```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>


/*
 * DBFILENAME Ã© o nome do arquivo com os dados dos estabelecimentos.
 *
 * O arquivo guarda na primeira linha o nÃºmero de estabelecimentos existentes
 * e nas linhas seguintes os dados de cada estabelecimento, um por linha,
 * separado por ponto-e-vÃ-rgula. O arquivo deve ter uma linha em branco no
 * final.
 *
 * Os dados sÃ£o:
 *      id;x;y;categoria;nome;endereÃ§o;pontuacao;votos
 * onde:
 *      - 'id' Ã© um nÃºmero de quatro dÃ-gitos;
 *      - 'x' Ã© a coordenada x (0-1000);
 *      - 'y' Ã© a coordenada y (0-1000);
 *      - 'categoria' Ã© uma string com a categoria do estabelecimento;
 *      - 'nome' Ã© uma string com o nome do estabelecimento;
 *      - 'endereco' Ã© uma string com o endereÃ§o;
 *      - 'pontuacao' Ã© a soma das notas das ao estabelecimento;
 *  - 'votos' Ã© a quantidade de notas recebidas.
 *
 * As strings tem comprimento mÃ¡ximo de 256 caracteres.
 */
#define DBFILENAME "db.csv"
/* os limites das coordenaras de posicionamento */
#define POSLIMMAX 1000
#define POSLIMMIN 0
/* tamanho do backlog da funÃ§Ã£o listen */
#define BACKLOG 100
/* nÃºmero mÃ¡ximo de estabelecimentos na lista */
#define ITEMLIM 8000
/* tamanho mÃ¡ximo das strings usadas */
#define BUFLEN 1024
#define MAXSTRLEN 256



/* === Tipos auxiliares ================================================= */

typedef enum
{
        FALSE = 0,
        TRUE = 1
} bool_t;



/* === Estrutura de dados dos estabelecimentos ========================== */

typedef struct item {
        int id;
        int posx;
        int posy;
        char categoria[MAXSTRLEN];
        char nome[MAXSTRLEN];
        char endereco[MAXSTRLEN];
```



```c
        int pontuacao;
        int votos;
} item_t;

typedef struct res_busca {
        int id;
        char nome[MAXSTRLEN];
} res_busca_t;



/* === Ferramentas para tratamento de erros ============================= */

typedef enum {
    NO_ERROR = 0,
    USAGE_ERROR,
    PORT_OUT_RANGE,
    DB_FEWER_ROWS,
    BIND_ERROR,
    BUSCA_NOT_SET,
    MYERROR_LIM

} my_error_t;

const char* my_error_desc[] =
{
    "",
    "usage: ./servidor_info_tcp <port number>",
    "error: port number must be between 1 and 8000",
    "error: database constains less rows than specified",
    "error: failed to bind",
    "error: vector 'busca' is not set",
    ""
};

void pMyError(my_error_t e, const char *function)
{
    if(NO_ERROR < e && e < MYERROR_LIM) {
        fprintf(stderr, "%s: %s.\n", function, my_error_desc[e]);
    }
}



/* === Estruturas de dados com as informaÃ§Ãµes dos estabelecimentos ======== */

static item_t *lista = NULL;
static int lista_len = 0;
static res_busca_t *busca = NULL;
static int cli_posx = -1;       // -1 significa posiÃ§Ã£o do cliente
static int cli_posy = -1;       // nÃ£o configurada ainda



/* ========================================================================
 * === Pragmas das funÃ§Ãµes ================================================
 * ======================================================================== */

/* === FunÃ§Ãµes de processamento de informaÃ§Ãµes ============================
 */
void readItemInfo(item_t *item, char *buf);
void readDB(void);
void writeDB(void);

/* === FunÃ§Ãµes de busca e impressÃ£o de informaÃ§Ãµes ======================
 */
int buscar(const int x, const int y, char *categoria);
void sendBusca(int N, int S);
void sendInfoID(int id, int S);
```

```
void sendCategorias(int S);
void votarID(int id, int nota, int S);

/* === FunÃ§Ãµes auxiliares de networking ================================= */
int bindTCP(char *port);
void serverReady(int S);

/* === Interpretados de comandos ======================================== */
int interpretador(char *cmd, int S);



/* ========================================================================
 * === MAIN ===============================================================
 * ======================================================================== */

int main(int argc, char *argv[])
{
        /* variÃ¡veis para criaÃ§Ã£o de conexÃµes */
        int listen_socket, connect_sock;
        struct sockaddr_storage remote_st;
        socklen_t st_len;
        /* variÃ¡veis das informaÃ§Ãµes do socket do cliente */
        struct sockaddr_in si;
        socklen_t si_len;
        /* variÃ¡veis para fork de processos */
        struct sigaction sa;
        pid_t pid;

        char buf[BUFLEN];
        int len;
        int i;
        int cmd = 0;

        /* verificando argumentos */
        if (argc < 2) {
                pMyError(USAGE_ERROR, __func__);
                exit(EXIT_FAILURE);
        }

        /* verificaÃ§Ã£o da porta usada: a porta deve ser well known */
        i = atoi(argv[1]);
        if (0 >= i && i > 8000) {
                pMyError(PORT_OUT_RANGE, __func__);
                exit(EXIT_FAILURE);
        }

        /* espera por conexÃµes no listen_socket */
        listen_socket = bindTCP(argv[1]);
        if (listen(listen_socket, BACKLOG) == -1) {
                perror("listen");
                exit(EXIT_FAILURE);
        }

        /* configurando o handler para encerrar os procesos-zumbis */
    sa.sa_handler = SIG_DFL;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_NOCLDWAIT;
    if (sigaction(SIGCHLD, &sa, NULL) == -1) {
        perror("sigaction");
        exit(1);
    }

    while(1) {

        /* aceita a conexÃ£o de um cliente por um socket novo connect_sock */
                if ((connect_sock = accept(listen_socket, (struct sockaddr *)&re
mote_st, &st_len)) < 0) {
                        perror("accept");
```



```
                        exit(EXIT_FAILURE);
                }

                /* faz o fork do processo */
                pid = fork();
                if (pid < 0) {
                        perror("fork");
                        exit(EXIT_FAILURE);
                }

                /* caso o processo seja o filho */
                if (pid == 0) {

                        /* fecha o socket que estÃ¡ esperando por novos clientes
 */
                        close(listen_socket);

                        /* coleta as informaÃ§Ãµes do socket e imprime na stdout
 */
                        si_len = sizeof(si);
                        if (getpeername(connect_sock, (struct sockaddr *)&si, &s
i_len) < 0) {
                                perror("getpeername");
                                close(connect_sock);
                                exit(EXIT_FAILURE);
                        }
                        inet_ntop(AF_INET, &(si.sin_addr), buf, INET_ADDRSTRLEN)
;
                        printf("IP address: %s; Port number: %d\n", buf, ntohs(si.sin_por
t));

                        /* main loop */
                        while (len = recv(connect_sock, buf, sizeof(buf), 0)) {

                                /* chamada do interpretador de comandos */
                                cmd = interpretador(buf, connect_sock);

                                /* encerra o processo filho por ordem do cliente
 */
                                if (cmd == -1) {
                                        return 0;
                                }

                                serverReady(connect_sock);
                        }

                        close(connect_sock);
                }
                /* processo pai fecha sua cÃ³pia do socket do cliente */
                else {
                        close(connect_sock);
                }

        }

        return 0;
}



/* ========================================================================
 * === ImplementaÃ§Ã£o das funÃ§Ãµes =========================================
 * ======================================================================== */

/* === FunÃ§Ãµes de processamento de informaÃ§Ãµes ============================
*/

void readItemInfo(item_t *item, char *buf)
/*
```

```
 * desc         :       Preenche a estrutura de dados item com as informaÃ§Ãµes
contidas
 *                               no buffer.
 *
 * params       :       1.      Estrutura de dados de saÃ-da.
 *                              2.      Buffer para a entrada de informaÃ§Ãµes.
 *
 * output       :       Nenhuma.
 */
{
        int i = 0;
        char *tok;

        tok = strtok(buf,";");
        while (tok != NULL) {
                switch (i) {
                        case 0:
                                item->id = atoi(tok);
                                break;
                        case 1:
                                item->posx = atoi(tok);
                                break;
                        case 2:
                                item->posy = atoi(tok);
                                break;
                        case 3:
                                strncpy(item->categoria, tok, MAXSTRLEN);
                                break;
                        case 4:
                                strncpy(item->nome, tok, MAXSTRLEN);
                                break;
                        case 5:
                                strncpy(item->endereco, tok, MAXSTRLEN);
                                break;
                        case 6:
                                item->pontuacao = atoi(tok);
                                break;
                        case 7:
                                item->votos = atoi(tok);
                                break;
                }
                i++;
                tok = strtok(NULL,";");
        }
}

void readDB(void)
/*
 * desc         :       LÃª as informaÃ§Ãµes de todos os estabelecimentos para u
m vetor
 *                              global (lista) na memÃ³ria. A entrada Ã© um arqu
ivo de texto
 *                              DBFILENAME (cujo formato Ã© dado junto a "#defin
e DBFILENAME").
