```c
//ICSI 333. System Fundamentals
//Spring 2022
//Dutta Sourav
//Daniel St Andrews: 001530150

/*
Purpose:
    This program takes commands from the user to perform various operations
    on a string linkedlist that is printed to a file at the end of the run.

    Function Summaries and Parameter Descriptions:

    void ina(struct node **h, struct node **t, int num, char str[255]) - Adds a member into the list after a specified index.
        -struct node **h -> Address of the head of the list.
        -struct node **t -> Address of the tail of the list.
        -int num -> Index to insert after in the list
        -char str[255] -> The string to be inserted after the specified point in the list.

    void inb(struct node **h, struct node **t, int num, char str[255]) - Adds a member into the list before a specified index.
        -struct node **h -> Address of the head of the list.
        -struct node **t -> Address of the tail of the list.
        -int num -> Index to insert before in the list
        -char str[255] -> The string to be inserted before the specified point in the list.

    void rep(struct node* h, int pos, char new[255]) - Replaces the text at a specified index.
        -struct node *h -> Pointer to the head of the list.
        -int pos -> Index in the list to replace.
        -char new[255] -> The new string to replace the string at the specified index.

    void prn(struct node *head) - Prints the list to the screen.
        -struct node *head -> Pointer to the head of the list.

    void del(struct node **ph, int pos, char x[255]) - Deletes a node from the list and reorders indexes.
        -struct node **ph -> Address to the head of the list.
        -int pos -> Index that will be deleted in the list.
        -char x[255] -> String to be deleted from the list.

    void insert_node(struct node **h, struct node **t, char v[255]) - Inserts a node into the linked list.
        -struct node **h -> Address of the head of the list.
        -struct node **t -> Address of the tail of the list.
        -char str[255] -> The string to be inserted in the list.

    void delete_node(struct node **ph, char x[255]) - Deletes a node from the list without regard to index.
        -struct node **ph -> Address to the head of the list.
        -char x[255] -> String to be deleted from the list.

    void print_list(struct node *h) - Prints the list to the screen.
        -struct node *h -> Pointer to the head of the list

    void printToFile(struct node *h) - Prints the list to the output file.
        -struct node *h -> Pointer to the head of the list

    struct node* search_list(struct node *h, char str[255]) - Searches the list for a member and returns the node.
        -struct node *h -> Pointer to the head of the list.
        -char str[255] -> String to be searched for in the list.
    @return -> Returns either the node of the found member or NULL if the member is not in the list.

    struct node{}
        -char text[]
        -int index
        -struct node *next
*/

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

/* Struct for each node in the linked list. */
struct node {
    char text[255];
    int index;
    struct node *next;
};

/* Pointers to the first and last nodes of list are used */
/* to facilitate insertion at the end of list. */
struct node *head, *tail;

//Index Initializer for use in assignment.
int ind = -1;

//Prototypes
```

```
void insert_node (struct node **h, struct node **t, char v[255]);
void print_list(struct node *h);
void delete_node(struct node **ph, char x[255]);
void printToFile(struct node *h, FILE *out_file);

struct node *search_list(struct node *h, char x[255]);

void rep(struct node* h, int pos, char new[255]);
void prn(struct node *head);
void rearrange(struct node **h);
void ina(struct node **h, struct node **t, int num, char str[255]);
void inb(struct node **h, struct node **t, int num, char str[255]);
void del(struct node** h, struct node **t, int pos);


int main(void) {

    //Thought the list had to be written to output file but since it works cleanly I left it. Prints to file on end.
