DSA Practicals

1. Array Operations (Menu Driven)

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

#define MAX 100

int arr[MAX];
int n = 0;

void insert() {
   if (n == MAX) {
      printf("Array is full.\n");
      return;
   }
   int val, pos;
   printf("Enter position (0 to %d): ", n);
   scanf("%d", &pos);
   if (pos < 0 || pos > n) {
      printf("Invalid position.\n");
      return;
   }
   printf("Enter value: ");
   scanf("%d", &val);
   for (int i = n; i > pos; i--)
      arr[i] = arr[i - 1];
   arr[pos] = val;
   n++;
   printf("%d inserted.\n", val);
}

void delete() {
   if (n == 0) {
      printf("Array is empty.\n");
      return;
   }
   int pos;
   printf("Enter position (0 to %d): ", n - 1);
   scanf("%d", &pos);
   if (pos < 0 || pos >= n) {
      printf("Invalid position.\n");
      return;
   }
   printf("Deleted element: %d\n", arr[pos]);
   for (int i = pos; i < n - 1; i++)
      arr[i] = arr[i + 1];
   n--;
}
```

```
void display() {
   if (n == 0) {
      printf("Array is empty.\n");
      return;
   }
   printf("Array elements: ");
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}

int main() {
   int choice;
   do {
      printf("\n--- ARRAY OPERATIONS MENU ---\n");
      printf("1. Insert\n");
      printf("2. Delete\n");
      printf("3. Display\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1: insert(); break;
         case 2: delete(); break;
         case 3: display(); break;
         case 4: printf("Exiting...\n"); exit(0);
         default: printf("Invalid choice.\n");
      }
   } while (choice != 4);
   return 0;
}
```

2. String Operations (Menu Driven)

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

char str[100];

void reverse() {
   int len = strlen(str);
   for (int i = 0; i < len / 2; i++) {
```

```
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }
    printf("Reversed string: %s\n", str);
}

void palindrome() {
    int len = strlen(str), flag = 1;
    for (int i = 0; i < len / 2; i++)
        if (tolower(str[i]) != tolower(str[len - i - 1]))
            flag = 0;
    if (flag)
        printf("The string is a palindrome.\n");
    else
        printf("The string is not a palindrome.\n");
}

void length() {
    printf("Length of string: %lu\n", strlen(str));
}

int main() {
    int choice;
    printf("Enter a string: ");
    gets(str);
    do {
        printf("\n--- STRING MENU ---\n");
        printf("1. Reverse\n");
        printf("2. Check Palindrome\n");
        printf("3. Find Length\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
```

```
        scanf("%d", &choice);
        switch (choice) {
            case 1: reverse(); break;
            case 2: palindrome(); break;
            case 3: length(); break;
            case 4: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice.\n");
        }
    } while (choice != 4);
    return 0;
}
```

3.Recursion Programs (Menu Driven)

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

int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}

int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int power(int a, int b) {
    if (b == 0) return 1;
    return a * power(a, b - 1);
}

int main() {
```

```
int choice;
do {
   printf("\n--- RECURSION MENU ---\n");
   printf("1. Factorial\n");
   printf("2. Fibonacci\n");
   printf("3. Power (a^b)\n");
   printf("4. Exit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);
   switch (choice) {
      case 1: {
         int n;
         printf("Enter n: ");
         scanf("%d", &n);
         printf("Factorial = %d\n", factorial(n));
         break;
      }
      case 2: {
         int n;
         printf("Enter n: ");
         scanf("%d", &n);
         printf("Fibonacci = %d\n", fibonacci(n));
         break;
      }
      case 3: {
         int a, b;
         printf("Enter base and exponent: ");
         scanf("%d %d", &a, &b);
         printf("Result = %d\n", power(a, b));
         break;
      }
      case 4: printf("Exiting...\n"); exit(0);
      default: printf("Invalid choice.\n");
```

