_box("No such request exists.", "");
    render_menu();
}

```

```
        return;
    }

    /* Display request parameters on screen */
    printf("\nRequest Details\n");
    printf("\nRequest Id : %lu", request->request_id);
    printf("\nRequested By : %s (id : %lu)", request->requested_by->user_name, request->requested_by->user_id);
    printf("\nRequest Status : %s", get_request_status_string(request->status));
    printf("\nAddress : %s", request->address);
    printf("\nComments : %s", request->comment);

    printf("\n\n*****");

    printf("\nEnter new Details\n\n");

    printf("%s", enter_new_status_label);
    printf("\n%s", enter_new_status_suggestion_label);

    gotoxy((int)strlen(enter_new_status_label), 14);

    scanf("%d", &new_status);
    flush_buffer();

    printf("\n%s", enter_new_comment_label);
    fgets(new_comment, STR_MAX_LENGTH, stdin);

    /* If user press just enter. (i.e. empty string) */
    if(strcmp(new_comment, "\n") == 0){
        strcpy(new_comment, request->comment);
    }else{
        new_comment[strlen(new_comment) - 1] = '\0';
    }

    /* Store updated data in 'request' */
    request->status = new_status;

    /* Do not copy 'new_comment' into 'request->comment', */
    /* that requires expansion in size of 'request->comment', */
    /* if 'new_comment' is bigger than 'request->comment' in length. */
    /* Just free memory created by 'request->comment' (otherwise memory will leak.) and create new one */

    free(request->comment);
    request->comment = strdup(new_comment);

    /* Replace Data in file */
```

```

        replace_request_in_file(request_fp, request);

        /* free memory occupied by array of users and requests */
        release_requests();
        release_users();

        /* Refresh all users */
        users = read_all_users_from_file(user_fp);

        /* Refresh all requests */
        requests = read_all_requests_from_file(request_fp);

        show_dialogue_box("Request updated successfully.", "");
        render_menu();
    }

void close_application(int status, const char * message){
    fclose(user_fp);
    fclose(request_fp);
    release_users();
    release_requests();
    system("clear");
    printf("%s", message);
    exit(status);
}

void show_dialogue_box(const char * message, char * id){

    int message_length = (int)strlen(message);

    const char * press_any_key = "Press any key to continue";
    int info_length = (int)strlen(press_any_key);

    int left_offset = 15;
    int length = 50;
    int line_offset = 9;

    system("clear");

    /* Line 1 */
    gotoxy(left_offset, line_offset++);
    printf("*****");

    /* Line 2 */
    gotoxy(left_offset, line_offset);
    printf("");

```

```
gotoxy(left_offset + length - 1, line_offset++);
printf("**");

/* Line 3 */
gotoxy(left_offset, line_offset);
printf("**");
gotoxy((80 - message_length)/2, line_offset);
printf("%s", message);
gotoxy(left_offset + length - 1, line_offset++);
printf("**");

if(strcmp(id, "") != 0){
    /* Line 4 */
    gotoxy(left_offset, line_offset);
    printf("**");
    gotoxy((int)((80 - strlen(id))/2), line_offset);
    printf("%s", id);
    gotoxy(left_offset + length - 1, line_offset++);
    printf("**");
}

/* Line 5 */
gotoxy(left_offset, line_offset);
printf("**");
gotoxy((int)((80 - info_length)/2), line_offset);
printf("%s", press_any_key);
gotoxy(left_offset + length - 1, line_offset++);
printf("**");

/* Line 6 */
gotoxy(left_offset, line_offset);
printf("**");
gotoxy(left_offset + length - 1, line_offset++);
printf("**");

/* Line 7 */
gotoxy(left_offset, line_offset++);
printf("*****");

getchar();

}

void gotoxy(int x,int y){
    printf("%c[%d;%df",0x1B,y,x);
}
```

User-h

```
/* user.h */

/* We are using #ifndef macro to prevent multiple inclusion of header files */

#ifndef user_h
#define user_h

#include <stdio.h>

/* Structure which stores data of user. */

typedef struct User {

    /* Unique identifier associated with every user. */
    unsigned long user_id;