    FILE *out_file = fopen("out.txt", "w");

    //Number taken in for use in commands.
    int arg1;
    //Variable for use in while loop.
    int temp = -1;
    //Input string.
    char input[100];
    //End string.
    char end[4] = "end";
    //Command string.
    char command[4];

    //Initialize Variables for LinkedList
    head = tail = NULL;

    //Command prompt until end is read
    while (temp != 0) {

        printf("Command?"); fflush(stdout);
        scanf("%s", command);
        if (strcmp(command, end) == 0)
        {
            printToFile(head, out_file);
            fclose(out_file);
            break;
        }
        if (strcmp(command, "ina") == 0)
        {
            scanf("%d", &arg1);
            scanf("%s", input);
            ina(&head, &tail, arg1, input);
        }
        else if (strcmp(command, "inb") == 0)
        {
            scanf("%d", &arg1);
            scanf("%s", input);
            inb(&head, &tail, arg1, input);
        }
        else if (strcmp(command, "del") == 0)
        {
            scanf("%d", &arg1);
            del(&head, &tail, arg1);
        }
        else if (strcmp(command, "rep") == 0)
        {
            scanf("%d", &arg1);
            scanf("%s", input);
            rep(head, arg1, input);
        }
        else if (strcmp(command, "prn") == 0)
        {
            prn(head);
        }
    }
    return 0;
} /* End of main. */

void rearrange(struct node **h)
{
    int i = 0;
    struct node *temp;
    temp = *h;
    while (temp != NULL)
    {
        temp->index = ++i;
        temp = temp->next;
    }

}

void ina(struct node **h, struct node **t, int num, char str[255])
{

    //New Empty Node
    struct node *newNode;
    if ((newNode = (struct node *)malloc(sizeof(struct node))) == NULL) {
        printf("Node allocation failed. \n");fflush(stdout);
        exit(1);
    }
    strcpy(newNode->text, str);
    newNode->next = NULL;

    int cond = -1;

    if (*h == NULL)
    {
        insert_node(h, t, str);
        printf("Ok.\n");fflush(stdout);
    }
    else
    {
        if (search_list(*h, str) == NULL)
        {
            //Index is not in the list.
            if ((*t)->index < num)
            {
                insert_node(h, t, str);
                printf("Item inserted at end of list.\n");fflush(stdout);
                return;
            }
            //Insert after the last member of the list.
            else if ((*t)->index == num)
            {
                insert_node(h, t, str);
                printf("Ok.\n");fflush(stdout);
                return;
            }

            struct node* temp = *h;

            //Needs to go to specified point in list.
            while (num--)
            {
                temp = temp->next;

            }
            newNode->next = temp->next;
            newNode->index = temp->index;
            newNode->index++;
            temp->next = newNode;
            printf("Ok.\n");fflush(stdout);
            rearrange(h);

        }
        else
        {
            printf("Such text exists already. \n");fflush(stdout);
        }
    }
}

void inb(struct node **h, struct node **t, int num, char str[255])
{
    struct node *newNode;
    if ((newNode = (struct node *)malloc(sizeof(struct node))) == NULL) {
        printf("Node allocation failed. \n");fflush(stdout);
        exit(1);
    }
    strcpy(newNode->text, str);
    newNode->next = NULL;

    if (*h == NULL)
    {
        insert_node(h, t, str);
        printf("Ok. \n");fflush(stdout);
    }
    else
    {
        if (search_list(*h, str) == NULL)
        {
            if (num == 1 || num == 0)
            {
                struct node* temp;
                temp = *h;
                newNode->next = temp;
                *h = newNode;
                printf("Ok. \n");fflush(stdout);
                rearrange(h);
                return;
            }
            if ((*t)->index < num)
            {
                struct node* temp;
                temp = *h;
                newNode->next = temp;
                *h = newNode;
                printf("Text inserted at beginning. \n");fflush(stdout);
                rearrange(h);
            }
            else
            {
                struct node *temp, *temp2, *prev;
                temp = *h;

                while (temp->index != num && temp != NULL)
                {
                    prev = temp;
                    temp = temp->next;
                }
                if ((temp2 = (struct node *)malloc(sizeof(struct node))) == NULL) {