 *
 * params       :       Nenhum.
 *
 * output       :       Nenhuma.
 */
{
        FILE *db;
        char buf[BUFLEN];
        int i;

        db = fopen (DBFILENAME,"r");
        if (db == NULL) {
                perror("failed to open the database");
                exit(EXIT_FAILURE);
        }
```



```
        /* lÃª o nÃºmero de estabelecimentos */
        fgets(buf, BUFLEN, db);
        lista_len = atoi(buf);

        /* cria a lista de estabelecimentos */
        lista = calloc(lista_len, sizeof(item_t));

        /* lÃª as informaÃ§Ãµes de cada estabelecimento */
        i = lista_len;
        while (!feof(db) && i > 0) {
                fgets(buf, BUFLEN, db);
                readItemInfo(&lista[lista_len - (i--)], buf);
        }
        /* tratamento de erro para quando temos menos estabelecimentos do que N
*/
        if (i > 0) {
                pMyError(DB_FEWER_ROWS, __func__);
                exit(EXIT_FAILURE);
        }

        fclose(db);
}

void writeDB(void)
/*
 * desc         :       Escreve as informaÃ§Ãµes de todos os estabelecimentos pr
esentes
 *                              na memÃ³ria em um arquivo DBFILENAME.
 *
 * params       :       Nenhum.
 *
 * output       :       Nenhuma.
 */
{
        FILE *db;
        char buf[BUFLEN];
        int i;

        db = fopen (DBFILENAME,"w");
        if (db == NULL) {
                perror("failed to write to the database");
                exit(EXIT_FAILURE);
        }

        fprintf(db, "%d\n", lista_len);

        for (i = 0; i < lista_len; i++) {
                fprintf(db, "%.4d;", lista[i].id);
                fprintf(db, "%d;%d;", lista[i].posx, lista[i].posy);
                fprintf(db, "%s;", lista[i].categoria);
                fprintf(db, "%s;", lista[i].nome);
                fprintf(db, "%s;", lista[i].endereco);
                fprintf(db, "%d;", lista[i].pontuacao);
                fprintf(db, "%d\n", lista[i].votos);

        }

        fprintf(db, "\n");

        fclose(db);
}



/* === FunÃ§Ãµes de busca e impressÃ£o de informaÃ§Ãµes ========================
 */

int buscar(const int x, const int y, char *categoria)
```

```
/*
 * desc         :       Busca em um raio de 100 unidades da posiÃ§Ã£o (x,y) todos os
 *                              estabelecimentos da categoria dada; ou busca em um raio de
 *                              100 unidades todos os estabelecimentos de qualquer categoria,
 *                              caso categoria = NULL; ou lista todos os estabelecimentos de
 *                              uma categoria, caso x = -1 e y = -1; ou lista todos os
 *                              estabelecimentos, caso x = -1, y = -1 e categoria = NULL.
 *
 * params       :       1.      Coordenada x da posiÃ§Ã£o do centro da busca.
 *                              2.      Coordenada y da posiÃ§Ã£o do centro da busca.
 *                              3.      Categoria dos estabelecimentos a serem buscados.
 *
 * output       :       O nÃºmero de estabelecimentos encontrados pela busca.
 *                              (O vetor da estrutura busca tambÃ©m contÃ©m dados de saÃ-da.)
 */
{
        int out = 0;
        int tmp[ITEMLIM];
        int dx, dy;
        int i;

        readDB();

        /* procura na lista pelos estabelecimentos que atendem aos critÃ©rios */
        for (i = 0; i < lista_len; i++) {
                dx = lista[i].posx - x;
                dy = lista[i].posy - y;
                if ((dx*dx + dy*dy <= 10000 || (x == -1 && y == -1))
                && (categoria == NULL
                || strcmp(lista[i].categoria, categoria) == 0)) {
                        tmp[out++] = i;
                }

        }

        /* monta o resultado da busca com o nome e id dos estabelecimentos */
        busca = calloc(out, sizeof(res_busca_t));
        for (i = 0; i < out; i++) {
                busca[i].id = lista[tmp[i]].id;
                strncpy(busca[i].nome, lista[tmp[i]].nome, MAXSTRLEN);
        }

        writeDB();
        free(lista);
        lista_len = 0;

        return out;
}

void sendBusca(int N, int S)
/*
 * desc         :       Envia o ID e o nome de cada estabelecimento retornado por uma
 *                              busca.
 *
 * params       :       1.      NÃºmeros de resultados da busca.
 *                              2.      Socket pelo qual serÃ£o enviados os dados.
 *
 * output       :       Nenhuma.
```



```
 */
{
        char buf[BUFLEN];
        int len;
        int i;

        if (N == 0) {
                sprintf(buf, "Nenhum estabelecimento encontrado.\n\r");
                if (send(S, buf, strlen(buf), 0) == -1) {
                        perror("send");
                }
                return;
        }

        len = 0;
        sprintf(buf, "( ID ) Nome\n");
        if (send(S, buf, strlen(buf), 0) == -1) {
                perror("send");
        }
        memset(buf, 0, sizeof(buf));

        for (i = 0; i < N; i++) {
                sprintf(buf, "(%.4d) %s\n", busca[i].id, busca[i].nome);
                if (send(S, buf, strlen(buf), 0) == -1) {
                        perror("send");
        }
        memset(buf, 0, sizeof(buf));
        }

        free(busca);
}

void sendInfoID(int id, int S)
/*
 * desc         :       Envia ao socket S uma mensagem contendo todas as informaÃ§Ãµes
 *                              do estabelecimento com identificador id.
 *
 * params       :       1.      id do estabelecimento selecionado.
 *                              2.      socket para enviar a saÃ-da de dados.
 *
 * output       :       Nenhuma.
 */
{
        bool_t exist_id = FALSE;
        char *error_msg = "NÃ£o existe o estabelecimento com o id dado.\n";
        char buf[BUFLEN];
        int i;

        readDB();

        /* procura pela id na lista */
        for (i = 0; i < lista_len; i++) {

                if (lista[i].id == id) {
                        exist_id = TRUE;

                        /* caso encontre o id, prepara a mensagem e envia por S */
                        snprintf(buf, BUFLEN,
                                "<<< %s >>>\n ID: %.4d\tCategoria: %s\n EndereÃ§o: %s\n PosiÃ§Ã£o: (%d,%d)\tNota: %.2f\n",
                                lista[i].nome, lista[i].id, lista[i].categoria,
                                lista[i].endereco, lista[i].posx, lista[i].posy,
                                (float)(lista[i].pontuacao / lista[i].votos));
                        if (send(S, buf, strlen(buf), 0) == -1) {
                        perror("send");
                }
                }
```

```
        }

        /* caso nÃ£o encontre o id, envia uma mensagem de erro */
        if (!exist_id) {
                if (send(S, error_msg, strlen(error_msg), 0) == -1) {
                perror("send");
        }
        }

        writeDB();
        free(lista);
        lista_len = 0;
}

void sendCategorias(int S)
/*
 * desc         :       Envia ao cliente a lista das categorias de estabelecimen
tos
 *                              existentes.
 *
 * params       :       1.      Socket para enviar a saÃ-da de dados.
 *
 * output       :       Nenhuma.
 */
{
        char buf[BUFLEN];
        char categorias[lista_len][MAXSTRLEN];
        int i, j, n;
        bool_t igual;

        readDB();

        /* primeira categoria */
        strncpy(categorias[0], lista[0].categoria, strlen(lista[0].categoria));
        n = 1;

        /* percorre a lista de estabelecimentos procurando pelas categorias */
        for (i = 1; i < lista_len; i++) {

                igual = FALSE;

                /* verifica se a categoria corrente jÃ¡ foi encontrada */
                for (j = 0; j < n; j++) {
                        if (strcmp(categorias[j], lista[i].categoria) == 0) {
                                igual = TRUE;
                        }
                }

                /* se a categoria corrente Ã© nova, insere no resultado */
                if (igual == FALSE) {
                        sprintf(categorias[n++], "%s", lista[i].categoria);
                }
        }

        /* envia a lista das categorias pelo socket S */
        for (i = 0; i < n; i++) {
                sprintf(buf, "%s\n\r", categorias[i]);
                if (send(S, buf, strlen(buf), 0) == -1) {
                        perror("send");
                }
        }

        writeDB();
        free(lista);
        lista_len = 0;
}

void votarID(int id, int nota, int S)
/*
```



```
 * desc         :       Computa o voto dado em um estabelecimento.