```
        }
    } while (choice != 4);
    return 0;
}
```

3. Stack using Array (Menu Driven)

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

int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}

int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int power(int a, int b) {
    if (b == 0) return 1;
    return a * power(a, b - 1);
}

int main() {
    int choice;
    do {
        printf("\n--- RECURSION MENU ---\n");
        printf("1. Factorial\n");
        printf("2. Fibonacci\n");
        printf("3. Power (a^b)\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
```

```
        scanf("%d", &choice);
        switch (choice) {
            case 1: {
                int n;
                printf("Enter n: ");
                scanf("%d", &n);
                printf("Factorial = %d\n", factorial(n));
                break;
            }
            case 2: {
                int n;
                printf("Enter n: ");
                scanf("%d", &n);
                printf("Fibonacci = %d\n", fibonacci(n));
                break;
            }
            case 3: {
                int a, b;
                printf("Enter base and exponent: ");
                scanf("%d %d", &a, &b);
                printf("Result = %d\n", power(a, b));
                break;
            }
            case 4: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice.\n");
        }
    } while (choice != 4);
    return 0;
}
```

4. Stack using Array (Menu Driven)

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

```
#include <stdlib.h>

#define MAX 5
int stack[MAX], top = -1;

void push() {
   int val;
   if (top == MAX - 1)
      printf("Stack Overflow.\n");
   else {
      printf("Enter element to push: ");
      scanf("%d", &val);
      stack[++top] = val;
      printf("%d pushed.\n", val);
   }
}

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

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