                    printf("Node allocation failed. \n");fflush(stdout);
                    exit(1);
                }
                temp2->index = temp->index;
                strcpy(temp2->text, str);
                temp2->next = prev->next;
                prev->next = temp2;
                rearrange(h);
                printf("Ok. \n");fflush(stdout);
                return;

            }
        }
        else
        {
            printf("Such text exists already.\n");
        }

    }

}

void rep(struct node* h, int pos, char new[255])
{
    while (h != NULL) {

        if (h->index == pos) {

            if (strcmp(h->text, new) == 0) {

                printf("Such text exists already.");fflush(stdout);
                break;
            }
            else {

                strcpy(h->text, new);
                printf("Replaced.\n");fflush(stdout);
                return;
            }
        }
        else {

            h = h->next;
        }
    }
    printf("No such index.");
}

void del(struct node** h, struct node **t, int pos)
{
    struct node* temp;
    struct node* prev;
    temp = *h;
    if ((*t)->index < pos)
    {
        printf("No such index.");fflush(stdout);
    }
    else
    {
        if (temp != NULL && temp->index == pos)
        {
            *h = temp->next;
            free(temp);
            printf("Deleted.\n");fflush(stdout);
        }
        else
        {
            while (temp != NULL && temp->index != pos)
            {
                prev = temp;
                temp = temp->next;
            }
            prev->next = temp->next;
            free(temp);
            printf("Deleted.\n");fflush(stdout);
        }
        rearrange(h);
    }
}

void prn(struct node *head)
{
    print_list(head);
}

void printToFile(struct node *h, FILE *out_file)
{
    /* Prints the values stored in the nodes of the list */
    /* pointed to by h. */
    if (h == NULL) {
        fprintf(out_file, "The list is empty.\n");
    }
    else {
        fprintf(out_file, "Values in the list are:\n");
        while (h != NULL) {
            fprintf(out_file, "Index: %d %s\n", h->index, h->text);
            h = h->next;
        }
    }
}

void insert_node (struct node **h, struct node **t, char v[255]) {
    /* Creates a new node with value given by parameter v */
    /* and inserts that node at the end of the list whose */
    /* first and last nodes are pointed to by *h and *t */
    /* respectively. */

    struct node *temp;
    if ((temp = (struct node *)malloc(sizeof(struct node))) == NULL) {
        printf("Node allocation failed. \n");fflush(stdout);
        exit(1); /* Stop program */
    }

    /* Space for node obtained. Copy the value v into the node */
    /* and insert the node at the end of the list. */
    strcpy(temp->text, v);
    temp->next = NULL;

    if (*h == NULL) {
        /* List is currently empty. */
        *h = *t = temp;
        ind = 1;
        (*h)->index = ind;
    }
    else { /* The list is not empty. Use *t to add the node */
        /* at the end. */
        (*t)->next = temp;
        *t = (*t)->next;
        ind++;
        (*t)->index = ind;
    }
} /* End of insert_node. */
```

```c
void print_list(struct node *h) {
    /* Prints the values stored in the nodes of the list */
    /* pointed to by h. */
    if (h == NULL) {
        printf("The list is empty.\n");fflush(stdout);
    }
    else {
        printf("Values in the list are:\n");fflush(stdout);
        while (h != NULL) {
            printf("Index: %d %s\n", h->index, h->text);fflush(stdout);
            h = h->next;
        }
    }
} /* End of print_list */

struct node *search_list(struct node *h, char x[255]) {
    /* Returns a pointer to the first node which contains the value */
    /* given by x. If there is no such node, the function returns */
    /* NULL. */
    while (h != NULL) {
        if (strcmp(h->text, x) == 0)
            return h;
        h = h->next;
    }
    /* Here, there is no node with value x. */
    return NULL;
} /* End of search_list */

void delete_node (struct node **ph, char x[255]) {
    struct node *temp = *ph;
    struct node *prev;

    if (temp != NULL && strcmp(temp->text, x) == 0) {
        *ph = temp->next;
        free(temp);
        return;
    }
    while (temp != NULL && strcmp(temp->text, x) != 0) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        return;
    }
    prev->next = temp->next;
    free(temp);
}/* End of delete_node */
```