    /* User name which is of type string aka 'char *'. */
    char * user_name;

    /* User contact no. is of type string aka 'char *'. */
    char * user_contact;

} User;

/* User file name in which all user's data will be stored. */
extern const char * user_file_name;

/* Global variable which contains all user's data in memory. */
extern User ** users;

/* Global variable which stores total count of users. */
extern unsigned long number_of_users;

/* Global File pointer, so we can access user file in any function. */
extern FILE * user_fp;

/* Variable which keeps track of last inserted user id. */
extern unsigned long last_user_id;

/*
    Function which give 'User' based on their 'user_id'.
    input  : user id.
    output : Pointer to 'User' which contains user's information in structure format.
*/

extern User * get_user ( unsigned long user_id );
```

```
/*
    Function which converts 'User' data into string, so that we can store this o/p in file.
    input  : User information in structure format.
    output : comma separated string representation of 'user_info'.
*/

char * get_user_string ( User * user );

/*
    Function which converts string (which we read from i/p file) into 'User' data.
    input  : comma separated string which contains user's information.
    output : Pointer to 'User' which contains all user's information in structure format.
*/

User * get_user_info ( char * user_info );

/*
    Function which reads all lines from file, converts and returns array of all users.
    input  : FILE pointer of user file.
    output : Pointer to array of all users.
*/

User ** read_all_users_from_file ( FILE * fp );

/*
    Function which replace specific user in file.
    input  : FILE pointer of user file.
    input  : User to be replaced.
*/

void replace_user_in_file ( FILE * fp , User * user );

/* Function which releases memory created by users (in heap). */

void release_users ( void );

#endif /* user_h */
```

User-c

```
/* user.c */

#include "user.h"
#include "helpers.h"
#include "application.h"

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

```

#include <string.h>

const char * user_file_name = "users.txt";

User ** users = NULL;

unsigned long last_user_id = 0;

FILE * user_fp = NULL;

unsigned long number_of_users = 0;

User * get_user ( unsigned long user_id ){
    for (int i = 0 ; i < number_of_users; i++) {
        if(users[i]->user_id == user_id){
            return users[i];
        }
    }

    return NULL;
}

char * get_user_string(User * user_info){

    char user_id_string[ID_LENGTH];

    char * user_info_string = (char *)malloc(sizeof(char) * STR_MAX_LENGTH);

    /* Convert user id into string. */
    sprintf(user_id_string, "%lu", user_info->user_id);

    /* Conversion of multiple strings into single delimited string. */
    strcpy(user_info_string, strcat(user_id_string, DELIMITER));
    user_info_string = strcat(user_info_string, user_info->user_name);
    user_info_string = strcat(user_info_string, DELIMITER);
    user_info_string = strcat(user_info_string, user_info->user_contact);

    /* We need to release memory created by 'user_info_string'. */
    return user_info_string;
}

User * get_user_info ( char * user_info ){

    char * token = NULL;