```
int main() {
   int choice;
   do {
      printf("\n--- STACK MENU ---\n");
      printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
      printf("Enter choice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1: push(); break;
         case 2: pop(); break;
         case 3: display(); break;
         case 4: printf("Exiting...\n"); exit(0);
         default: printf("Invalid choice.\n");
      }
   } while (choice != 4);
   return 0;
}
```

5.Queue using Array (Menu Driven)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5

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

void enqueue() {
   int val;
   if (rear == MAX - 1)
      printf("Queue Overflow.\n");
   else {
      if (front == -1) front = 0;
      printf("Enter element: ");
```

```
        scanf("%d", &val);
        queue[++rear] = val;
        printf("%d enqueued.\n", 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 elements: ");
        for (int i = front; i <= rear; i++)
            printf("%d ", queue[i]);
        printf("\n");
    }
}

int main() {
    int choice;
    do {
        printf("\n--- QUEUE MENU ---\n");
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: enqueue(); break;
```

```
            case 2: dequeue(); break;
            case 3: display(); break;
            case 4: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice.\n");
        }
    } while (choice != 4);
    return 0;
}



/*==========================
  STACK USING LINKED LIST
  File: stack_linkedlist_menu.c
==========================*/
#include <stdio.h>
#include <stdlib.h>

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

Node* top = NULL;

void push() {
    int x;
    printf("Enter element to push: ");
    scanf("%d", &x);
    Node* t = (Node*)malloc(sizeof(Node));
    if (!t) { printf("Memory allocation failed.\n"); return; }
    t->data = x;
    t->next = top;
    top = t;
```

```
    printf("%d pushed.\n", x);
}

void pop() {
    if (top == NULL) {
        printf("Stack Underflow.\n");
        return;
    }
    Node* t = top;
    printf("Popped: %d\n", t->data);
    top = top->next;
    free(t);
}

void peek() {
    if (top == NULL) printf("Stack is empty.\n");
    else printf("Top element: %d\n", top->data);
}

void display() {
    if (top == NULL) { printf("Stack is empty.\n"); return; }
    printf("Stack (top to bottom):\n");
    for (Node* p = top; p; p = p->next) printf("%d\n", p->data);
}

int main() {
    int choice;
    do {
        printf("\n--- STACK (LINKED LIST) MENU ---\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Peek\n");
        printf("4. Display\n");
```

```
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: peek(); break;
            case 4: display(); break;
            case 5: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice.\n");
        }
    } while (choice != 5);
    return 0;
}



/*==========================
  QUEUE USING LINKED LIST
  File: queue_linkedlist_menu.c
==========================*/
#include <stdio.h>
#include <stdlib.h>

typedef struct QNode {
    int data;
    struct QNode* next;
} QNode;

QNode *front = NULL, *rear = NULL;

void enqueue() {
    int x;
    printf("Enter element to enqueue: ");
```

```
    scanf("%d", &x);
    QNode* t = (QNode*)malloc(sizeof(QNode));
    if (!t) { printf("Memory allocation failed.\n"); return; }
    t->data = x; t->next = NULL;
    if (rear == NULL) front = rear = t;
    else { rear->next = t; rear = t; }
    printf("%d enqueued.\n", x);
}

void dequeue() {
    if (front == NULL) { printf("Queue Underflow.\n"); return; }
    QNode* t = front;
    printf("Dequeued: %d\n", t->data);
    front = front->next;
    if (front == NULL) rear = NULL;
    free(t);
}

void peek() {
    if (front == NULL) printf("Queue is empty.\n");
    else printf("Front element: %d\n", front->data);
}

void display() {
    if (front == NULL) { printf("Queue is empty.\n"); return; }
    printf("Queue elements: ");
    for (QNode* p = front; p; p = p->next) printf("%d ", p->data);
    printf("\n");
}

int main() {
    int choice;
    do {
```

```
        printf("\n--- QUEUE (LINKED LIST) MENU ---\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Peek\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: enqueue(); break;
            case 2: dequeue(); break;
            case 3: peek(); break;
            case 4: display(); break;
            case 5: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice.\n");
        }
    } while (choice != 5);
    return 0;
}



/*==========================
  SINGLY LINKED LIST (SLL)
  File: sll_menu.c
==========================*/
#include <stdio.h>
#include <stdlib.h>

typedef struct SNode {
    int data;
    struct SNode* next;
} SNode;
```

```
SNode* head = NULL;

SNode* new_node(int x) {
   SNode* t = (SNode*)malloc(sizeof(SNode));
