Data Structures Practical List

1. Write a program to implement Selection Sort algorithm.

```
#include <stdio.h>

void selectionSort(int arr[], int n) {
    int i, j, min_idx, temp;
    for (i = 0; i < n-1; i++) {
        min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;
        temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

void printArray(int arr[], int n) {
    for (int i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

2. Write a program to implement Bubble sort algorithm.

```
#include <stdio.h>
```

```
void bubbleSort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1]) {
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}

void printArray(int arr[], int n) {
    for (int i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {5, 1, 4, 2, 8};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

3. Write a program to implement Insertion sort.

```
#include <stdio.h>

void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
```

```
        arr[j + 1] = key;
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    insertionSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

4. Write a program to implement Merge sort.

```
#include <stdio.h>

void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

    for (i = 0; i < n1; i++) L[i] = arr[l + i];
    for (j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    i = j = 0;
    k = l;
    while (i < n1 && j < n2)
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
```

```
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void printArray(int arr[], int n) {
    for (int i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
    mergeSort(arr, 0, n - 1);
    printArray(arr, n);
    return 0;
}
```

5. Write a program to implement Quick sort

```
#include <stdio.h>

void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high], i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
```

```
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int size) {
    for (int i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printArray(arr, n);
    return 0;
}
```

6. Write a program to implement Linear and Binary Search.

```
#include <stdio.h>

// Linear Search
int linearSearch(int arr[], int n, int x) {
    for (int i = 0; i < n; i++)
```

```
        if (arr[i] == x)
            return i;
    return -1;
}

// Binary Search (array must be sorted)
int binarySearch(int arr[], int l, int r, int x) {
    while (l <= r) {
        int m = l + (r - l) / 2;
        if (arr[m] == x)
            return m;
        if (arr[m] < x)
            l = m + 1;
        else
            r = m - 1;
    }
    return -1;
}

int main() {
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 10;
    int linResult = linearSearch(arr, n, x);
    int binResult = binarySearch(arr, 0, n - 1, x);

    printf("Linear Search: Element is at index %d\n", linResult);
    printf("Binary Search: Element is at index %d\n", binResult);
    return 0;
}
```

7. Write a Program to implement following operations on Singly Linked List:
    a. Insertion
    b. Deletion
    c. Search a given value

```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

// Insertion at End
void insert(struct Node** head, int data) {
    struct Node* newNode = malloc(sizeof(struct Node));
    struct Node* last = *head;
    newNode->data = data;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    while (last->next != NULL)
        last = last->next;
    last->next = newNode;
}

// Delete by Value
void deleteNode(struct Node** head, int key) {
    struct Node *temp = *head, *prev = NULL;
    if (temp != NULL && temp->data == key) {
        *head = temp->next;
        free(temp);
        return;
    }
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) return;
    prev->next = temp->next;
    free(temp);
```

```
}

// Search for a Value
int search(struct Node* head, int key) {
    while (head != NULL) {
        if (head→data == key)
            return 1;
        head = head→next;
    }
    return 0;
}

// Display
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d → ", node→data);
        node = node→next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 30);
    printList(head);

    deleteNode(&head, 20);
    printList(head);

    int found = search(head, 30);
    printf("Search for 30: %s\n", found ? "Found" : "Not Found");

    return 0;
}
```

8. Write a Program to implement Parenthesis Checker using Stack.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

char stack[MAX];
int top = -1;

void push(char c) {
    if (top < MAX - 1)
        stack[++top] = c;
}

char pop() {
    if (top >= 0)
        return stack[top--];
    return '\0';
}

int isMatchingPair(char opening, char closing) {
    return (opening == '(' && closing == ')') ||
           (opening == '{' && closing == '}') ||
           (opening == '[' && closing == ']');
}

int isBalanced(char* expr) {
    for (int i = 0; expr[i]; i++) {
        if (expr[i] == '(' || expr[i] == '{' || expr[i] == '[')
            push(expr[i]);
        else if (expr[i] == ')' || expr[i] == '}' || expr[i] == ']') {
            if (top == -1 || !isMatchingPair(pop(), expr[i]))
                return 0;
        }
    }
    return top == -1;
}

int main() {
```

```
    char expr[MAX];
    printf("Enter expression: ");
    scanf("%s", expr);
    printf("Expression is %s\n", isBalanced(expr) ? "Balanced" : "Not Balanced"
    return 0;
}
```

