LAB SESSION 4:

SHAYAN HUSSAIN CS-22100

1. Write a program that simulates a simple address book. Define a structure to store contact information (name, email, phone number). Allow the user to add new contacts to the address book dynamically. Use dynamic memory allocation for storing the contacts using malloc and update the memory allocation using realloc when adding new contacts. Implement an option to delete a contact and free the memory. Ensure that memory is properly managed throughout the program's execution.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_EMAIL_LENGTH 50
#define MAX_NAME_LENGTH 50
#define MAX_PHONE_LENGTH 15
struct Contact {
    char name[MAX_NAME_LENGTH];
    char email[MAX_EMAIL_LENGTH];
    char phone[MAX_PHONE_LENGTH];
};
void addContact(struct Contact **addressBook, int *numContacts) {
    *numContacts += 1;
    *addressBook = realloc(*addressBook, (*numContacts) * sizeof(struct Contact));
    struct Contact newContact;
    printf("Enter name: ");
    scanf("%s", newContact.name);
    printf("Enter email: ");
    scanf("%s", newContact.email);
    printf("Enter phone number: ");
    scanf("%s", newContact.phone);
```

```
    (*addressBook)[(*numContacts) - 1] = newContact;
    printf("Contact added successfully!\n");
}
void deleteContact(struct Contact **addressBook, int *numContacts) {
    if (*numContacts == 0) {
        printf("Address book is empty. No contacts to delete.\n");
        return;
    }
    char nameToDelete[MAX_NAME_LENGTH];
    printf("Enter the name of the contact to delete: ");
    scanf("%s", nameToDelete);
    int found = 0;
    for (int i = 0; i < *numContacts; i++) {
        if (strcmp((*addressBook)[i].name, nameToDelete) == 0) {
            found = 1;
            for (int j = i; j < *numContacts - 1; j++) {
                (*addressBook)[j] = (*addressBook)[j + 1];
            }
            *numContacts -= 1;
            *addressBook = realloc(*addressBook, (*numContacts) * sizeof(struct Contact));
            printf("Contact deleted successfully!\n");
            break;
        }
    }
    if (!found) {
        printf("Contact not found in the address book.\n");
    }
}
// Function to print all contacts in the address book
void printContacts(struct Contact *addressBook, int numContacts) {
  if (numContacts == 0) {
        printf("Address book is empty.\n");
        return;
```

```
    }
    printf("Contacts in the address book:\n");
    for (int i = 0; i < numContacts; i++) {
        printf("Contact %d:\n", i + 1);
        printf("Name: %s\n", addressBook[i].name);
        printf("Email: %s\n", addressBook[i].email);
        printf("Phone: %s\n", addressBook[i].phone);
        printf("\n");
    }
}
void freeMemory(struct Contact **addressBook) {
    free(*addressBook);
    *addressBook = NULL;
}
int main() {
    struct Contact *addressBook = NULL;
    int numContacts = 0;
    int choice;
    do {
        printf("Address Book Menu:\n");
        printf("1. Add a contact\n");
        printf("2. Delete a contact\n");
        printf("3. Print all contacts\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                addContact(&addressBook, &numContacts);
                break;
            case 2:
                deleteContact(&addressBook, &numContacts);
                break;
```

```
            case 3:
                printContacts(addressBook, numContacts);
                break;
            case 4:
                freeMemory(&addressBook);
                printf("Exiting the address book. Memory freed.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
                break;
        }
    } while (choice != 4);
    return 0;
}
```

OUTPUT:

```
Address Book Menu:
1. Add a contact
2. Delete a contact
3. Print all contacts
4. Exit
Enter your choice: 1
Enter name: SHAYAN
Enter email: shayanhussain268@gmail.com
Enter phone number: 03242123466
Contact added successfully!
Address Book Menu:
1. Add a contact
2. Delete a contact
3. Print all contacts
4. Exit
Enter your choice: 3
Contacts in the address book:
Contact 1:
Name: SHAYAN
Email: shayanhussain268@gmail.com
Phone: 03242123466
```

2. Write a C program to merge two sorted singly linked lists into a single sorted linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};
void append(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   struct Node* last = *head_ref;
   new_node->data = new_data;
   new_node->next = NULL;
   if (*head_ref == NULL) {
      *head_ref = new_node;
      return;
   }
   while (last->next != NULL)
      last = last->next;
   last->next = new_node;
}
struct Node* mergeLists(struct Node* head1, struct Node* head2) {
   struct Node* mergedList = NULL;
   if (head1 == NULL)
      return head2;
   else if (head2 == NULL)
      return head1;
   if (head1->data <= head2->data) {
      mergedList = head1;
      mergedList->next = mergeLists(head1->next, head2);
   } else {
      mergedList = head2;
```

```
        mergedList->next = mergeLists(head1, head2->next);
    }
    return mergedList;
}
void displayList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}
void freeList(struct Node* node) {
    struct Node* temp;
    while (node != NULL) {
        temp = node;
        node = node->next;
        free(temp);
    }
}
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    append(&list1, 2);
    append(&list1, 5);
    append(&list1, 7);
    append(&list2, 3);
    append(&list2, 6);
    append(&list2, 8);
    printf("First sorted linked list: ");
    displayList(list1);
    printf("Second sorted linked list: ");
```

```
    displayList(list2);
    struct Node* mergedList = mergeLists(list1, list2);
    printf("Merged sorted linked list: ");
    displayList(mergedList);
    freeList(mergedList);
    return 0;
}
```

OUTPUT:

```
First sorted linked list: 2 5 7
Second sorted linked list: 3 6 8
Merged sorted linked list: 2 3 5 6 7 8
```

3. Write a C program that converts a singly linked list into an array and returns it.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void append(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head_ref;
    new_node->data = new_data;
    new_node->next = NULL;
    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
}
```

```
int* convertLinkedListToArray(struct Node* head, int* size) {
    struct Node* temp = head;
    int count = 0;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    int* arr = (int*)malloc(count * sizeof(int));
    temp = head;
    int index = 0;
    while (temp != NULL) {
        arr[index++] = temp->data;
        temp = temp->next;
    }
    *size = count;
    return arr;
}
void displayArray(int* arr, int size) {
    printf("Array representation of the linked list: [ ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("]\n");
}
void freeLinkedList(struct Node* head) {
    struct Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
```

```
}
int main() {
   struct Node* head = NULL;
   append(&head, 2);
   append(&head, 5);
   append(&head, 7);
   int size;
   int* arr = convertLinkedListToArray(head, &size);
   displayArray(arr, size);
   freeLinkedList(head);
   free(arr);
   return 0;
}
```

OUTPUT:

```
Array representation of the linked list: [ 2 5
    7 ]
```

4. Write a C program that removes elements with odd indices from a singly linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};
void append(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   struct Node* last = *head_ref;
   new_node->data = new_data;
   new_node->next = NULL;
   if (*head_ref == NULL) {
      *head_ref = new_node;
      return;
```

```
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
}
void removeOddIndices(struct Node** head_ref) {
    struct Node* temp = *head_ref;
    struct Node* prev = NULL;
    int index = 0;
    while (temp != NULL) {
        if (index % 2 != 0) {
            if (prev == NULL) {
                *head_ref = temp->next;
                free(temp);
                temp = *head_ref;
            } else {
                prev->next = temp->next;
                free(temp);
                temp = prev->next;
            }
        } else {
            prev = temp;
            temp = temp->next;
        }
        index++;
    }
}
void displayList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
```

```
    }
    printf("\n");
}
void freeLinkedList(struct Node* head) {
    struct Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}
int main() {
    struct Node* head = NULL;
    append(&head, 1);
    append(&head, 2);
    append(&head, 3);
    append(&head, 4);
    append(&head, 5);
    append(&head, 6);
    printf("Original linked list: ");
    displayList(head);
    removeOddIndices(&head);
    printf("Linked list after removing elements with odd indices: ");
    displayList(head);
    freeLinkedList(head);
    return 0;
}
```

OUTPUT:

```
Original linked list: 1 2 3 4 5 6
Linked list after removing elements with odd
    indices: 1 3 5
```