   if (!t) { printf("Memory allocation failed.\n"); exit(0); }
   t->data = x; t->next = NULL; return t;
}

void insert_begin() {
   int x; printf("Enter value: "); scanf("%d", &x);
   SNode* t = new_node(x);
   t->next = head; head = t;
   printf("Inserted at beginning.\n");
}

void insert_end() {
   int x; printf("Enter value: "); scanf("%d", &x);
   SNode* t = new_node(x);
   if (!head) head = t;
   else { SNode* p = head; while (p->next) p = p->next; p->next = t; }
   printf("Inserted at end.\n");
}

void insert_pos() {
   int pos, x; printf("Enter position (0-based) and value: ");
   scanf("%d %d", &pos, &x);
   if (pos < 0) { printf("Invalid position.\n"); return; }
   if (pos == 0) { SNode* t = new_node(x); t->next = head; head = t; printf("Inserted.\n"); return; }
   SNode* p = head; for (int i = 0; p && i < pos - 1; i++) p = p->next;
   if (!p) { printf("Invalid position.\n"); return; }
   SNode* t = new_node(x); t->next = p->next; p->next = t; printf("Inserted.\n");
}
```

```
void delete_pos() {
   int pos; printf("Enter position (0-based): "); scanf("%d", &pos);
   if (!head || pos < 0) { printf("Invalid.\n"); return; }
   if (pos == 0) { SNode* t = head; head = head->next; printf("Deleted: %d\n", t->data); free(t); return; }
   SNode* p = head; for (int i = 0; p && i < pos - 1; i++) p = p->next;
   if (!p || !p->next) { printf("Invalid position.\n"); return; }
   SNode* t = p->next; p->next = t->next; printf("Deleted: %d\n", t->data); free(t);
}

void delete_value() {
   if (!head) { printf("List empty.\n"); return; }
   int x; printf("Enter value to delete (first occurrence): "); scanf("%d", &x);
   SNode *p = head, *prev = NULL;
   while (p && p->data != x) { prev = p; p = p->next; }
   if (!p) { printf("Value not found.\n"); return; }
   if (!prev) head = p->next; else prev->next = p->next;
   printf("Deleted: %d\n", p->data); free(p);
}

void search() {
   int x; printf("Enter value to search: "); scanf("%d", &x);
   int idx = 0;
   for (SNode* p = head; p; p = p->next, idx++)
      if (p->data == x) { printf("Found at index %d\n", idx); return; }
   printf("Not found.\n");
}

void reverse_iter() {
   SNode *prev = NULL, *cur = head, *nxt;
   while (cur) { nxt = cur->next; cur->next = prev; prev = cur; cur = nxt; }
   head = prev; printf("List reversed (iterative).\n");
}
```

```
void display() {
    if (!head) { printf("List is empty.\n"); return; }
    printf("List: ");
    for (SNode* p = head; p; p = p->next) printf("%d -> ", p->data);
    printf("NULL\n");
}

int main() {
    int choice;
    do {
        printf("\n--- SINGLY LINKED LIST MENU ---\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete at Position\n");
        printf("5. Delete by Value\n");
        printf("6. Search\n");
        printf("7. Reverse (Iterative)\n");
        printf("8. Display\n");
        printf("9. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: insert_begin(); break;
            case 2: insert_end(); break;
            case 3: insert_pos(); break;
            case 4: delete_pos(); break;
            case 5: delete_value(); break;
            case 6: search(); break;
            case 7: reverse_iter(); break;
            case 8: display(); break;
            case 9: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice.\n");
```

```
        }
    } while (choice != 9);
    return 0;
}



/*==========================
  DOUBLY LINKED LIST (DLL)
  File: dll_menu.c
==========================*/
#include <stdio.h>
#include <stdlib.h>

typedef struct DNode {
    int data;
    struct DNode* prev;
    struct DNode* next;
} DNode;

DNode* head = NULL;

DNode* dnew(int x) {
    DNode* t = (DNode*)malloc(sizeof(DNode));
    if (!t) { printf("Memory allocation failed.\n"); exit(0); }
    t->data = x; t->prev = t->next = NULL; return t;
}

void insert_front() {
    int x; printf("Enter value: "); scanf("%d", &x);
    DNode* t = dnew(x);
    t->next = head;
    if (head) head->prev = t;
    head = t;
```

```
    printf("Inserted at front.\n");
}

void insert_back() {
    int x; printf("Enter value: "); scanf("%d", &x);
    DNode* t = dnew(x);
    if (!head) head = t;
    else {
        DNode* p = head; while (p->next) p = p->next;
        p->next = t; t->prev = p;
    }
    printf("Inserted at back.\n");
}

void insert_pos() {
    int pos, x; printf("Enter position (0-based) and value: ");
    scanf("%d %d", &pos, &x);
    if (pos < 0) { printf("Invalid position.\n"); return; }
    if (pos == 0) { insert_front(); return; }
    DNode* p = head;
    for (int i = 0; p && i < pos - 1; i++) p = p->next;
    if (!p) { printf("Invalid position.\n"); return; }
    DNode* t = dnew(x);
    t->next = p->next; t->prev = p;
    if (p->next) p->next->prev = t;
    p->next = t;
    printf("Inserted.\n");
}

void delete_pos() {