 *
 * params       :       1.      ID do estabelecimento selecionado.
 *                              2.      Nota a ser dada ao estabelecimento.
 *                              3.      Socket para enviar a saÃ-da de dados.
 *
 * output       :       Nenhuma.
 */
{
        bool_t exist_id = FALSE;
        char *error_msg = "NÃ£o existe o estabelecimento com o id dado.\n";
        char buf[BUFLEN];
        int i;

        readDB();

        /* procura pela id na lista */
        for (i = 0; i < lista_len; i++) {

                if (lista[i].id == id) {
                        exist_id = TRUE;

                        /* caso exista o id, atualiza a pontuaÃ§Ã£o e votos */
                        lista[i].pontuacao += nota;
                        lista[i].votos += 1;

                        /* prepara a mensagem e envia por S */
                        sprintf(buf, "Nota %d dada a %s.\n\r", nota, lista[i].nome);
                        if (send(S, buf, strlen(buf), 0) == -1) {
                        perror("send");
                }
                }
        }

        /* caso nÃ£o encontre o id, envia uma mensagem de erro */
        if (!exist_id) {
                if (send(S, error_msg, strlen(error_msg), 0) == -1) {
                perror("send");
        }
        }

        writeDB();
        free(lista);
        lista_len = 0;
}


/* === FunÃ§Ãµes auxiliares de networking ================================= */

int bindTCP(char *port)
/*
 * desc         :       Associa a porta dada a um socket a ser retornado.
 *
 * params       :       1.      String contendo a porta a ser usada.
 *
 * output       :       O socket ao qual a porta foi associada.
 */
{
        int socket_fd;
        struct addrinfo hints, *servinfo, *p;
        int status;
        int optval = 1;

        /* inicialmente retornando erro */
        socket_fd = -1;

        /* estrutura a ser usada para obter um endereÃ§o IP */
        memset(&hints, 0, sizeof(hints));
```

```
        hints.ai_family = AF_INET;                      /* IPv4 */
        hints.ai_socktype = SOCK_STREAM;        /* TCP */
        hints.ai_flags = AI_PASSIVE;

        /* obtem a lista ligada com as associaÃ§Ãµes possÃ-veis */
        if ((status = getaddrinfo(NULL, port, &hints, &servinfo)) != 0) {
                fprintf(stderr, "%s: getaddrinfo error: %s\n", __func__, gai_strerror(st
atus));
                exit(EXIT_FAILURE);
        }

        /* percorre servinfo tentando fazer o bind */
        for (p = servinfo; p != NULL; p = p->ai_next) {

                /* tenta obter um socket */
                if ((socket_fd = socket(p->ai_family, p->ai_socktype, p->ai_prot
ocol)) == -1) {
                        perror("socket");
                        continue;
                }

                /* configura o socket para permitir reuso do endereÃ§o IP */
                if (setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &optval, siz
eof(int)) == -1) {
                        perror("setsockopt");
                        exit(EXIT_FAILURE);
                }

                /* tenta associar o endereÃ§o IP obtido de servinfo com o socket
 */
                if (bind(socket_fd, p->ai_addr, p->ai_addrlen) == -1) {
                        close(socket_fd);
                        perror("bind");
                        sleep(1);
                        continue;
                }

                break;
        }

        /* tratamento de erro caso o bind falhe */
        if (p == NULL) {
                pMyError(BIND_ERROR, __func__);
                exit(EXIT_FAILURE);
        }

        freeaddrinfo(servinfo);

        return socket_fd;
}

void serverReady(int S)
{
        char msg[MAXSTRLEN];

        memset(msg, 0, sizeof(msg));
        sprintf(msg, "SERVER_READY\n\r");
        if (send(S, msg, strlen(msg), 0) == -1) {
                perror("send");
        }
}



/* === Interpretados de comandos ========================================== */

int interpretador(char *cmd, int S)
/*
 * desc         :       Interpretador dos comandos recebidos pelo servidor.
```



```
 *
 * params       :       1.      Buffer contendo a linha de comando.
 *                              2.      Socket para o qual respostas sÃ£o enviad
as.
 *
 * output       :       0 para continuar e -1 para encerrar a conexÃ£o.
 */
{
        char msg[BUFLEN];
        char tmp[BUFLEN];
        char *tok;
        int id, nota;
        int i;

        if ((tok = strtok(cmd, " \n\r")) == NULL) {
                return 0;
        }

        /* entrando a posiÃ§Ã£o do cliente */
        if (strcmp(tok, "posicao") == 0) {

                if ((tok = strtok(NULL, ",")) == NULL) {
                        return 0;
                }
                else {
                        cli_posx = atoi(tok);
                }
                if ((tok = strtok(NULL, ",")) == NULL) {
                        return 0;
                }
                else {
                        cli_posy = atoi(tok);
                }

                /* tratamento de erro na entrada das coordenadas */
                if (POSLIMMIN > cli_posx || cli_posx > POSLIMMAX
                || POSLIMMIN > cli_posy || cli_posy > POSLIMMAX) {
                        strncpy(msg, "Coordenadas devem estar entre ", (size_t)(30));
                        memset(tmp, 0, sizeof(tmp));
                        sprintf(tmp, "%d", POSLIMMIN);
                        strncat(msg, tmp, strlen(tmp));
                        strncat(msg, " e ", (size_t)(3));
                        memset(tmp, 0, sizeof(tmp));
                        sprintf(tmp, "%d", POSLIMMAX);
                        strncat(msg, tmp, strlen(tmp));
                        strcat(msg, ".\n\r");
                        if (send(S, msg, strlen(msg), 0) == -1) {
                                perror("send");
                        }
                        cli_posx = -1;
                        cli_posy = -1;
                }
                else {
                        strncpy(msg, "PosiÃ§Ã£o atual (", (size_t)(30));
                        memset(tmp, 0, sizeof(tmp));
                        sprintf(tmp, "%d", cli_posx);
                        strncat(msg, tmp, strlen(tmp));
                        strncat(msg, ",", (size_t)(3));
                        memset(tmp, 0, sizeof(tmp));
                        sprintf(tmp, "%d", cli_posy);
                        strncat(msg, tmp, strlen(tmp));
                        strcat(msg, ").\n\r");
                        if (send(S, msg, strlen(msg), 0) == -1) {
                                perror("send");
                        }
                }
        }
        else
        /* requisitando a lista das categorias existentes */
```

```
        if (strcmp(tok, "categorias") == 0) {
                sendCategorias(S);
        }
        else
        /* requisitando alguma listagem de estabelecimentos */
        if (strcmp(tok, "buscar") == 0 || strcmp(tok, "listar") == 0) {

                if ((tok = strtok(NULL, "\n\r")) == NULL) {
                        return 0;
                }

                /* listar todos os estabelecimentos */
                if (strcmp(tok, "todos") == 0) {

                        i = buscar(-1, -1, NULL);
                        sendBusca(i, S);
                }
                else
                if (strcmp(tok, "perto") == 0) {

                        if ((tok = strtok(NULL, "\n\r")) == NULL) {
                                return 0;
                        }
                        /* listar todos os estabelecimentos a 100m */
                        if (strcmp(tok, "todos") == 0) {

                                if (cli_posx == -1 || cli_posy == -1) {
                                        strcpy(msg, "Informe sua posiÃ§Ã£o antes.\n\r");
                                        if (send(S, msg, strlen(msg), 0) == -1)
{
                                                perror("send");
                                        }
                                }
                                else {
                                        i = buscar(cli_posx, cli_posy, NULL);
                                        sendBusca(i, S);
                                }
                        }
                        else
                        /* listar todos os estabelecimentos de um categoria
                         * e que estejam a menos de 100m */
                        if (strcmp(tok, "categoria") == 0) {

                                if ((tok = strtok(NULL, "\n\r")) == NULL) {
                                        return 0;
                                }

                                if (cli_posx == -1 || cli_posy == -1) {
                                        strcpy(msg, "Informe sua posiÃ§Ã£o antes.\n\r");
                                        if (send(S, msg, strlen(msg), 0) == -1)
{
                                                perror("send");
                                        }
                                }
                                else {
                                        i = buscar(cli_posx, cli_posy, tok);
                                        sendBusca(i, S);
                                }
