DSA LAB INTERNALS

Program 1. STRING OPERATIONS

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

int search(char p[],char t[])
{
    int n, m, i, j;

    n = strlen(t);
    m = strlen(p);

    for(i=0; i<=n-m; i++)
    {
        j=0;
        while(j<m && p[j] == t[i+j]){
            j++;
        }

        if (j==m) return i;
    }
    return -1;
}

void replace(char p[] ,char t[] ,char r[] , int pos)
{
    int i,k;
    char d[30];

    for(k=0;k<pos;k++) d[k] = t[k];

    for(i=0;i<strlen(r);i++) d[k++] = r[i];

    pos += strlen(p);

    for(i=pos;i<=strlen(t);i++) d[k++] = t[i];

    for(i=0;i<=strlen(d);i++) t[i] = d[i];
}

void main(){
    char t[30],p[30],r[30];
    int pos;

    printf("\nSTR: "); scanf("%[^\n]",t);
```

```
    printf("PAT: "); scanf("%s",p);
    printf("REP: "); scanf("%s",r);

    pos = search(p,t);

    if(pos == -1)
    {
        printf("Pattern String not found!!\n\n");
        return;
    }

    for(;;)
    {
        replace(p,t,r,pos);
        pos = search(p,t);

        if(pos == -1) break;
    }

    printf("FINAL : %s\n\n",t);
}
```

Program 2. STACK OPERATIONS

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

#define STACK_SIZE 10

int stack[10];

void push(int item, int stack[], int *top)
{
    if (*top == STACK_SIZE - 1)
    {
        printf("Stack overflow\n");
        return;
    }
    stack[++(*top)] = item;
}

void pop(int *top, int stack[])
{
    if (*top == -1)
    {
        printf("Stack underflow\n");
        return;
    }
    printf("Item deleted = %d\n", stack[(*top)--]);
```

```c
}

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


void palindrome(char str[],int top)
{
    int i;
    for(i=0;str[i]!='\0';i++)
        stack[++top] = str[i];

    for(i=0;str[i]!='\0';i++)
    {
        if(str[i] == stack[top--]) continue;
        printf("%s : Is not a Palindrome\n",str);

        return;
    }
    printf("%s : is  a Palindrome\n",str);
}

int main()
{
    int choice, item, top = -1;
    char str[10];

    while (1)
    {
        printf("1.Push\n2.Pop\n3.Display\n4.Is Palindrome or not ?\n5.Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter item to push: ");
                scanf("%d", &item);
```

```
                push(item, stack, &top);
                break;

            case 2:
                pop(&top, stack);
                break;

            case 3:
                display(top, stack);
                break;

            case 4:
                printf("Enter a string: ");
                scanf(" %[^\n]",str);

                palindrome(str,top);
                break;

            default:
                exit(0);
        }
    }
    return 0;
}
```

Program 3. INFIX TO POSTFIX

```
#include<stdio.h>

int F(char symbol)
{
    switch(symbol)
    {
        case '#': return -1;

        case '+':
        case '-': return 2;

        case '*':
        case '/': return 4;

        case '^':
        case '$': return 5;

        case '(': return 0;

        default: return 8;
    }
```

```
}

int G(char symbol)
{
    switch(symbol)
    {
        case ')': return 0;

        case '+':
        case '-': return 1;

        case '*':
        case '/': return 3;

        case '^':
        case '$': return 6;

        case '(': return 9;

        default : return 7;
    }
}

void infix_2_postix(char infix[], char postfix[])
{
    int i, j=0, top = -1;
    char s[20];
    s[++top] = '#';

    for(i=0; infix[i] != '\0'; i++)
    {
        while(F(s[top]) > G(infix[i]))
        {
            postfix[j++] = s[top--];
        }

        if(F(s[top]) != G(infix[i]))
            s[++top] = infix[i];
        else
            top--;
    }

    while(s[top] != '#')
    {
        postfix[j++] = s[top--];
    }

    postfix[j] = '\0';
```

```
}

void main()
{
    char infix[50], postfix[50];

    printf("Enter the infix expession : ");
    scanf("%s",infix);

    infix_2_postix(infix,postfix);

    printf("Postfix Expression is : %s\n",postfix);
}
```

```
Enter the infix expession : ((A+(B-C)*D)^E+F)
Postfix Expression is : ABC-D*+E^F+
```