    int pos; printf("Enter position (0-based): "); scanf("%d", &pos);
    if (!head || pos < 0) { printf("Invalid.\n"); return; }
    DNode* p = head;
```

```
    if (pos == 0) {
        head = head->next;
        if (head) head->prev = NULL;
        printf("Deleted: %d\n", p->data); free(p); return;
    }
    for (int i = 0; p && i < pos; i++) p = p->next;
    if (!p) { printf("Invalid position.\n"); return; }
    if (p->prev) p->prev->next = p->next;
    if (p->next) p->next->prev = p->prev;
    printf("Deleted: %d\n", p->data); free(p);
}

void delete_value() {
    if (!head) { printf("List empty.\n"); return; }
    int x; printf("Enter value to delete (first occurrence): "); scanf("%d", &x);
    DNode* p = head;
    while (p && p->data != x) p = p->next;
    if (!p) { printf("Value not found.\n"); return; }
    if (p->prev) p->prev->next = p->next; else head = p->next;
    if (p->next) p->next->prev = p->prev;
    printf("Deleted: %d\n", p->data); free(p);
}

void display_forward() {
    if (!head) { printf("List is empty.\n"); return; }
    printf("Forward: ");
    for (DNode* p = head; p; p = p->next) printf("%d <-> ", p->data);
    printf("NULL\n");
}

void display_reverse() {
    if (!head) { printf("List is empty.\n"); return; }
    DNode* p = head; while (p->next) p = p->next;
```

```
    printf("Reverse: ");
    for (; p; p = p->prev) printf("%d <-> ", p->data);
    printf("NULL\n");
}

int main() {
    int choice;
    do {
        printf("\n--- DOUBLY LINKED LIST MENU ---\n");
        printf("1. Insert at Front\n");
        printf("2. Insert at Back\n");
        printf("3. Insert at Position\n");
        printf("4. Delete at Position\n");
        printf("5. Delete by Value\n");
        printf("6. Display Forward\n");
        printf("7. Display Reverse\n");
        printf("8. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: insert_front(); break;
            case 2: insert_back(); break;
            case 3: insert_pos(); break;
            case 4: delete_pos(); break;
            case 5: delete_value(); break;
            case 6: display_forward(); break;
            case 7: display_reverse(); break;
            case 8: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice.\n");
        }
    } while (choice != 8);
    return 0;
}
```

```
/*====================================
  CIRCULAR SINGLY LINKED LIST (CSLL)
  File: cll_menu.c
====================================*/
#include <stdio.h>
#include <stdlib.h>

typedef struct CNode {
    int data;
    struct CNode* next;
} CNode;

CNode* head = NULL;

CNode* cnew(int x) {
    CNode* t = (CNode*)malloc(sizeof(CNode));
    if (!t) { printf("Memory allocation failed.\n"); exit(0); }
    t->data = x; t->next = t; return t;
}

void insert_front() {
    int x; printf("Enter value: "); scanf("%d", &x);
    CNode* t = cnew(x);
    if (!head) { head = t; }
    else {
        CNode* tail = head;
        while (tail->next != head) tail = tail->next;
        t->next = head; tail->next = t; head = t;
    }
    printf("Inserted at front.\n");
}
```

```
void insert_back() {
   int x; printf("Enter value: "); scanf("%d", &x);
   CNode* t = cnew(x);
   if (!head) { head = t; }
   else {
      CNode* tail = head;
      while (tail->next != head) tail = tail->next;
      tail->next = t; t->next = head;
   }
   printf("Inserted at back.\n");
}

void delete_value() {
   if (!head) { printf("List empty.\n"); return; }
   int x; printf("Enter value to delete: "); scanf("%d", &x);
   CNode *cur = head, *prev = NULL;
   do {
      if (cur->data == x) {
         if (prev) prev->next = cur->next;
         else {
            // deleting head
            CNode* tail = head;
            while (tail->next != head) tail = tail->next;
            if (tail == head) { free(cur); head = NULL; printf("Deleted.\n"); return; }
            tail->next = cur->next; head = cur->next;
         }
         free(cur); printf("Deleted.\n"); return;
      }
      prev = cur; cur = cur->next;
   } while (cur != head);
   printf("Value not found.\n");
}
```

```
void display() {
    if (!head) { printf("List is empty.\n"); return; }
    CNode* p = head;
    printf("Circular List: ");
    do { printf("%d -> ", p->data); p = p->next; } while (p != head);
    printf("(back to head)\n");
}

int main() {
    int choice;
    do {
        printf("\n--- CIRCULAR SLL MENU ---\n");
        printf("1. Insert at Front\n");
        printf("2. Insert at Back\n");
        printf("3. Delete by Value\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: insert_front(); break;
            case 2: insert_back(); break;
            case 3: delete_value(); break;
            case 4: display(); break;
            case 5: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice.\n");
        }
    } while (choice != 5);
    return 0;
}
```