                        }
                }
                else
                /* listar todos os estabelecimentos de um categoria */
                if (strcmp(tok, "categoria") == 0) {

                        if ((tok = strtok(NULL, "\n\r")) == NULL) {
                                return 0;
                        }

                        i = buscar(-1, -1, tok);
```



```
                        sendBusca(i, S);
                }
        }
        else
        /* requisitando informaÃ§Ãµes acerca do id */
        if (strcmp(tok, "info") == 0) {
                if ((tok = strtok(NULL, ",")) != NULL) {
                        id = atoi(tok);
                }
                sendInfoID(id, S);
        }
        else
        /* votando no id */
        if (strcmp(tok, "votar") == 0) {

                if ((tok = strtok(NULL, "\n\r")) == NULL) {
                        return 0;
                }
                else {
                        id = atoi(tok);
                }
                if ((tok = strtok(NULL, "\n\r")) == NULL) {
                        return 0;
                }
                else {
                        nota = atoi(tok);
                }

                /* tratamento de erro para nota fora do intervalo */
                if (0 > nota || nota > 10 ) {
                        sprintf(msg, "Nota deve estar entre 0 e 10.\n\r");
                        if (send(S, msg, strlen(msg), 0) == -1) {
                                perror("send");
                        }
                }
                else {
                        votarID(id, nota, S);
                }
        }
        else
        /* requisitando encerramento da conexÃ£o */
        if (strcmp(tok, "sair") == 0) {
                return -1;
        }
}


/* === FunÃ§Ãµes auxiliares de depuraÃ§Ã£o ================================
*/

void DEBUG_printItem(item_t *item)
{
        printf("*** item ***\n");
        printf("Id: %d\t", item->id);
        printf("PosiÃ§Ã£o: %d,%d\n", item->posx, item->posy);
        printf("Categoria: %s\t", item->categoria);
        printf("Nome: %s\n", item->nome);
        printf("EndereÃ§o: %s\n", item->endereco);
        printf("Pontos acumul.: %d\t", item->pontuacao);
        printf("Total de votos: %d\n", item->votos);
        printf("\n");
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>



#define BUFLEN 1024



/* === Tipos auxiliares ================================================= */

typedef enum
{
        FALSE = 0,
        TRUE = 1
} bool_t;



/* === Ferramentas para tratamento de erros ============================= */

typedef enum {
    NO_ERROR = 0,
    USAGE_ERROR,
    PORT_OUT_RANGE,
    CONNECT_ERROR,
    MYERROR_LIM

} my_error_t;

const char* my_error_desc[] =
{
    "",
    "usage: ./cliente_info_tcp <port number>",
    "error: port number must be between 1 and 8000",
    "error: failed to connect"
    ""
};

void pMyError(my_error_t e, const char *function)
{
    if(NO_ERROR < e && e < MYERROR_LIM) {
        fprintf(stderr, "%s: %s.\n", function, my_error_desc[e]);
    }
}



/* ========================================================================
 * === Pragmas das funÃ§Ãµes ============================================
 * ======================================================================== */

/* === FunÃ§Ãµes auxiliares de networking ================================= */
int connectTCP(char *endIP, char *port);



/* ========================================================================
 * === MAIN ===============================================================
 * ======================================================================== */
```



```c
int main(int argc, char *argv[])
{
        int sock;

        fd_set rfds0, rfds1;
        struct timeval tv;

        int maxfd, rval;

        bool_t server_ready;

        char cmd[BUFLEN];
        char buf[BUFLEN];
        int len;
        int i, n;

        /* verificando argumentos */
        if (argc < 3) {
                pMyError(USAGE_ERROR, __func__);
                exit(EXIT_FAILURE);
        }

        i = atoi(argv[2]);
        if (0 >= i && i > 8000) {
                pMyError(PORT_OUT_RANGE, __func__);
                exit(EXIT_FAILURE);
        }

        sock = connectTCP(argv[1], argv[2]);

        maxfd = fileno(stdin);
        maxfd = (maxfd < sock) ? sock : maxfd;

        FD_ZERO(&rfds0);

        FD_SET(fileno(stdin), &rfds0);
        FD_SET(sock, &rfds0);

        tv.tv_sec = 0;
        tv.tv_usec = 0;

        memset(buf, 0, sizeof(buf));

        server_ready = TRUE;

        while (1) {

                rfds1 = rfds0;

        if ((rval = select(maxfd + 1, &rfds1, NULL, NULL, &tv)) == -1) {
                        perror("select");
                        exit(EXIT_FAILURE);
                }
                else if (rval > 0) {

                        if (FD_ISSET(fileno(stdin), &rfds1) && server_ready) {
                                memset(cmd, 0, sizeof(cmd));
                                fgets(cmd, BUFLEN, stdin);

                                if (send(sock, cmd, strlen(cmd), 0) == -1) {
                                        perror("send");
                                }

                                server_ready = FALSE;

                                if (strcmp(cmd, "sair\n") == 0) {
                                        break;
                                }
                        }
```

```
                        else if (FD_ISSET(sock, &rfds1)) {
                                memset(buf, 0, sizeof(buf));
                                len = recv(sock, buf, BUFLEN, 0);

                                if (strstr(buf, "SERVER_READY\n\r") != NULL) {
                                        server_ready = TRUE;
                                }
                                if (strcmp(buf, "\n") != 0 || strcmp(buf, "\r") !
= 0) {
                                        printf("%s", buf);
                                }
                        }

                }

        }

        return 0;
}



/* ========================================================================
 * === ImplementaÃ§Ã£o das funÃ§Ãµes ========================================
 * ======================================================================== */

/* === FunÃ§Ãµes auxiliares de networking ================================== */

int connectTCP(char *endIP, char *port)
/*
 * desc         :       Conecta a um endereÃ§o IP e porta a um socket a ser reto
rnado.
 *
 * params       :       1.      String contendo o endereÃ§o IP a ser usado.
 *                      2.      String contendo a porta a ser usada.
 *
 * output       :       O socket da conexÃ£o criada.
 */
{
        int socket_fd;
        struct addrinfo hints, *servinfo, *p;
        int status;

        /* inicialmente retornando erro */
        socket_fd = -1;

        /* estrutura a ser usada para obter um endereÃ§o IP */
        memset(&hints, 0, sizeof(hints));
        hints.ai_family = AF_INET;                      /* IPv4 */
        hints.ai_socktype = SOCK_STREAM;        /* TCP */

        /* obtem a lista ligada com as associaÃ§Ãµes possÃ-veis */
        if ((status = getaddrinfo(endIP, port, &hints, &servinfo)) != 0) {
                fprintf(stderr, "%s: getaddrinfo error: %s\n", __func__, gai_strerror(st
atus));
                exit(EXIT_FAILURE);
        }

        /* percorre servinfo tentando se conectar */
        for (p = servinfo; p != NULL; p = p->ai_next) {

                /* tenta obter um socket */
                if ((socket_fd = socket(p->ai_family, p->ai_socktype, p->ai_prot
ocol)) == -1) {
                        perror("socket");
                        continue;
                }

                /* tenta se conectar ao endereÃ§o IP dado */
```



```
                if (connect(socket_fd, p->ai_addr, p->ai_addrlen) == -1) {
                        close(socket_fd);
                        perror("connect");
                        continue;
                }

                break;
        }

        /* tratamento de erro caso o bind falhe */
        if (p == NULL) {
                pMyError(CONNECT_ERROR, __func__);
                exit(EXIT_FAILURE);
        }

        freeaddrinfo(servinfo);

        return socket_fd;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>


/*
 * DBFILENAME Ã© o nome do arquivo com os dados dos estabelecimentos.
 *
 * O arquivo guarda na primeira linha o nÃºmero de estabelecimentos existentes
 * e nas linhas seguintes os dados de cada estabelecimento, um por linha,
 * separado por ponto-e-vÃ-rgula. O arquivo deve ter uma linha em branco no
 * final.
 *
 * Os dados sÃ£o:
 *      id;x;y;categoria;nome;endereÃ§o;pontuacao;votos
 * onde:
 *      - 'id' Ã© um nÃºmero de quatro dÃ-gitos;
 *      - 'x' Ã© a coordenada x (0-1000);
 *      - 'y' Ã© a coordenada y (0-1000);
 *      - 'categoria' Ã© uma string com a categoria do estabelecimento;
 *      - 'nome' Ã© uma string com o nome do estabelecimento;
 *      - 'endereco' Ã© uma string com o endereÃ§o;
 *      - 'pontuacao' Ã© a soma das notas das ao estabelecimento;
 *  - 'votos' Ã© a quantidade de notas recebidas.
 *
 * As strings tem comprimento mÃ¡ximo de 256 caracteres.