Program 4A) STACK APPPLICATIONS

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

double compute(double op1, char op, double op2)
{
    switch(op)
    {
        case '+': return op1+op2;

        case '-': return op1-op2;

        case '*': return op1*op2;

        case '/': return op1/op2;

        case '^':
        case '$': return pow(op1,op2);
    }
}

double evaluate(char postfix[])
{
    int i, top =-1;

    double stack[20], op1, op2;

    for(i=0; postfix[i] != '\0'; i++)
    {
        if(postfix[i] >= '0' && postfix[i] < '9')
```

```
            stack[++top] = postfix[i] - '0';
        else
        {
            op2 = stack[top--];
            op1 = stack[top--];
            stack[++top] = compute(op1, postfix[i], op2);
        }
    }
    return stack[top--];
}

void main()
{
    char postfix[20];
    double result;

    printf("Enter the postfix expression : ");
    scanf("%s",postfix);

    result = evaluate(postfix);
    printf("Result : %lf\n",result);
}
```

```
Enter the postfix expression : 632-5*+2^3+
Result : 124.000000
```

B) TOWER OF HANOI

```
#include<stdio.h>

void transfer(int n, char source, char temp, char destination)
{
    if(n == 0) return;

    transfer(n-1, source, destination, temp);

    printf("Move disk %d from %c to %c\n",n,source,destination);

    transfer(n-1, temp, source, destination);
}

void main()
{
    int n;

    printf("Enter the number of disks : ");
```

```
    scanf("%d",&n);

    transfer(n,'A','B','C');
}
```

```
Enter the number of disks : 2
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
```

Program 5. CIRCULAR QUEUE

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

#define Q_SIZE 5

void display(int queue[], int front, int count)
{
    int i, temp;

    if(count == 0)
    {
        printf("Queue is empty\n");
        return;
    }

    printf("QUEUE : ");

    temp = front;

    for(i=0; i<count; i++)
    {
        printf("%d ",queue[temp]);
        temp = (temp+1) % Q_SIZE;
    }

    printf("\n");
}

void insert_rear(int item, int queue[], int *rear, int *count)
{
    if(*count == Q_SIZE)
    {
        printf("Queue is full\n");
        return;
    }
```

```c
    *rear = (*rear + 1) % Q_SIZE;

    queue[*rear] = item;

    (*count)++;
}

void delete_front(int queue[], int *front, int *count)
{
    if(*count == 0)
    {
        printf("Queue is empty\n");
        return;
    }

    printf("Item deleted : %d\n",queue[*front]);

    (*front) = (*front + 1) % Q_SIZE;
    (*count)--;
}

void main()
{
    int choice, item, queue[10], front = 0, rear = -1, count = 0;

    for(;;)
    {
        printf("1.Insert\n2.Delete\n3.Display\n4.Exit\nEnter yur choice : ");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:
                    printf("Enter the item : ");
                    scanf("%d",&item);

                    insert_rear(item, queue, &rear, &count);
                    break;

            case 2: delete_front(queue, &front, &count);
                    break;

            case 3: display(queue, front, count);
                    break;

            default: exit(0);
        }
```

```
    }
}
```

Program 6. SINGLY LINKED LIST

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

struct node
{
    int sem;
    char phone[50];
    char usn[50];
    char name[50];
    char prog[50];
    struct node* link;
};

typedef struct node* NODE;

NODE createNode()
{
    NODE newNode = (NODE)malloc(sizeof(struct node));

    if(newNode == NULL)
    {
        printf("Memory allocation failed\n\n");
        exit(0);
    }

    newNode->link = NULL;
    return newNode;
}

void read_student_details(char usn[], char name[], char prog[], int *sem, char
phone[])
{
    printf("Enter the student details\n");
    printf("USN : "); scanf("%s", usn);
    printf("Name : "); scanf("%s", name);
    printf("Program : "); scanf("%s", prog);
    printf("SEM : "); scanf("%d", sem);
    printf("Phone No. : "); scanf("%s", phone);
}

NODE insert_front(char usn[], char name[], char prog[], int sem, char phone[],
NODE first)
{
```