```

```
/* Set 'COLUMN_TAG' to 0 so that we can read new values. */
/* 'COLUMN_TAG' will help us to track which column we are going to read. */

int COLUMN_TAG = 0;

/* Keep copy of original 'user_info' because this string will be modified. */

char * user_info_copy = strdup(user_info);

User * read_user = (User *) malloc(sizeof(User));

while((token = tokenize_string(user_info_copy, DELIMITER, &user_info_copy)) != NULL){

    switch (COLUMN_TAG) {
        case 0:
            read_user->user_id = atoi(token);
            COLUMN_TAG++;
            free(token);
            break;
        case 1:
            read_user->user_name = strdup(token);
            COLUMN_TAG++;
            free(token);
            break;
        case 2:
            read_user->user_contact = strdup(token);
            COLUMN_TAG++;
            free(token);
            break;
        default:
            close_application(EXIT_FAILURE, "Error while parsing user database.....");
            break;
    }
}

free(user_info_copy);

/* Check if we have read anything or not. */

if(COLUMN_TAG != 3){
    /* If not then just release memory created by 'read_user'. */
    free(read_user);
}

return read_user;
}
```

```
User ** read_all_users_from_file( FILE * fp ){

    /* All users (pointer to 'User') will be stored in this array. */
    User ** all_users = NULL;

    /* Read each line from file and prepare user detail. */
    User * read_user = NULL;

    /* Total number of users. */
    unsigned long user_count = 0;

    /* Read line from file and store in this variable. */
    char * user_info = (char *)malloc(sizeof(char) * STR_MAX_LENGTH);

    /* Set file pointer to beginning of file. */
    rewind(fp);

    /* Start reading file. */

    while(fgets(user_info, STR_MAX_LENGTH, fp) != NULL){

        if(strcmp(user_info, "\n") == 0)
            continue;

        /* Get User structure from string, which we have read from file */

        /* We do not need to release memory occupied by 'read_user' because, */
        /* We are going to store address of this variable into 'all_users' array, */
        /* which will be released at end of the program. */

        read_user = get_user_info(user_info);

        /* If we cannot create user successfully, then just continue to next cycle. */
        if(read_user == NULL){
            continue;
        }

        /* Allocate memory for 'all_users' array. */
        all_users = (User **)realloc(all_users, sizeof(User *) * (1 + user_count));

        /* Store address of user into 'all_users' array. */
        all_users[user_count++] = read_user;

    }

    /* Release memory occupied by 'user_info'. */
```

```
    free(user_info);

    /* Keep track of number of users in global variable. */
    last_user_id = number_of_users = user_count;

    return all_users;
}

void replace_user_in_file (FILE * fp, User * updated_user){

    User * read_user = NULL;

    /* below function can leak memory, so clear memory before returning from this function. */
    char * updated_user_string = get_user_string(updated_user);

    char line[STR_MAX_LENGTH];

    /* Open temp file in write mode. */
    FILE * write_fp = fopen("temp.txt", "w");

    /* Set file pointer to beginning of file. */
    rewind(fp);

    while ((fgets(line, STR_MAX_LENGTH, fp)) != NULL){

        /* Create user from string, which we have read from file. */
        /* Memory created by below function will be in heap. */
        /* So release that memory before loop ends. */

        if((read_user = get_user_info(line)) == NULL){
            continue;
        }

        if(read_user->user_id == updated_user->user_id){

            /* If user id matches, then replace line with 'updated_user_string'. */
            fputs(updated_user_string, write_fp);

            /* As 'updated_user_string' does not contain "\n", add one manually. */
            fputs("\n", write_fp);

        }else{
            /* Write 'line' as it is in file. */
            /* Do not need to put extra "\n", as 'line' already contains "\n". */
            fputs(line, write_fp);
        }
    }
}
```

```

        free(read_user);
    }

    /* Close temp file. */
    fclose(write_fp);

    /* Close original file. */
    fclose(fp);

    /* Remove original file from disk. */
    remove(user_file_name);

    /* Rename temporary file with original file name. */
    rename("temp.txt", user_file_name);

    /* Open that file in 'a+' mode so we have reached state here we left off. */
    user_fp = fopen(user_file_name, "a+");

    free(updated_user_string);
}

void release_users(){
    if(!users){
        return;
    }
    for(int i = 0 ; i < number_of_users ; i++){
        free(users[i]->user_name);
        free(users[i]->user_contact);
    }
    free(users);
    users = NULL;
}

```

Request-h

```

/* request.h */

/* We are using #ifndef macro to prevent multiple inclusion of header files */

#ifndef request_h
#define request_h

#include "user.h"
#include <stdio.h>

/* Type of request status can be 'Pending', 'In process' or 'Rejected' */