 */
#define DBFILENAME "db.csv"
/* os limites das coordenaras de posicionamento */
#define POSLIMMAX 1000
#define POSLIMMIN 0
/* tamanho do backlog da funÃ§Ã£o listen */
#define BACKLOG 100
/* nÃºmero mÃ¡ximo de estabelecimentos na lista */
#define ITEMLIM 8000
/* tamanho mÃ¡ximo das strings usadas */
#define BUFLEN 1024
#define MAXSTRLEN 256



/* === Tipos auxiliares ================================================= */

typedef enum
{
        FALSE = 0,
        TRUE = 1
} bool_t;



/* === Estrutura de dados dos estabelecimentos ========================== */

typedef struct item {
        int id;
        int posx;
        int posy;
        char categoria[MAXSTRLEN];
        char nome[MAXSTRLEN];
        char endereco[MAXSTRLEN];
```



```c
        int pontuacao;
        int votos;
} item_t;

typedef struct res_busca {
        int id;
        char nome[MAXSTRLEN];
} res_busca_t;



/* === Ferramentas para tratamento de erros ============================= */

typedef enum {
    NO_ERROR = 0,
    USAGE_ERROR,
    PORT_OUT_RANGE,
    DB_FEWER_ROWS,
    BIND_ERROR,
    BUSCA_NOT_SET,
    MYERROR_LIM

} my_error_t;

const char* my_error_desc[] =
{
    "",
    "usage: ./servidor_info_tcp <port number>",
    "error: port number must be between 1 and 8000",
    "error: database constains less rows than specified",
    "error: failed to bind",
    "error: vector 'busca' is not set",
    ""
};

void pMyError(my_error_t e, const char *function)
{
    if(NO_ERROR < e && e < MYERROR_LIM) {
        fprintf(stderr, "%s: %s.\n", function, my_error_desc[e]);
    }
}



/* === Estruturas de dados com as informaÃ§Ãµes dos estabelecimentos ======== */

static item_t *lista = NULL;
static int lista_len = 0;
static res_busca_t *busca = NULL;
static int cli_posx = -1;       // -1 significa posiÃ§Ã£o do cliente
static int cli_posy = -1;       // nÃ£o configurada ainda



/* ========================================================================
 * === Pragmas das funÃ§Ãµes ================================================
 * ======================================================================== */

/* === FunÃ§Ãµes de processamento de informaÃ§Ãµes ============================
 */
void readItemInfo(item_t *item, char *buf);
void readDB(void);
void writeDB(void);

/* === FunÃ§Ãµes de busca e impressÃ£o de informaÃ§Ãµes ======================
 */
int buscar(const int x, const int y, char *categoria);
void sendBusca(int N, int S, struct sockaddr *R, int format);
void sendInfoID(int id, int S, struct sockaddr *R);
```

```
int listarCategorias(void);
void votarID(int id, int nota, int S, struct sockaddr *R);

/* === FunÃ§Ãµes auxiliares de networking ================================= */
char *sendACK(int S, char *buf, struct sockaddr *R);
int bindUDP(char *port);
void serverReady(int S, struct sockaddr *R);

/* === Interpretados de comandos ======================================== */
int interpretador(char *cmd, int S, struct sockaddr *R);



/* ========================================================================
 * === MAIN ===============================================================
 * ======================================================================== */

int main(int argc, char *argv[])
{
        int sock;
        struct sockaddr_storage remote_st;
        socklen_t st_len;

        char *cmd;
        char msg[BUFLEN];
        char buf[BUFLEN];
        int len;
        int i;
        int rval = 0;

        /* verificando argumentos */
        if (argc < 2) {
                pMyError(USAGE_ERROR, __func__);
                exit(EXIT_FAILURE);
        }

        /* verificaÃ§Ã£o da porta usada: a porta deve ser well known */
        i = atoi(argv[1]);
        if (0 >= i && i > 8000) {
                pMyError(PORT_OUT_RANGE, __func__);
                exit(EXIT_FAILURE);
        }

        /* faz o bind da porta ao sock */
        sock = bindUDP(argv[1]);

        st_len = sizeof(remote_st);

    /* main loop */
        while ((len = recvfrom(sock, buf, sizeof(buf), 0,
        (struct sockaddr *)&remote_st, &st_len)) != -1) {

                if (len > 0) {

                        /* envia uma confirmaÃ§Ã£o de recebimento */
                        cmd = sendACK(sock, buf, (struct sockaddr *)&remote_st);

                        /* interpretador de comandos */
                        rval = interpretador(cmd, sock, (struct sockaddr *)&remo
te_st);

                }

                memset(buf, 0, sizeof(buf));

                serverReady(sock, (struct sockaddr *)&remote_st);

        }
```



```
        close(sock);

        return 0;
}



/* ========================================================================
 * === ImplementaÃ§Ã£o das funÃ§Ãµes ======================================
 * ======================================================================== */

/* === FunÃ§Ãµes de processamento de informaÃ§Ãµes ============================
*/

void readItemInfo(item_t *item, char *buf)
/*
 * desc         :       Preenche a estrutura de dados item com as informaÃ§Ãµes
contidas
 *                              no buffer.
 *
 * params       :       1.      Estrutura de dados de saÃ-da.
 *                              2.      Buffer para a entrada de informaÃ§Ãµes.
 *
 * output       :       Nenhuma.
 */
{
        int i = 0;
        char *tok;

        tok = strtok(buf,";");
        while (tok != NULL) {
                switch (i) {
                        case 0:
                                item->id = atoi(tok);
                                break;
                        case 1:
                                item->posx = atoi(tok);
                                break;
                        case 2:
                                item->posy = atoi(tok);
                                break;
                        case 3:
                                strncpy(item->categoria, tok, MAXSTRLEN);
                                break;
                        case 4:
                                strncpy(item->nome, tok, MAXSTRLEN);
                                break;
                        case 5:
                                strncpy(item->endereco, tok, MAXSTRLEN);
                                break;
                        case 6:
                                item->pontuacao = atoi(tok);
                                break;
                        case 7:
                                item->votos = atoi(tok);
                                break;
                }
                i++;
                tok = strtok(NULL,";");
        }
}

void readDB(void)
/*
 * desc         :       LÃª as informaÃ§Ãµes de todos os estabelecimentos para u
m vetor
 *                              global (lista) na memÃ³ria. A entrada Ã© um arqu
ivo de texto
 *                              DBFILENAME (cujo formato Ã© dado junto a "#defin
```

```
e DBFILENAME").
 *
 * params       :       Nenhum.
 *
 * output       :       Nenhuma.
 */
{
        FILE *db;
        char buf[BUFLEN];
        int i;

        db = fopen (DBFILENAME,"r");
        if (db == NULL) {
                perror("failed to open the database");
                exit(EXIT_FAILURE);
        }

        /* lÃª o nÃºmero de estabelecimentos */
        fgets(buf, BUFLEN, db);
        lista_len = atoi(buf);

        /* cria a lista de estabelecimentos */
        lista = calloc(lista_len, sizeof(item_t));

        /* lÃª as informaÃ§Ãµes de cada estabelecimento */
        i = lista_len;
        while (!feof(db) && i > 0) {
                fgets(buf, BUFLEN, db);
                readItemInfo(&lista[lista_len - (i--)], buf);
        }
        /* tratamento de erro para quando temos menos estabelecimentos do que N
*/
        if (i > 0) {
                pMyError(DB_FEWER_ROWS, __func__);
                exit(EXIT_FAILURE);
        }

        fclose(db);
}

void writeDB(void)
/*
 * desc         :       Escreve as informaÃ§Ãµes de todos os estabelecimentos pr
esentes
 *                              na memÃ³ria em um arquivo DBFILENAME.
 *
 * params       :       Nenhum.
 *
 * output       :       Nenhuma.
 */
{
        FILE *db;
        char buf[BUFLEN];
        int i;

        db = fopen (DBFILENAME,"w");
        if (db == NULL) {
                perror("failed to write to the database");
                exit(EXIT_FAILURE);
        }

        fprintf(db, "%d\n", lista_len);

        for (i = 0; i < lista_len; i++) {
                fprintf(db, "%.4d;", lista[i].id);
                fprintf(db, "%d;%d;", lista[i].posx, lista[i].posy);
                fprintf(db, "%s;", lista[i].categoria);
                fprintf(db, "%s;", lista[i].nome);
                fprintf(db, "%s;", lista[i].endereco);
```



```
                fprintf(db, "%d;", lista[i].pontuacao);
                fprintf(db, "%d\n", lista[i].votos);

        }

        fprintf(db, "\n");

        fclose(db);
}


/* === FunÃ§Ãµes de busca e impressÃ£o de informaÃ§Ãµes ========================
 */

int buscar(const int x, const int y, char *categoria)
/*
 * desc         :       Busca em um raio de 100 unidades da posiÃ§Ã£o (x,y) todo
s os
 *                              estabelecimentos da categoria dada; ou busca em
um raio de
 *                              100 unidades todos os estabelecimentos de qualqu
er categoria,
 *                              caso categoria = NULL; ou lista todos os estabel
ecimentos de
 *                              uma categoria, caso x = -1 e y = -1; ou lista to
dos os
 *                              estabelecimentos, caso x = -1, y = -1 e categori
a = NULL.