```
    NODE newNode = createNode();

    strcpy(newNode->usn, usn);
    strcpy(newNode->name, name);
    strcpy(newNode->prog, prog);
    strcpy(newNode->phone, phone);

    newNode->sem = sem;
    newNode->link = first;

    return newNode;
}

NODE delete_front(NODE first)
{
    NODE cur = first;

    if(first == NULL)
    {
        printf("\nStudent list is empty\n\n");
        return NULL;
    }

    printf("\nStudent : %s's details deleted\n\n", first->name);
    first = first->link;
    free(cur);

    return first;
}

NODE insert_rear(NODE first, char usn[], char name[], char prog[], int sem, char
phone[])
{
    NODE newNode = createNode();

    strcpy(newNode->usn, usn);
    strcpy(newNode->name, name);
    strcpy(newNode->prog, prog);
    strcpy(newNode->phone, phone);

    newNode->sem = sem;

    if(first == NULL)
    {
        return newNode;
    }

    NODE cur = first;
```

```
    while(cur->link != NULL)
    {
        cur = cur->link;
    }

    cur->link = newNode;

    return first;
}

NODE delete_rear(NODE first)
{
    NODE cur = first, prev = NULL;

    if(first == NULL)
    {
        printf("\nList is empty\n\n");
        return NULL;
    }

    while(cur->link != NULL)
    {
        prev = cur;
        cur = cur->link;
    }

    if(prev == NULL)
    {
        free(first);
        return NULL;
    }

    prev->link = NULL;
    printf("\nStudent : %s's details deleted\n\n", cur->name);
    free(cur);

    return first;
}

int count_node(NODE first)
{
    int count = 0;
    NODE temp = first;

    while(temp != NULL)
    {
        count++;
        temp = temp->link;
```

```
    }

    return count;
}

void display_list(NODE first)
{
    NODE temp = first;

    if(first == NULL)
    {
        printf("\nList is empty\n\n");
        return;
    }

    printf("STUDENT LIST\n");
    printf("%-10s %-15s %-10s %-5s %-15s\n", "USN", "NAME", "PROGRAM", "SEM",
"PHONE NO.");

    while(temp != NULL)
    {
        printf("%-10s %-15s %-10s %-5d %-15s\n", temp->usn, temp->name, temp-
>prog, temp->sem, temp->phone);
        temp = temp->link;
    }
    printf("\n");
}

void main()
{
    char usn[50], name[50], prog[50], phone[50];
    int sem, choice, count = 0;
    NODE first = NULL;

    for(;;)
    {
        printf("1.Insert at Front\n2.Insert at Rear\n3.Delete at Front\n4.Delete
at Rear\n5.Display\n6.Count no. of Nodes\n7.Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:
                read_student_details(usn, name, prog, &sem, phone);
                first = insert_front(usn, name, prog, sem, phone, first);
                break;
```

```c
            case 2:
                read_student_details(usn, name, prog, &sem, phone);
                first = insert_rear(first, usn, name, prog, sem, phone);
                break;

            case 3:
                first = delete_front(first);
                break;

            case 4:
                first = delete_rear(first);
                break;

            case 5:
                display_list(first);
                break;

            case 6:
                count = count_node(first);
                printf("The number of students in the list is: %d\n\n", count);
                break;

            case 7:
                exit(0);

            default:
                printf("Invalid choice! Please try again.\n\n");
        }
    }
}
```