```

```
typedef enum request_status {
    PENDING,
    IN_PROCESS,
    REJECTED
} request_status;

/* Function will return string based on 'request_status' */

char * get_request_status_string(request_status);

/* Structure which stores data of user */

typedef struct Request{

    /* Unique identifier associated with every request */
    unsigned long request_id;

    /* User by which connection request was created */
    User * requested_by;

    /* Current status of request */
    request_status status;

    /* Comments on request. If operator decides to reject this request, */
    /* he can optionally add reason for rejection in comments. */
    char * comment;

    /* Address at which electricity needs to be installed. */
    char * address;

} Request;

/* Request file name in which data of all requests will be stored. */
extern const char * request_file_name;

/* Global variable which contains data of all requests. */
extern Request ** requests;

/* Global variable which stores total count of all requests. */
extern unsigned long number_of_requests;

/* Global File pointers, so we can access it in any of the functions */
extern FILE * request_fp;

/* Variable which keeps track of last inserted request id. */
```

```

extern unsigned long last_request_id;

/*
    Function which give 'Request' based on their 'request_id'.
    input  : request id.
    output : Pointer to 'User' which contains request information in structure format.
*/

extern Request * get_request ( unsigned long request_id );

/*
    Function which converts 'Request' data into string, so that we can store this o/p in file.
    input  : Request information in structure format.
    output : comma separated string representation of 'request_info'.
*/

char * get_request_string ( Request * request );

/*
    Function which converts string (which we read from i/p file) into 'Request' data.
    input  : comma separated string which contains request information.
    output : Pointer to 'Request' which contains request information in structure format.
*/

Request * get_request_info ( char * request_info_string );

/*
    Function which reads all lines from file, converts and returns array of all requests.
    input  : FILE pointer of user file.
    output : Pointer to array of all requests.
*/

Request ** read_all_requests_from_file ( FILE * fp );

/*
    Function which replace specific request in file
    input  : FILE pointer of request file.
    input  : Request to be replaced.
*/

void replace_request_in_file ( FILE * fp , Request * request );

/*
    Function which returns all Request details, for given user id
    input  : user id.
    input  : address of variable which contains number of request.
    output : Pointer to array of all requests of specified user.
*/

```

```
*/

Request ** get_request_of_user(unsigned long user_id, int * request_count);

/* Function which releases memory created by requests (in heap.) */

void release_requests(void);

#endif /* request_h */
```

Request-c

```
/* request.c */

#include "request.h"
#include "helpers.h"
#include "application.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

Request ** requests = NULL;

const char * request_file_name = "requests.txt";

unsigned long number_of_requests = 0;

FILE * request_fp = NULL;

unsigned long last_request_id = 0;

Request * get_request(unsigned long request_id){
    for (int i = 0; i < number_of_requests; i++) {
        if(requests[i]->request_id == request_id){
            return requests[i];
        }
    }

    return NULL;
}

char * get_request_string(Request * request_info){

    char request_id_string[ID_LENGTH], user_id_string[ID_LENGTH], request_status[2];

    char * request_info_string = (char *)malloc(sizeof(char) * STR_MAX_LENGTH);
```

```

    if(request_info->requested_by == NULL){
        /* If we can not get who has made request, then just show message and quit program */
        close_application(EXIT_FAILURE, "Some request does not have user.");
    }

    /* sprintf will be used to convert integer into string. */

    sprintf(request_id_string, "%lu", request_info->request_id);
    sprintf(user_id_string, "%lu", request_info->requested_by->user_id);
    sprintf(request_status, "%d", request_info->status);

    /* Conversion of multiple strings into single delimited string. */

    strcpy(request_info_string, strcat(request_id_string, DELIMITER));
    request_info_string = strcat(request_info_string, user_id_string);
    request_info_string = strcat(request_info_string, DELIMITER);
    request_info_string = strcat(request_info_string, request_status);
    request_info_string = strcat(request_info_string, DELIMITER);
    request_info_string = strcat(request_info_string, request_info->comment);
    request_info_string = strcat(request_info_string, DELIMITER);
    request_info_string = strcat(request_info_string, request_info->address);

    /* We need to release memory created by 'user_info_string'. */
    return request_info_string;
}

Request * get_request_info ( char * request_info ){

    char * token = NULL;

    /* Set 'COLUMN_TAG' to 0 so that we can read new values. */
    /* 'COLUMN_TAG' will help us to track, which column we are going to read. */

    int COLUMN_TAG = 0;

    /* Keep copy of original 'request_info' because this string will be modified. */
    /* Need to free memory pointed to by 'request_info_copy'. */

    char * request_info_copy = strdup(request_info);

    Request * read_request = (Request *) malloc(sizeof(Request));

    User * user = NULL;

    while((token = tokenize_string(request_info_copy, DELIMITER, &request_info_copy)) != NULL){

        switch (COLUMN_TAG) {