9. Write a program to convert Infix expression to Postfix expression.

```
#include <stdio.h>
#include <ctype.h>

#define MAX 100

char stack[MAX];
int top = -1;

int precedence(char op) {
    switch (op) {
        case '+': case '-': return 1;
        case '*': case '/': return 2;
        case '^': return 3;
        default: return 0;
    }
}

void push(char c) { stack[++top] = c; }
char pop() { return stack[top--]; }
char peek() { return stack[top]; }

void infixToPostfix(char* infix, char* postfix) {
    int i = 0, k = 0;
    char c;
    while ((c = infix[i++]) != '\0') {
        if (isalnum(c)) postfix[k++] = c;
        else if (c == '(') push(c);
        else if (c == ')') {
            while (top != -1 && peek() != '('
```

```
            postfix[k++] = pop();
        pop();
      }
      else {
        while (top != -1 && precedence(peek()) >= precedence(c))
          postfix[k++] = pop();
        push(c);
      }
   }
   while (top != -1) postfix[k++] = pop();
   postfix[k] = '\0';
}

int main() {
   char infix[MAX], postfix[MAX];
   printf("Enter infix expression: ");
   scanf("%s", infix);
   infixToPostfix(infix, postfix);
   printf("Postfix: %s\n", postfix);
   return 0;
}
```

10. Write a program to implement Circular Queue using array.

```
#include <stdio.h>
#define SIZE 5

int queue[SIZE], front = -1, rear = -1;

int isFull() {
   return (front == (rear + 1) % SIZE);
}

int isEmpty() {
   return (front == -1);
}

void enqueue(int val) {
```

```
    if (isFull())
        printf("Queue is Full\n");
    else {
        if (isEmpty())
            front = 0;
        rear = (rear + 1) % SIZE;
        queue[rear] = val;
    }
}

void dequeue() {
    if (isEmpty())
        printf("Queue is Empty\n");
    else {
        printf("Deleted: %d\n", queue[front]);
        if (front == rear)
            front = rear = -1;
        else
            front = (front + 1) % SIZE;
    }
}

void display() {
    if (isEmpty()) {
        printf("Queue is Empty\n");
        return;
    }
    printf("Queue: ");
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear) break;
        i = (i + 1) % SIZE;
    }
    printf("\n");
}

int main() {
```

```
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    enqueue(40);
    enqueue(50);
    enqueue(60);  // Should show full
    display();
    return 0;
}
```

11. Write a Program for Inorder, Preorder, Postorder and Level order traversal techniques.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node *left, *right;
};

struct Node* newNode(int data) {
    struct Node* node = malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

void inorder(struct Node* root) {
    if (root) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(struct Node* root) {
```

```
    if (root) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

void levelOrder(struct Node* root) {
    struct Node* queue[100];
    int front = 0, rear = 0;
    if (root) queue[rear++] = root;

    while (front < rear) {
        struct Node* temp = queue[front++];
        printf("%d ", temp->data);
        if (temp->left) queue[rear++] = temp->left;
        if (temp->right) queue[rear++] = temp->right;
    }
}

int main() {
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    printf("Inorder: "); inorder(root); printf("\n");
    printf("Preorder: "); preorder(root); printf("\n");
    printf("Postorder: "); postorder(root); printf("\n");
```

```
    printf("Level Order: "); levelOrder(root); printf("\n");

    return 0;
}
```

12. Write a program to implement Linear and Quadratic Probing

```
#include <stdio.h>
#define SIZE 10

int hashTable[SIZE];

void init() {
    for (int i = 0; i < SIZE; i++) hashTable[i] = -1;
}

void insertLinear(int key) {
    int index = key % SIZE;
    while (hashTable[index] != -1)
        index = (index + 1) % SIZE;
    hashTable[index] = key;
}

void insertQuadratic(int key) {
    int i = 0, index;
    while (i < SIZE) {
        index = (key + i * i) % SIZE;
        if (hashTable[index] == -1) {
            hashTable[index] = key;
            return;
        }
        i++;
    }
    printf("Table Full\n");
}

void display() {
    for (int i = 0; i < SIZE; i++)
```

```
        printf("%d ", hashTable[i]);
    printf("\n");
}

int main() {
    init();
    insertLinear(23);
    insertLinear(43);
    insertLinear(13);
    insertQuadratic(33);
    insertQuadratic(53);
    display();
    return 0;
}
```

13. Write a program to implement Linear Probing and Double Hashing

```
#include <stdio.h>
#define SIZE 10

int table[SIZE];

void init() {
    for (int i = 0; i < SIZE; i++) table[i] = -1;
}

int hash1(int key) { return key % SIZE; }
int hash2(int key) { return 7 - (key % 7); }

void insertLinear(int key) {
    int index = hash1(key);
    while (table[index] != -1)
        index = (index + 1) % SIZE;
    table[index] = key;
}

void insertDoubleHash(int key) {
    int i = 0, index;
```

```
    while (i < SIZE) {
        index = (hash1(key) + i * hash2(key)) % SIZE;
        if (table[index] == -1) {
            table[index] = key;
            return;
        }
        i++;
    }
    printf("Table Full\n");
}

void display() {
    for (int i = 0; i < SIZE; i++)
        printf("%d ", table[i]);
    printf("\n");
}

int main() {
    init();
    insertLinear(10);
    insertLinear(20);
    insertDoubleHash(30);
    insertDoubleHash(40);
    display();
    return 0;
}
```

14. Find the Winner of the Circular Game using Queue.