 *
 * params       :       1.      Coordenada x da posiÃ§Ã£o do centro da busca.
 *                              2.      Coordenada y da posiÃ§Ã£o do centro da b
usca.
 *                              3.      Categoria dos estabelecimentos a serem b
uscados.
 *
 * output       :       O nÃºmero de estabelecimentos encontrados pela busca.
 *                              (O vetor da estrutura busca tambÃ©m contÃ©m dado
s de saÃ-da.)
 */
{
        int out = 0;
        int tmp[ITEMLIM];
        int dx, dy;
        int i;

        readDB();

        /* procura na lista pelos estabelecimentos que atendem aos critÃ©rios */
        for (i = 0; i < lista_len; i++) {
                dx = lista[i].posx - x;
                dy = lista[i].posy - y;
                if ((dx*dx + dy*dy <= 10000 || (x == -1 && y == -1))
                && (categoria == NULL
                || strcmp(lista[i].categoria, categoria) == 0)) {
                        tmp[out++] = i;
                }

        }

        /* monta o resultado da busca com o nome e id dos estabelecimentos */
        busca = calloc(out, sizeof(res_busca_t));
        for (i = 0; i < out; i++) {
                busca[i].id = lista[tmp[i]].id;
                strncpy(busca[i].nome, lista[tmp[i]].nome, MAXSTRLEN);
        }

        writeDB();
        free(lista);
```

```
        lista_len = 0;

        return out;
}

void sendBusca(int N, int S, struct sockaddr *R, int format)
/*
 * desc         :       Envia o ID e o nome de cada estabelecimento retornado po
r uma
 *                              busca.
 *
 * params       :       1.      NÃºmeros de resultados da busca.
 *                              2.      Socket pelo qual serÃ£o enviados os dado
s.
 *                              3.      InformaÃ§Ãµes de rede sobre o cliente.
 *                              4.      Formato das mensagens: 0 para lista de e
stabelecimentos
 *                                      ou 1 para lista de categorias.
 *
 * output       :       Nenhuma.
 */
{
        char buf[BUFLEN];
        int len;
        int i, j;

        if (N == 0) {
                sprintf(buf, "Nenhum estabelecimento encontrado.\n\r");
                if (sendto(S, buf, strlen(buf), 0, R, sizeof(struct sockaddr_sto
rage)) == -1) {
                        perror("sendto");
                }
                return;
        }

        len = 0;
        if (format == 0) {
                sprintf(buf, "( ID ) Nome\n");
        }
        else if (format == 1) {
                sprintf(buf, "Categorias:\n");
        }
        len = strlen(buf);

        j = 0;
        for (i = 0; i < N; i++) {
                /* montando mensagens menores que 1024 bytes */
                if (j <= 2) {
                        if (format == 0) {
                                sprintf(buf+len, "(%.4d) %s\n", busca[i].id, busca
[i].nome);
                        }
                        else if (format == 1) {
                                sprintf(buf+len, " %s\n", busca[i].nome);
                        }
                        len = strlen(buf);
                }
                if (j == 2) {
                        if (sendto(S, buf, strlen(buf), 0, R, sizeof(struct sock
addr_storage)) == -1) {
                                perror("sendto");
                        }
                        memset(buf, 0, sizeof(buf));
                        len = 0;
                        j = 0;
                        }
                        else {
                                j++;
                        }
```



```
        }

        /* se alguma mensagem foi montada mas nÃ£o enviada, envia */
        if (j > 0) {
                if (sendto(S, buf, strlen(buf), 0, R, sizeof(struct sockaddr_sto
rage)) == -1) {
                        perror("sendto");
                }
                }

        free(busca);
}

void sendInfoID(int id, int S, struct sockaddr *R)
/*
 * desc         :       Envia ao socket S uma mensagem contendo todas as informa
Ã§Ãµes
 *                              do estabelecimento com identificador id.
 *
 * params       :       1.      ID do estabelecimento selecionado.
 *                              2.      Socket para enviar a saÃ-da de dados.
 *                              3.      InformaÃ§Ãµes de rede sobre o cliente.
 *
 * output       :       Nenhuma.
 */
{
        bool_t exist_id = FALSE;
        char *error_msg = "NÃ£o existe o estabelecimento com o id dado.\n";
        char buf[BUFLEN];
        int i;

        readDB();

        /* procura pela id na lista */
        for (i = 0; i < lista_len; i++) {

                if (lista[i].id == id) {
                        exist_id = TRUE;

                        /* caso encontre o id, prepara a mensagem e envia por S
*/
                        snprintf(buf, BUFLEN,
                                "<<< %s >>>\n ID: %.4d\tCategoria: %s\n EndereÃ§o: %s\n PosiÃ§
Ã£o: (%d,%d)\tNota: %.2f\n",
                                lista[i].nome, lista[i].id, lista[i].categoria,
                                lista[i].endereco, lista[i].posx, lista[i].posy,
                                (float)(lista[i].pontuacao / lista[i].votos));
                        if (sendto(S, buf, strlen(buf), 0, R, sizeof(struct sock
addr_storage)) == -1) {
                                perror("sendto");
                        }
                        }
        }

        /* caso nÃ£o encontre o id, envia uma mensagem de erro */
        if (!exist_id) {
                if (sendto(S, error_msg, strlen(error_msg), 0, R, sizeof(struct
sockaddr_storage)) == -1) {
                        perror("sendto");
                }
                }

        writeDB();
        free(lista);
        lista_len = 0;
}

int listarCategorias(void)
/*
```

```
 * desc         :       Gera lista das categorias de estabelecimentos existentes
.
 *
 * params       :       Nenhum.
 *
 * output       :       NÃºmero de categorias.
 */
{
        char buf[BUFLEN];
        char categorias[lista_len][MAXSTRLEN];
        int i, j, n;
        bool_t igual;

        readDB();

        busca = calloc(lista_len, sizeof(res_busca_t));

        /* primeira categoria */
        strncpy(busca[0].nome, lista[0].categoria, strlen(lista[0].categoria));
        n = 1;

        /* percorre a lista de estabelecimentos procurando pelas categorias */
        for (i = 1; i < lista_len; i++) {

                igual = FALSE;

                /* verifica se a categoria corrente jÃ¡ foi encontrada */
                for (j = 0; j < n; j++) {
                        if (strcmp(busca[j].nome, lista[i].categoria) == 0) {
                                igual = TRUE;
                        }
                }

                /* se a categoria corrente Ã© nova, insere no resultado */
                if (igual == FALSE) {
                        sprintf(busca[n++].nome, "%s", lista[i].categoria);
                }
        }

        writeDB();
        free(lista);
        lista_len = 0;

        return n;
}

void votarID(int id, int nota, int S, struct sockaddr *R)
/*
 * desc         :       Computa o voto dado em um estabelecimento.
 *
 * params       :       1.      ID do estabelecimento selecionado.
 *                              2.      Nota a ser dada ao estabelecimento.
 *                              3.      Socket para enviar a saÃ-da de dados.
 *                              4.      InformaÃ§Ãµes de rede sobre o cliente.
 *
 * output       :       Nenhuma.
 */
{
        bool_t exist_id = FALSE;
        char *error_msg = "NÃ£o existe o estabelecimento com o id dado.\n";
        char buf[BUFLEN];
        int i;

        readDB();

        /* procura pela id na lista */
        for (i = 0; i < lista_len; i++) {

                if (lista[i].id == id) {
```



```
                        exist_id = TRUE;

                        /* caso exista o id, atualiza a pontuaÃ§Ã£o e votos */
                        lista[i].pontuacao += nota;
                        lista[i].votos += 1;

                        /* prepara a mensagem e envia por S */
                        sprintf(buf, "Nota %d dada a %s.\n\r", nota, lista[i].nome);
                        if (sendto(S, buf, strlen(buf), 0, R, sizeof(struct sock
addr_storage)) == -1) {
                                perror("sendto");
                        }
                        }
        }

        /* caso nÃ£o encontre o id, envia uma mensagem de erro */
        if (!exist_id) {
                if (sendto(S, error_msg, strlen(error_msg), 0, R, sizeof(struct
sockaddr_storage)) == -1) {
                perror("sendto");
        }
        }

        writeDB();
        free(lista);
        lista_len = 0;
}



/* === FunÃ§Ãµes auxiliares de networking ================================== */

char *sendACK(int S, char *buf, struct sockaddr *R)
/*
 * desc         :       Envia uma mensagem de confirmaÃ§Ã£o de recebimento para
 *                              o cliente.