```

```
        case 0:
            read_request->request_id = atoi(token);
            COLUMN_TAG++;
            free(token);
            break;
        case 1:
            user = get_user(atoi(token));
            free(token);
            /* It is not necessary, that user mentioned by request does exists */
            if(!user){
                free(read_request);
                free(request_info_copy);
                return NULL;
            }
            read_request->requested_by = user;
            COLUMN_TAG++;
            break;
        case 2:
            read_request->status = atoi(token);
            COLUMN_TAG++;
            free(token);
            break;
        case 3:
            read_request->comment = strdup(token);
            COLUMN_TAG++;
            free(token);
            break;
        case 4:
            read_request->address = strdup(token);
            COLUMN_TAG++;
            free(token);
            break;
        default:
            close_application(EXIT_FAILURE, "Error while parsing user database.....");
            break;
    }
}

free(request_info_copy);

/* Check if we have read everything or not */

if(COLUMN_TAG != 5){
    /* If not then just release memory created by 'read_request' */
    free(read_request);
}
```

```

        return read_request;
    }

Request ** read_all_requests_from_file( FILE * fp ){

    /* All requests (pointer to 'Request') will be stored in this array. */
    Request ** all_requests = NULL;

    /* Read each line from file and prepare request detail. */
    Request * read_request = NULL;

    /* Total number of requests. */
    unsigned long request_count = 0;

    /* Read line from file and store in this variable. */
    char * request_info = (char *)malloc(sizeof(char) * STR_MAX_LENGTH);

    /* Set file pointer to beginning of file. */
    rewind(fp);

    /* Start reading file. */

    while(fgets(request_info, STR_MAX_LENGTH, fp) != NULL){

        if(strcmp(request_info, "\n") == 0)
            continue;

        /* Get Request structure from string, which we have read from file */

        /* We do not need to release memory occupied by 'read_request' because, */
        /* We are going to store address of this variable into 'all_requests' array, */
        /* which will be released at end of the program. */

        read_request = get_request_info(request_info);

        /* If we can not create request successfully, then just continue to next cycle. */
        if(read_request == NULL){
            continue;
        }

        /* Allocate memory for 'all_requests' array. */
        all_requests = (Request **)realloc(all_requests, sizeof(Request *) * (1 + request_count));

        /* Store address of request into 'all_requests' array. */
        all_requests[request_count++] = read_request;
    }
}

```

```
    }

    /* Release memory occupied by 'request_info'. */
    free(request_info);

    /* Keep track of number of users in global variable. */
    last_request_id = number_of_requests = request_count;

    return all_requests;
}

Request ** get_request_of_user(unsigned long user_id, int * request_count){

    Request ** user_requests = NULL;
    int count = 0;

    for(int i = 0 ; i < number_of_requests ; i++){
        if(requests[i]->requested_by->user_id == user_id){
            user_requests = (Request **)realloc(user_requests, sizeof(Request *) * (1 + count));
            user_requests[count++] = requests[i];
        }
    }
    /* Modify count. */
    *request_count = count;

    return user_requests;
}

void replace_request_in_file (FILE * fp, Request * updated_request){

    Request * read_request = NULL;

    /* below function can leak memory, so clear memory before returning from this function. */
    char * updated_request_string = get_request_string(updated_request);

    char line[STR_MAX_LENGTH];

    /* Open temp file in write mode. */
    FILE * write_fp = fopen("temp.txt", "w");

    rewind(fp);

    while ((fgets(line, STR_MAX_LENGTH, fp)) != NULL){

        /* Create request from string, which we have read from file. */
```

```

    /* Memory created by below function will be in heap. */
    /* So release that memory before loop ends. */

    if((read_request = get_request_info(line)) == NULL){
        continue;
    }

    if(read_request->request_id == updated_request->request_id){

        /* If request id matches, then replace line with 'updated_request_string'. */
        fputs(updated_request_string, write_fp);

        /* As 'updated_request_string' does not contain "\n", add one manually. */
        fputs("\n", write_fp);

    }else{
        /* Write 'line' as it is in file. */
        /* Do not need to put extra "\n", as 'line' already contains "\n". */
        fputs(line, write_fp);
    }

    free(read_request);
}

/* Close temp file. */
fclose(write_fp);

/* Close original file. */
fclose(fp);

/* Remove original file from disk. */
remove(request_file_name);

/* Rename temporary file with original file name. */
rename("temp.txt", request_file_name);

/* Open that file in 'a+' mode so we have reached state here we left of. */
request_fp = fopen(request_file_name, "a+");

free(updated_request_string);
}

char * get_request_status_string(request_status status){

    switch (status) {

