Laboratory 5

Title of the Laboratory Exercise: Linked Lists

Introduction and Purpose of Experiment

A Linked List is a linear form of Data Structure, which is dynamically allocated, and size of can be increased to any length, provided there's memory available to allocate it.

1. Aim and Objectives
   - To develop a Linked List for Strings and provide basic linked list operations
2. Pseudo Codes

```
1. newLink
      1. Begin
      2. newList = malloc(sizeof *newList)
      3. newList -> next = NULL
      4. newList -> data = data
      5. return newList
      6. End
2. add_to_end_of_list
      1. Begin
      2. if (head == NULL) head = toad; return;
      3. toadd -> next = head
      4. head = toadd
      5. End
3. add_to_beg_of_list
      1. Begin
      2. if (head == NULL) head = toadd; return;
      3. add_to_beg_list(head -> next, toadd)
      4. End
4. remove_from_end_of_list
      1. Begin
      2. if (head == NULL) print "List is Empty"
      3. temp = head
      4. head = head -> next
      5. return temp
      6. End
5. print_list
      1. Begin
      2. if (head == NULL) return;
      3. print_list(head -> next)
      4. print head -> data
      5. End
```

3. Implementation in C

linked_list.h

```
struct LinkedList {
    void* data;
    struct LinkedList* next;
};
typedef struct LinkedList LinkedList;
```

linked_list.c

```
LinkedList* newLink(void * data) {
    LinkedList* newList = malloc(sizeof *newList);
    newList -> next = NULL; newList -> data = data; return newList;
}
void add_at_end_of_list(LinkedList** head, LinkedList* toadd) {
    if (*head == NULL) {*head = toadd; return;} toadd -> next = (*head); *head = toadd;
}
void add_at_beg_of_list(LinkedList** head, LinkedList* toadd) {
    if (*head == NULL) {*head = toadd;return;} add_at_beg_of_list(&((*head) -> next), toadd);
}
LinkedList* remove_from_end_of_list(LinkedList** head) {
    if (*head == NULL) printf("\nList is Empty !\n");
    LinkedList* temp = *head; *head = (*head) -> next; return temp;
}
LinkedList* remove_from_beg_of_list(LinkedList** head) {
```

```
    if (*head == NULL) printf("\nList is Empty !\n");
    LinkedList* temp = *head; LinkedList* prev = NULL;
    while (temp -> next != NULL) { prev = temp; temp = temp -> next;}
    if (prev == NULL) return NULL; prev -> next = NULL; return temp;
}
void print_list(LinkedList* head) {
    if (head == NULL) {return;} print_list(head -> next); ds((char*)(head -> data));
}
```

main.c

```
void menu();
int main() {
    LinkedList* head = NULL;
    char* input;
    while(1) {menu();int choice = next_int();
        switch (choice) {
            case 1:printf("\nEnter your Data : ");get_line(&input);
                add_at_end_of_list(&head, newLink(input));break;
            case 2:printf("\nEnter your Data : ");get_line(&input);
                add_at_beg_of_list(&head, newLink(input));break;
            case 3: {LinkedList *deleted = remove_from_end_of_list(&head);
                if (deleted != NULL) {printf("\nDeleted Link : ");print_link(deleted);}}
                break;
            case 4: {LinkedList *deleted = remove_from_beg_of_list(&head);
                if (deleted != NULL) {printf("\nDeleted Link : ");print_link(deleted);}}
                break;
            case 5: print_list(head);break;
            case 6:return 0;
            default:printf("\nInvalid Choice !\n");
          }
    }
}
void menu() {
    printf("\n---------- Linked Lists ----------------------\n"
     "1.\tAdd at the End of List\n" "2.\tAdd at the Beginning of List\n"
     "3.\tRemove from the End of the List\n" "4.\tRemove from the Beginning of the List\n"
     "5.\tDisplay the List\n" "6.\tExit\n" "\nYour Choice : "); }
```

4. Explanation

This is a menu driven program where Data can be added to the List at the beginning and at the end, the data/links can be removed from both beginning and end. The head pointer variable points to the beginning of the List. The Linked list implemented here is a singly linked list, hence the head pointer is very important. The user input is taken as a string and added to the list, although the list implemented here can take in any kind of data since it's a void*. The data is sent as a formal parameter to newLink which creates a new Link using malloc and returns it, this Link is then passes as a formal parameter to add to the list, either at the end or at the beginning. In add_at_end_of_list, the toAdd is made to point to what head was pointing to and the head pointer now points to the toAdd. In add_at_beg_of_list, a recursive function is used where the head -> next is passed as a formal parameter, when the value is found to be NULL the toAdd element is added there. Deletion works similarly, the element to be deleted from the linked list is found and the link before it is made to point to NULL and the deleted link is returned. When a link from the middle is to be removed then the link for the link to be deleted is made to point to the link that the toBeDeleted link was pointing to, pretty straight forward and simple. To print the linked list another recursive algorithm is used, since the structure itself is recursive in nature the algorithms too become recursive, to hence print, the head pointer is passes as the formal parameter, and head -> next is used a formal parameter inside the function, if it is NULL then nothing is printed, else the data stored in that link is printed.