 *
 * params       :       1.      Socket para envio da mensagem.
 *                              2.      String com a mensagem recebida.
 *                              3.      InformaÃ§Ãµes de rede sobre o cliente.
 *
 * output       :       Apontador para o primeiro caracter do conteÃºdo da mensa
gem
 *                              recebida.
 */
{
        char msg[BUFLEN];

        sprintf(msg, "ACK");
        strncat(msg, buf, (size_t)(6));

        if (sendto(S, msg, strlen(msg), 0, R,
        sizeof(struct sockaddr_storage)) == -1) {
                perror("sendto");
        }

        return &buf[6];
}

int bindUDP(char *port)
/*
 * desc         :       Associa a porta dada a um socket a ser retornado.
 *
 * params       :       1.      String contendo a porta a ser usada.
 *
 * output       :       O socket ao qual a porta foi associada.
 */
{
        int socket_fd;
```

```
        struct addrinfo hints, *servinfo, *p;
        int status;
        int optval = 1;

        /* inicialmente retornando erro */
        socket_fd = -1;

        /* estrutura a ser usada para obter um endereÃ§o IP */
        memset(&hints, 0, sizeof(hints));
        hints.ai_family = AF_INET;                      /* IPv4 */
        hints.ai_socktype = SOCK_DGRAM;         /* UDP */
        hints.ai_flags = AI_PASSIVE;

        /* obtem a lista ligada com as associaÃ§Ãµes possÃ-veis */
        if ((status = getaddrinfo(NULL, port, &hints, &servinfo)) != 0) {
                fprintf(stderr, "%s: getaddrinfo error: %s\n", __func__, gai_strerror(st
atus));
                exit(EXIT_FAILURE);
        }

        /* percorre servinfo tentando fazer o bind */
        for (p = servinfo; p != NULL; p = p->ai_next) {

                /* tenta obter um socket */
                if ((socket_fd = socket(p->ai_family, p->ai_socktype, p->ai_prot
ocol)) == -1) {
                        perror("socket");
                        continue;
                }

                /* configura o socket para permitir reuso do endereÃ§o IP */
                if (setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &optval, siz
eof(int)) == -1) {
                        perror("setsockopt");
                        exit(EXIT_FAILURE);
                }

                /* tenta associar o endereÃ§o IP obtido de servinfo com o socket
 */
                if (bind(socket_fd, p->ai_addr, p->ai_addrlen) == -1) {
                        close(socket_fd);
                        perror("bind");
                        sleep(1);
                        continue;
                }

                break;
        }

        /* tratamento de erro caso o bind falhe */
        if (p == NULL) {
                pMyError(BIND_ERROR, __func__);
                exit(EXIT_FAILURE);
        }

        freeaddrinfo(servinfo);

        return socket_fd;
}

void serverReady(int S, struct sockaddr *R)
{
        char msg[MAXSTRLEN];

        memset(msg, 0, sizeof(msg));
        sprintf(msg, "SERVER_READY\n\r");

        if (sendto(S, msg, strlen(msg), 0, R,
        sizeof(struct sockaddr_storage)) == -1) {
```



```
                perror("sendto");
        }
}


/* === Interpretados de comandos ========================================== */

int interpretador(char *cmd, int S, struct sockaddr *R)
/*
 * desc         :       Interpretador dos comandos recebidos pelo servidor.
 *
 * params       :       1.      Buffer contendo a linha de comando.
 *                              2.      Socket para o qual respostas sÃ£o enviad
as.
 *                              3.      InformaÃ§Ãµes de rede sobre o cliente.
 *
 * output       :       0 para continuar e -1 para encerrar a conexÃ£o.
 */
{
        char msg[BUFLEN];
        char tmp[BUFLEN];
        char *tok;
        int id, nota;
        int i;

        if ((tok = strtok(cmd, " \n\r")) == NULL) {
                return 0;
        }

        /* entrando a posiÃ§Ã£o do cliente */
        if (strcmp(tok, "posicao") == 0) {

                if ((tok = strtok(NULL, ",")) == NULL) {
                        return 0;
                }
                else {
                        cli_posx = atoi(tok);
                }
                if ((tok = strtok(NULL, ",")) == NULL) {
                        return 0;
                }
                else {
                        cli_posy = atoi(tok);
                }

                /* tratamento de erro na entrada das coordenadas */
                if (POSLIMMIN > cli_posx || cli_posx > POSLIMMAX
                || POSLIMMIN > cli_posy || cli_posy > POSLIMMAX) {
                        strncpy(msg, "Coordenadas devem estar entre ", (size_t)(30));
                        memset(tmp, 0, sizeof(tmp));
                        sprintf(tmp, "%d", POSLIMMIN);
                        strncat(msg, tmp, strlen(tmp));
                        strncat(msg, " e ", (size_t)(3));
                        memset(tmp, 0, sizeof(tmp));
                        sprintf(tmp, "%d", POSLIMMAX);
                        strncat(msg, tmp, strlen(tmp));
                        strcat(msg, ".\n\r");
                        if (sendto(S, msg, strlen(msg), 0, R, sizeof(struct sock
addr_storage)) == -1) {
                                perror("sendto");
                        }
                        cli_posx = -1;
                        cli_posy = -1;
                }
                else {
                        strncpy(msg, "PosiÃ§Ã£o atual (", (size_t)(30));
                        memset(tmp, 0, sizeof(tmp));
                        sprintf(tmp, "%d", cli_posx);
```

```
                        strncat(msg, tmp, strlen(tmp));
                        strncat(msg, ",", (size_t)(3));
                        memset(tmp, 0, sizeof(tmp));
                        sprintf(tmp, "%d", cli_posy);
                        strncat(msg, tmp, strlen(tmp));
                        strcat(msg, ").\n\r");
                        if (sendto(S, msg, strlen(msg), 0, R, sizeof(struct sock
addr_storage)) == -1) {
                                perror("sendto");
                        }
                }
        }
        else
        /* requisitando a lista das categorias existentes */
        if (strcmp(tok, "categorias") == 0) {
                i = listarCategorias();
                sendBusca(i, S, R, 1);
        }
        else
        /* requisitando alguma listagem de estabelecimentos */
        if (strcmp(tok, "buscar") == 0 || strcmp(tok, "listar") == 0) {

                if ((tok = strtok(NULL, "\n\r")) == NULL) {
                        return 0;
                }

                /* listar todos os estabelecimentos */
                if (strcmp(tok, "todos") == 0) {

                        i = buscar(-1, -1, NULL);
                        sendBusca(i, S, R, 0);
                }
                else
                if (strcmp(tok, "perto") == 0) {

                        if ((tok = strtok(NULL, "\n\r")) == NULL) {
                                return 0;
                        }
                        /* listar todos os estabelecimentos a 100m */
                        if (strcmp(tok, "todos") == 0) {

                                if (cli_posx == -1 || cli_posy == -1) {
                                        strcpy(msg, "Informe sua posiÃ§Ã£o antes.\n\r");
                                        if (sendto(S, msg, strlen(msg), 0, R, si
zeof(struct sockaddr_storage)) == -1) {
                                                perror("sendto");
                                        }
                                }
                                else {
                                        i = buscar(cli_posx, cli_posy, NULL);
                                        sendBusca(i, S, R, 0);
                                }
                        }
                        else
                        /* listar todos os estabelecimentos de um categoria
                         * e que estejam a menos de 100m */
                        if (strcmp(tok, "categoria") == 0) {

                                if ((tok = strtok(NULL, "\n\r")) == NULL) {
                                        return 0;
                                }

                                if (cli_posx == -1 || cli_posy == -1) {
                                        strcpy(msg, "Informe sua posiÃ§Ã£o antes.\n\r");
                                        if (sendto(S, msg, strlen(msg), 0, R, si
zeof(struct sockaddr_storage)) == -1) {
                                                perror("sendto");
                                        }
                                }
```



```
                                else {
                                        i = buscar(cli_posx, cli_posy, tok);
                                        sendBusca(i, S, R, 0);
                                }
                        }
                }
                else
                /* listar todos os estabelecimentos de um categoria */
                if (strcmp(tok, "categoria") == 0) {

                        if ((tok = strtok(NULL, "\n\r")) == NULL) {
                                return 0;
                        }

                        i = buscar(-1, -1, tok);
                        sendBusca(i, S, R, 0);
                }
        }
        else
        /* requisitando informaÃ§Ãµes acerca do id */
        if (strcmp(tok, "info") == 0) {
                if ((tok = strtok(NULL, ",")) != NULL) {
                        id = atoi(tok);
                }
                sendInfoID(id, S, R);
        }
        else
        /* votando no id */
        if (strcmp(tok, "votar") == 0) {

                if ((tok = strtok(NULL, "\n\r")) == NULL) {