```

```
        case PENDING:
            return "Pending";
        case IN_PROCESS:
            return "In Progress";
        case REJECTED:
            return "Rejected";
        default:
            break;
    }

    return "Pending";
}

void release_requests(){
    if(!requests){
        return;
    }
    for(int i = 0 ; i < number_of_requests ; i++){
        free(requests[i]->comment);
        free(requests[i]->address);
        requests[i]->requested_by = NULL;
    }

    free(requests);
    requests = NULL;
}
```

Helpers-h

```
/* helper.h */

#ifndef helpers_h
#define helpers_h

/*
    Flush buffer after user enter some input.
    If we won't flush buffer then 'enter' pressed by user will be stored in buffer,
    and it will be used as an input to next I/O function, which may cause issue.
*/

extern void flush_buffer(void);

/*
    Function which returns single string delimited by 'delimiter' from 'source'.
    Also we want to modify 'source' after reading token so we have used 'context' (char **),
    which is pointer to 'source', which will modify input 'source'
*/
```

```

    input : Delimited string.
    input : Delimiter by which string is separated.
    input : context i.e. address of 'string' (char *), so we have used char ** to catch
           address of string.
    output : Single token separated by 'delimiter'.
*/

```

```
extern char * tokenize_string(char * source, char const * delimiter, char ** context);
```

```
#endif /* helpers_h */
```

Helpers-c

```
/* helpers.c */
```

```
#include "helpers.h"
#include "application.h"
```

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

```
char * tokenize_string(char * source, char const * delimiter, char ** context){
```

```
    /* Example 'source' string is "A;B;C\n\0". */
    /* '\n' will be at end of the 'source' as we are reading from file. */

```

```
    if(source == NULL){
        return NULL;
    }

```

```
    /* String after 1st 'delimiter' occurs will be stored in this variable (";B;C\n\0"). */
    char * rest_string = strpbrk(source, delimiter);

```

```
    /* Creating a copy of above variable, as it will get modified */
    char copy_rest_string[STR_MAX_LENGTH];

```

```
    /* Token is string before 'delimiter' ("A\n\0") */
    char * token = NULL;

```

```
    /* 'rest_string' will be NULL when there is not any 'delimiter' (example string "C\n\0") */

```

```
    if(rest_string != NULL){
```

```
        /* Make copy of 'rest_string' so 'copy_rest_string' will be ";B;C\n\0" */
        strcpy(copy_rest_string, rest_string);

```

```
        /* 1st character of 'rest_string' is our 'delimiter' so we need to replace it by "\0" */