```
1.Insert at Front
2.Insert at Rear
3.Delete at Front
4.Delete at Rear
5.Display
6.Count no. of Nodes
7.Exit
Enter your choice: 1
Enter the student details
USN : 1VA23CI052
Name : KARTHIK
Program : AIML
SEM : 3
Phone No. : 8618399224
1.Insert at Front
2.Insert at Rear
3.Delete at Front
4.Delete at Rear
5.Display
6.Count no. of Nodes
7.Exit
Enter your choice: 5
STUDENT LIST
USN        NAME           PROGRAM    SEM   PHONE NO.
1VA23CI052 KARTHIK        AIML       3     8618399224
```

Program 7. DOUBLY LINKED LIST

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

typedef struct
{
    char ssn[20];
    char name[50];
    char department[20];
    char designation[20];
    char phone[20];
    float salary;
}EMPLOYEE;

struct node
{
    char ssn[20];
    char name[50];
    char department[20];
    char designation[20];
    char phone[20];
    float salary;
    struct node* llink;
    struct node* rlink;
};

typedef struct node* NODE;

// Function to create a new node
NODE getNode()
{
    NODE temp = (NODE)malloc(sizeof(struct node));

    if(!temp)
    {
        printf("Memory allocation failed\n");
        return NULL;
    }

    temp->llink = temp;
    temp->rlink = temp;

    return temp;
}

void dl_display(NODE head)
{
```

```
    if(head->rlink == head)
    {
        printf("List is empty\n");
        return;
    }

    printf("EMPLOYEE LIST : \n");
    printf("%-10s %-15s %-15s %-15s %-15s %-10s\n", "SSN", "NAME", "DEPARTMENT",
"DESIGNATION", "PHONE NO.", "SALARY");

    NODE cur = head->rlink;

    while(cur != head)
    {
        printf("%-10s %-15s %-15s %-15s %-15s %-10f\n",cur->ssn, cur->name, cur-
>department, cur->designation, cur->phone, cur->salary);
        cur = cur->rlink;
    }

    printf("NULL\n");
}

int count_node(NODE head)
{
    int count = 0;
    NODE temp = head->rlink;

    if(head->rlink == head) return 0;

    while(temp != head)
    {
        count++;
        temp = temp->rlink;
    }

    return count;
}


NODE dl_insert_rear(EMPLOYEE emp, NODE head)
{
    NODE temp = getNode();
    NODE last;

    strcpy(temp->ssn, emp.ssn);
    strcpy(temp->name, emp.name);
    strcpy(temp->department, emp.department);
    strcpy(temp->designation, emp.designation);
    strcpy(temp->phone, emp.phone);
```

```
    temp->salary = emp.salary;

    last = head->llink;
    temp->llink = last;
    last->rlink = temp;
    temp->rlink = head;
    head->llink = temp;

    return head;
}

NODE dl_insert_front(EMPLOYEE emp, NODE head)
{
    NODE temp = getNode();
    NODE first = NULL;

    strcpy(temp->ssn, emp.ssn);
    strcpy(temp->name, emp.name);
    strcpy(temp->department, emp.department);
    strcpy(temp->designation, emp.designation);
    strcpy(temp->phone, emp.phone);

    temp->salary = emp.salary;

    first = head->rlink;
    first->llink = temp;
    temp->rlink = first;
    head->rlink = temp;
    temp->llink = head;

    return head;
}

// Function to delete from the rear of the doubly linked list
NODE dl_delete_rear(NODE head)
{
    NODE last, prev;

    if(head->rlink == head)
    {
        printf("List is empty\n");
        return head;
    }

    last = head->llink;
    prev = last->llink;
    prev->rlink = head;
```

```
    head->llink = prev;

    printf("Details of Employee having SSN(%s)\n", last->ssn);
    free(last);

    return head;
}

NODE dl_delete_front(NODE head)
{
    NODE first, second;

    if(head->rlink == head)
    {
        printf("List is empty\n");
        return head;
    }

    first = head->rlink;
    second = first->rlink;
    head->rlink = second;
    second->llink = head;

    printf("Details of Employee having SSN(%s)\n", first->ssn);
    free(first);

    return head;
}

void read_employee_details(EMPLOYEE *emp)
{
    printf("Enter the student details\n");

    printf("SSN : ");
    scanf("%s", emp->ssn);

    printf("Name : ");
    scanf(" %[^\n]", emp->name);

    printf("Department : ");
    scanf(" %[^\n]", emp->department);

    printf("Designation : ");
    scanf(" %[^\n]", emp->designation);

    printf("Phone No. : ");
    scanf("%s", emp->phone);
```

```
    printf("Salary : ");
    scanf("%f",&emp->salary);
}

void main()
{
    int choice,count;
    NODE head = getNode();
    EMPLOYEE emp;

    for(;;)
    {
        printf("1. Insert at rear\n2. Insert at front\n3. Delete at rear\n4.
Delete at front\n5. Display\n6. Count\n7. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:
                read_employee_details(&emp);
                head = dl_insert_rear(emp, head);
                break;
            case 2:
                read_employee_details(&emp);
                head = dl_insert_front(emp, head);
                break;
            case 3:
                head = dl_delete_rear(head);
                break;
            case 4:
                head = dl_delete_front(head);
                break;
            case 5:
                dl_display(head);
                break;
            case 6:
                count = count_node(head);
                printf("Number of Employees : %d\n",count);
                break;
            case 7:
                exit(0);

            default:
                    printf("Invalid choice !!!\n");
                    break;
        }
    }
}
```