                        return 0;
                }
                else {
                        id = atoi(tok);
                }
                if ((tok = strtok(NULL, "\n\r")) == NULL) {
                        return 0;
                }
                else {
                        nota = atoi(tok);
                }

                /* tratamento de erro para nota fora do intervalo */
                if (0 > nota || nota > 10 ) {
                        sprintf(msg, "Nota deve estar entre 0 e 10.\n\r");
                        if (sendto(S, msg, strlen(msg), 0, R, sizeof(struct sock
addr_storage)) == -1) {
                                perror("sendto");
                        }
                }
                else {
                        votarID(id, nota, S, R);
                }
        }
        else
        /* requisitando encerramento da conexÃ£o */
        if (strcmp(tok, "sair") == 0) {
                return -1;
        }
}



/* === FunÃ§Ãµes auxiliares de depuraÃ§Ã£o ================================
*/

void DEBUG_printItem(item_t *item)
{
```

```
        printf("*** item ***\n");
        printf("Id: %d\t", item->id);
        printf("PosiÃ§Ã£o: %d,%d\n", item->posx, item->posy);
        printf("Categoria: %s\t", item->categoria);
        printf("Nome: %s\n", item->nome);
        printf("EndereÃ§o: %s\n", item->endereco);
        printf("Pontos acumul.: %d\t", item->pontuacao);
        printf("Total de votos: %d\n", item->votos);
        printf("\n");
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>



#define BUFLEN 1024



/* === Tipos auxiliares ================================================= */

typedef enum
{
        FALSE = 0,
        TRUE = 1
} bool_t;



/* === Ferramentas para tratamento de erros ============================== */

typedef enum {
    NO_ERROR = 0,
    USAGE_ERROR,
    PORT_OUT_RANGE,
    CONNECT_ERROR,
    MYERROR_LIM

} my_error_t;

const char* my_error_desc[] =
{
    "",
    "usage: ./cliente_info_tcp <port number>",
    "error: port number must be between 1 and 8000",
    "error: failed to connect"
    ""
};

void pMyError(my_error_t e, const char *function)
{
    if(NO_ERROR < e && e < MYERROR_LIM) {
        fprintf(stderr, "%s: %s.\n", function, my_error_desc[e]);
    }
}



static int msg_seq;



/* ========================================================================
 * === Pragmas das funÃ§Ãµes ============================================
 * ======================================================================== */

/* === FunÃ§Ãµes auxiliares de networking ================================= */
int msgSeq(void);
int connectUDP(char *endIP, char *port);
```



```c
/* ========================================================================
 * === MAIN ===============================================================
 * ======================================================================== */

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

        fd_set rfds0, rfds1;
        struct timeval tv;

        int maxfd, rval;

        char msg[BUFLEN];
        char cmd[BUFLEN];
        char buf[BUFLEN];
        int len;

        int i, n;

        bool_t server_ready;

        fd_set tmp_fds;
        char ack_msg[9];
        struct timeval tv1;

        tv1.tv_sec = 1;
        tv1.tv_usec = 0;



        /* verificando argumentos */
        if (argc < 3) {
                pMyError(USAGE_ERROR, __func__);
                exit(EXIT_FAILURE);
        }

        i = atoi(argv[2]);
        if (0 >= i && i > 8000) {
                pMyError(PORT_OUT_RANGE, __func__);
                exit(EXIT_FAILURE);
        }

        sock = connectUDP(argv[1], argv[2]);

        maxfd = fileno(stdin);
        maxfd = (maxfd < sock) ? sock : maxfd;

        FD_ZERO(&rfds0);

        FD_SET(fileno(stdin), &rfds0);
        FD_SET(sock, &rfds0);

        tv.tv_sec = 0;
        tv.tv_usec = 0;

        memset(buf, 0, sizeof(buf));

        server_ready = TRUE;

        while (1) {

                rfds1 = rfds0;

        if ((rval = select(maxfd + 1, &rfds1, NULL, NULL, &tv)) == -1) {
                        perror("select");
                        exit(EXIT_FAILURE);
                }
```

```
                else if (rval > 0) {

                        if (FD_ISSET(fileno(stdin), &rfds1) && server_ready) {

                                memset(cmd, 0, sizeof(cmd));
                                fgets(cmd, BUFLEN, stdin);

                                /* encerra o cliente */
                                if (strcmp(cmd, "sair\n") == 0) {
                                        break;
                                }

                                /* etiqueta a mensagem com o nÃºmero de sequÃªnc
ia */
                                sprintf(msg, "%.6d", msgSeq());
                                strncat(msg, cmd, strlen(cmd));

                                if (send(sock, msg, strlen(msg), 0) == -1) {
                                        perror("send");
                                }

                                FD_ZERO(&tmp_fds);

                                /* espera pelo ACK do servidor */
                                while (1) {
                                        FD_SET(sock, &tmp_fds);
                                        i = select(sock+1, &tmp_fds, NULL, NULL,
 &tv1);

                                        if ( i > 0) {

                                                memset(buf, 0, sizeof(buf));
                                                if ((len = recv(sock, buf, 9, 0)
) == -1) {
                                                        perror("recv");
                                                }

                                                sprintf(ack_msg, "ACK%.6d", msg_
seq);
                                                if (strncmp(ack_msg, buf, (size_
t)(9)) == 0) {
                                                        break;
                                                }
                                        }
                                        else if (i <= 0) {
                                                puts("Sem resposta do servidor.");
                                                break;
                                        }
                                }

                                server_ready = FALSE;

                        }
                        if (FD_ISSET(sock, &rfds1)) {
                                memset(buf, 0, sizeof(buf));
                                if ((len = recv(sock, buf, BUFLEN, 0)) == -1) {
                                        perror("recv");
                                }

                                if (strstr(buf, "SERVER_READY\n\r") != NULL) {
                                        server_ready = TRUE;
                                }
                                else if (strcmp(buf, "\n") != 0 || strcmp(buf, "\
r") != 0) {
                                        printf("%s", buf);
                                }
                        }

                }
```



```
        }

        return 0;
}



/* ========================================================================
 * === ImplementaÃ§Ã£o das funÃ§Ãµes ========================================
 * ======================================================================== */

/* === FunÃ§Ãµes auxiliares de networking ================================== */

int msgSeq(void)
/*
 * desc         :       Retorna um inteiro entre 0 e 999 999 para ser usado como
 *                      identificador de mensagens.
 *
 * params       :       Nenhum.
 *
 * output       :       Inteiro (mod 1 000 000).
 */
 {
        if (++msg_seq > 999999) {
                msg_seq = 0;
        }

        return msg_seq;
}


int connectUDP(char *endIP, char *port)
/*
 * desc         :       Conecta a um endereÃ§o IP e porta a um socket a ser reto
rnado.
 *
 * params       :       1.      String contendo o endereÃ§o IP a ser usado.
 *                      2.      String contendo a porta a ser usada.
 *
 * output       :       O socket da conexÃ£o criada.
 */
{
        int socket_fd;
        struct addrinfo hints, *servinfo, *p;
        int status;

        /* inicialmente retornando erro */
        socket_fd = -1;

        /* estrutura a ser usada para obter um endereÃ§o IP */
        memset(&hints, 0, sizeof(hints));
        hints.ai_family = AF_INET;                      /* IPv4 */
        hints.ai_socktype = SOCK_DGRAM;         /* UDP */

        /* obtem a lista ligada com as associaÃ§Ãµes possÃ-veis */
        if ((status = getaddrinfo(endIP, port, &hints, &servinfo)) != 0) {
                fprintf(stderr, "%s: getaddrinfo error: %s\n", __func__, gai_strerror(st
atus));
                exit(EXIT_FAILURE);
        }

        /* percorre servinfo tentando se conectar */
        for (p = servinfo; p != NULL; p = p->ai_next) {

                /* tenta obter um socket */
                if ((socket_fd = socket(p->ai_family, p->ai_socktype, p->ai_prot
ocol)) == -1) {
                        perror("socket");
```

```
                        continue;
                }

                /* tenta se conectar ao endereÃ§o IP dado */
                if (connect(socket_fd, p->ai_addr, p->ai_addrlen) == -1) {
                        close(socket_fd);
                        perror("connect");
                        continue;
                }

                break;
        }

        /* tratamento de erro caso o bind falhe */
        if (p == NULL) {
                pMyError(CONNECT_ERROR, __func__);
                exit(EXIT_FAILURE);
        }

        freeaddrinfo(servinfo);

        return socket_fd;
}
```