```

```

    *rest_string = 0;

    /* With above operation 'source' string will also get modified */
    /* and becomes "A\0B;C\n\0" i.e. "A\0" (string in C language is NULL terminated) */

    /* Create copy of modified 'source' string which is "A\0" */
    /* As this copy is created with 'strdup', memory in heap will be created */
    /* which we will clear later . */

    token = strdup(source);

    /* '*context' is pointing to 'source' */
    /* i.e. we are clearing memory pointed to by 'source' */
    /* Otherwise memory will be leaked as, 'source' string is getting modified */

    free(*context);

    /* point '*context' (i.e. 'source') to copied string which is ";B;C\0" */
    /* Notice +1 here, this is because we want to skip 1st 'delimiter' which is at 1st
character of 'copy_rest_string' */
    /* Do not need worry about memory created by below operation */
    /* because in next iteration, it will be freed */

    *context = strdup(copy_rest_string + 1);

} else {

    /* If we can not find any 'delimiter' then 'source' should be like "C\n\0" */
    /* So first replace last "\n" with "\0" */

    source[strlen(source) - 1] = '\0';

    /* Create token from 'source' which is now "C\0\0" i.e. "C\0" */
    token = strdup(source);

    /* Free memory pointed to by '*context' */
    free(*context);

    /* And set '*context' i.e. 'source' to NULL so that we can break loop (exit condition) */

    *context = NULL;
}

return token;
}

void flush_buffer(){
    while (getchar() != '\n');
}

```



Fig. 1 Welcome screen



Fig. 2 Main menu

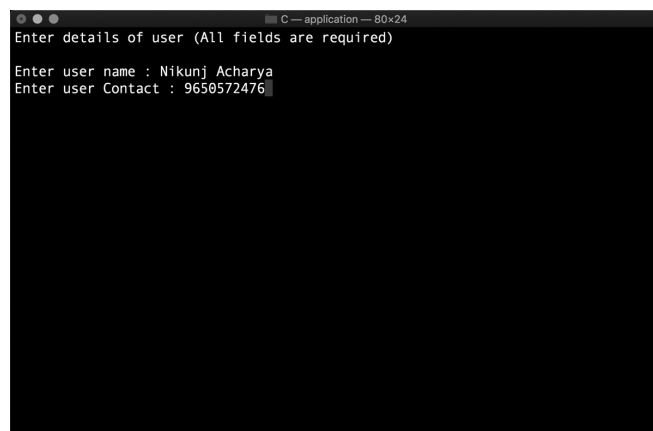


Fig. 3 Add user



Fig. 4 User success dialogue

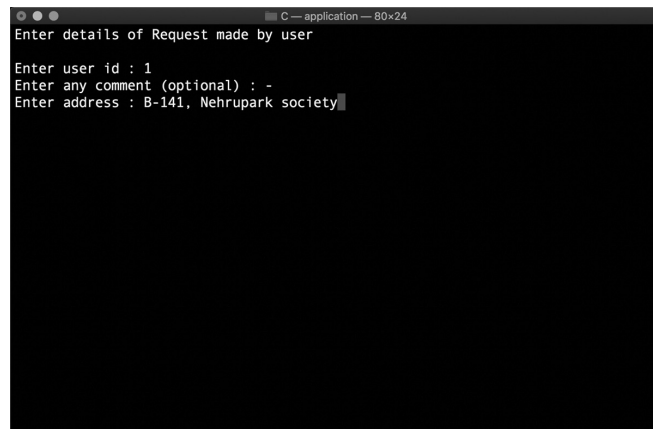


Fig. 5 Add request



Fig. 6 Request success dialogue

```
C — application — 80x24
User Details                                Request Details
User Id : 1                                Request Id : 1
User Name : Nikunj Acharya                 Request Status : Pending
User Contact : 9650572476                  Comments : -
                                           address : B-141, Nehrupark society

Press any key to return to main menu
```

Fig. 7 View user

```
C — application — 80x24
Enter Request id : 1
Request Details
Request Id : 1
Requested By : Nikunj Acharya (id : 1)
Request Status : Pending
Address : B-141, Nehrupark society
Comments : -
*****
Enter new Details
Enter request status : 2
(1 -> In progress, 2 -> Rejected)
Enter any comments : Approval required.
```

Fig. 8 Edit request

```
C — application — 80x24
Request Details
Request Id : 1
Requested By : Nikunj Acharya (id : 1)
Request Status : Rejected
Address : B-141, Nehrupark society
Comments : Approval required.

Press any key to return to main menu
```

Fig. 9 View request