E:\to do list.c

```c
// C program for the above approach
#include <stdio.h>
#include <stdlib.h>

// Renaming structure to avoid the
// repetitive use of struct keyword
typedef struct ToDo todo;

// Declaration of structure
struct ToDo {
    // char array as data part
    char buffer[101];

    // Pointer part to access addresses
    todo* next;

    // Count variable for counting
    // the number of nodes
    int count;
};

// Declare start pointer as null in
// the beginning
todo* start = NULL;

// Driver Code
int main()
{
    int choice;
    interface();

    while (1) {

        // Change console color and
        // text color
        system("Color 3F");

        // Clear the console
        system("cls");

        printf("1. To see your ToDo list\n");
        printf("2. To create new ToDo\n");
        printf("3. To delete your ToDo\n");
        printf("4. Exit");
        printf("\n\n\nEnter your choice\t:\t");

        // Choice from the user
        scanf("%d", &choice);

        switch (choice) {

        // Calling functions defined
        // below as per the user input
        case 1:
            seetodo();
            break;
        case 2:
            createtodo();
            break;
        case 3:
            deletetodo();
            break;
        case 4:
            exit(1);
            break;
        default:
            printf("\nInvalid Choice :-(\n");
            system("pause");
        }
    }
    return 0;
}

// Code for Splash screen
void interface()
{
    system("color 4F");
    printf("\n\n\n\n");
    printf("\t~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~\n");
    printf("\t~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~\n\n");
    printf("\t} : } : } : } : } : } "
        ": } : } : } : "
        "WELCOME TO the TODO APP "
        "    : { : { : { : { : { "
        ": { : { : { : {\n\n");
    printf("\t~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~\n");
    printf("\t~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~~~~~~~~~~~~~~~"
        "~~~~~~~~~~\n");
    printf("\n\n\n\t\t\t\t\t\t\t\"
        "t\t\t\t    "
        "@Sushant_Gaurav\n\n\n\n"
        "\n\n\n\t");

    // Pausing screen until user
    // presses any key
```

```c
    system("pause");
}

// To view all the todos
void seetodo()
{
    // Clearing the console
    system("cls");

    // Pointer to the node for traversal
    todo* temp;

    // temp is made to point the
    // start of linked list
    temp = start;

    // Condition for empty linked list
    if (start == NULL)
        printf("\n\nEmpty ToDo \n\n");

    // Traverse until last node
    while (temp != NULL) {

        // Print number of the node
        printf("%d.)", temp->count);

        // Print data of the node
        puts(temp->buffer);

        // Clear output console
        fflush(stdin);

        // Going to next node
        temp = temp->next;
    }

    printf("\n\n\n");
    system("pause");
}

// Function to insert a node todo
void createtodo()
{
    // Choose choice from user
    char c;

    // Pointers to node
    todo *add, *temp;
    system("cls");

    // Infinite loop which will
    // break if "n" is pressed
    while (1) {

        printf("\nWant to add new ToDo ??"
            + " Press 'y' for Yes and 'n' "
            + " for No :-)\n\t\t");
        fflush(stdin);

        // Input from user
        scanf("%c", &c);

        if (c == 'n')
            break;
        else {

            // If start node is NULL
            if (start == NULL) {

                // Dynamically allocating
                // memory to the newly
                // created node
                add = (todo*)calloc(1, sizeof(todo));

                // Using add pointer to
                // create linked list
                start = add;
                printf("\nType it.....\n");

                // Input from user
                fflush(stdin);
                gets(add->buffer);

                // As first input so
                // count is 1
                add->count = 1;

                // As first node so
                // start's next is NULL
                start->next = NULL;
            }
            else {
                temp = (todo*)calloc(1, sizeof(todo));
                printf("\nType it.....\n");
                fflush(stdin);
                gets(temp->buffer);

                // Insertion is at last
                // so pointer part is NULL
                temp->next = NULL;

                // add is now pointing
                // newly created node
                add->next = temp;
                add = add->next;
            }

            // Using the concept of
            // insertion at the end,
            // adding a todo

            // Calling function to adjust
            // the count variable
            adjustcount();
        }
    }
}

// Function to delete the todo
void deletetodo()
{
    system("cls");

    // To get the numbering of the
    // todo to be deleted
    int x;

    todo *del, *temp;
    printf("\nEnter the ToDo's number"
        + " that you want to remove.\n\t\t");

    // Checking empty condition
    if (start == NULL)
        printf("\n\nThere is no ToDo"
            + " for today :-)\n\n\n");
    else {
        scanf("%d", &x);

        // del will point to start
        del = start;

        // temp will point to start's
        // next so that traversal and
        // deletion is achieved easily
        temp = start->next;

        // Running infinite loop so
        // that user can delete and
        // asked again for choice
        while (1) {

            // When the values matches,
            // delete the node
            if (del->count == x) {

                // When the node to be
                // deleted is first node
                start = start->next;

                // Deallocating the memory
                // of the deleted node
                free(del);

                // Adjusting the count when
                // node is deleted
                adjustcount();
                break;
            }
```

```c

            if (temp->count == x) {
                del->next = temp->next;
                free(temp);
                adjustcount();
                break;
            }
            else {
                del = temp;
                temp = temp->next;
            }
        }
    }
    system("pause");
}

// Function to adjust the numbering
// of the nodes
void adjustcount()
{
    // For traversal, using
    // a node pointer
    todo* temp;
    int i = 1;
    temp = start;

    // Running loop until last node
    // and numbering it one by one
    while (temp != NULL) {
        temp->count = i;
        i++;
        temp = temp->next;
    }
}

```