```
#include <stdio.h>

int findWinner(int n, int k) {
    int winner = 0;
    for (int i = 2; i <= n; i++)
        winner = (winner + k) % i;
    return winner + 1;
}
```

```
int main() {
    int n = 5, k = 2;
    printf("Winner is person %d\n", findWinner(n, k));
    return 0;
}
```

15. Write a program to implement stack using array and perform various operations on it.

```
#include <stdio.h>
#define SIZE 100

int stack[SIZE], top = -1;

void push(int val) {
    if (top == SIZE - 1) printf("Stack Overflow\n");
    else stack[++top] = val;
}

void pop() {
    if (top == -1) printf("Stack Underflow\n");
    else printf("Popped: %d\n", stack[top--]);
}

void peek() {
    if (top == -1) printf("Stack is Empty\n");
    else printf("Top Element: %d\n", stack[top]);
}

void display() {
    if (top == -1) printf("Stack is Empty\n");
    else {
        printf("Stack: ");
        for (int i = top; i >= 0; i--) printf("%d ", stack[i]);
        printf("\n");
    }
}

int main() {
```

```
    push(10);
    push(20);
    push(30);
    display();
    pop();
    peek();
    display();
    return 0;
}
```

16. Write a program to implement queue using array.

```
#include <stdio.h>
#define SIZE 100

int queue[SIZE], front = -1, rear = -1;

void enqueue(int val) {
    if (rear == SIZE - 1) printf("Queue Overflow\n");
    else {
        if (front == -1) front = 0;
        queue[++rear] = val;
    }
}

void dequeue() {
    if (front == -1 || front > rear) printf("Queue Underflow\n");
    else printf("Dequeued: %d\n", queue[front++]);
}

void display() {
    if (front == -1 || front > rear) printf("Queue is Empty\n");
    else {
        printf("Queue: ");
        for (int i = front; i <= rear; i++) printf("%d ", queue[i]);
        printf("\n");
    }
}
```

```
int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    display();
    return 0;
}
```

17. Write a program to perform merging of two sorted Link Lists. (SLL)

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = NULL;
    return newNode;
}

struct Node* mergeSorted(struct Node* l1, struct Node* l2) {
    if (!l1) return l2;
    if (!l2) return l1;

    if (l1->data < l2->data) {
        l1->next = mergeSorted(l1->next, l2);
        return l1;
    } else {
        l2->next = mergeSorted(l1, l2->next);
        return l2;
```

```
    }
}

void printList(struct Node* head) {
    while (head) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

int main() {
    struct Node* a = createNode(1);
    a->next = createNode(3);
    a->next->next = createNode(5);

    struct Node* b = createNode(2);
    b->next = createNode(4);
    b->next->next = createNode(6);

    struct Node* merged = mergeSorted(a, b);
    printf("Merged List: ");
    printList(merged);

    return 0;
}
```

18. Write a Program to implement following operations on Singly Linked List:
    a. Reverse the given link list
    b. Deletion
    c. Search a given value

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
```

```
    struct Node* next;
};

struct Node* head = NULL;

void insert(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = head;
    head = newNode;
}

// Deletion
void deleteNode(int val) {
    struct Node *temp = head, *prev = NULL;

    if (temp && temp->data == val) {
        head = temp->next;
        free(temp);
        return;
    }

    while (temp && temp->data != val) {
        prev = temp;
        temp = temp->next;
    }

    if (!temp) {
        printf("Value not found\n");
        return;
    }

    prev->next = temp->next;
    free(temp);
}

// Reversing the List
void reverseList() {
```

```
    struct Node *prev = NULL, *curr = head, *next;
    while (curr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
}

// Searching the List
void search(int val) {
    struct Node* temp = head;
    while (temp) {
        if (temp->data == val) {
            printf("Value %d found\n", val);
            return;
        }
        temp = temp->next;
    }
    printf("Value %d not found\n", val);
}

void printList() {
    struct Node* temp = head;
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    insert(10);
    insert(20);
    insert(30);
    printList();
```

```
    search(20);
    deleteNode(20);
    printList();

    reverseList();
    printList();

    return 0;
}
```

19. Write a Program to implement following operations on Singly Linked List:
    a. Sort the list
    b. Deletion
    c. Insertion

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* head = NULL;

void insert(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode→data = val;
    newNode→next = head;
    head = newNode;
}

// Deletion
void deleteNode(int val) {
    struct Node *temp = head, *prev = NULL;

    if (temp && temp→data == val) {
```

```
        head = temp->next;
        free(temp);
        return;
    }

    while (temp && temp->data != val) {
        prev = temp;
        temp = temp->next;
    }

    if (!temp) {
        printf("Value not found\n");
        return;
    }

    prev->next = temp->next;
    free(temp);
}

// Sorting
void sortList() {
    struct Node *i, *j;
    int temp;
    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}

// Displaying
void printList() {
    struct Node* temp = head;
    while (temp) {
```

```
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    insert(30);
    insert(10);
    insert(20);
    printList();

    deleteNode(10);
    printList();

    sortList();
    printList();

    insert(15);
    printList();

    return 0;
}
```