Program 8. SINGLY CIRCULAR LIST

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

struct node
{
    int c,px,py,pz;
    struct node* link;
};

typedef struct node* NODE;

//Function to get a NODE
NODE getNode()
{
    NODE temp = (NODE)malloc(sizeof(struct node));

    if(!temp)
    {
        printf("\nMemory allocation failed\n");
        return NULL;
    }

    temp->link = temp;
    return temp;
}

float evaluate(float x, float y, float z, NODE head)
{
    float sum = 0;

    NODE cur = head->link;
    while(cur != head)
    {
        sum = sum + cur->c * pow(x,cur->px) * pow(y,cur->py) * pow(z,cur->pz);
        cur = cur->link;
    }

    return sum;
}


NODE insert_rear(NODE head, int c, int px, int py, int pz)
{
    NODE cur,temp;

    temp = getNode();
```

```
    temp->c = c;
    temp->px = px;
    temp->py = py;
    temp->pz = pz;

    cur = head->link;

    while(cur->link!=head)
        cur = cur->link;

    cur->link = temp;
    temp->link = head;

    return head;
}

//Function to read the polynomial
NODE read_poly()
{
    NODE head = getNode();
    int c,px,py,pz;

    printf("Enter the coefficient and power of x,y,z : ");

    for(;;)
    {
        scanf("%d",&c);
        if(c==0) break;
        scanf("%d %d %d",&px,&py,&pz);

        head = insert_rear(head,c,px,py,pz);
    }

    return head;
}

//Function to add the 2 polynomial
NODE add_2_poly(NODE h1, NODE h2)
{
    int sum;
    NODE h3= getNode();
    NODE p,q;

    for(p=h1->link; p!=h1; p=p->link)
    {
        for(q=h2->link; q!=h2; q=q->link)
        {
            if((p->px == q->px) && (p->py == q->py) && (p->pz == q->pz))
```

```
            {
                sum = p->c + q->c;
                if(sum!=0)
                {
                    h3 = insert_rear(h3,sum,p->px,p->py,p->pz);
                }
                q->c = 0;
                break;
            }
        }
        if(q==h2)
        {
            h3 = insert_rear(h3,p->c,p->px,p->py,p->pz);
        }
    }

    for (q = h2->link; q != h2; q = q->link)
    {
        if (q->c != 0)
        {
            h3 = insert_rear(h3, q->c, q->px,q->py,q->pz);
        }
    }

    return h3;
}

//Function to display polynomial
void display(NODE head)
{
    NODE temp = head->link;

    while(temp!=head)
    {
        if(temp->c > 0)
        {
            printf("+%dx^%d y^%d z^%d ",temp->c,temp->px,temp->py,temp->pz);
        }
        else
        {
            printf("%dx^%d y^%d z^%d ",temp->c,temp->px,temp->py,temp->pz);
        }

        temp = temp->link;
    }

    printf("\n");
}
```

```c
void main()
{
    NODE head1,head2,head3;
    int choice;
    float x,y,z,sum;

    for(;;)
    {
        printf("1.Add\n2.Evaluate\n3.Exit\nEnter your choice : ");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:
                    printf("Enter the terms of 1st Polynomial : \n");
                    head1 = read_poly();

                    printf("Enter the terms of 2nd Polynomial : \n");
                    head2 = read_poly();

                    printf("Polynomial 1 : ");
                    display(head1);

                    printf("\nPolynomial 2 : ");
                    display(head2);

                    head3 = add_2_poly(head1,head2);

                    printf("\nAddition of 2 polynomials is : ");
                    display(head3);

                    break;

            case 2:
                    printf("Enter a polynomial : ");
                    head1 = read_poly();

                    printf("Enter values of x,y and z : ");
                    scanf("%f %f %f",&x,&y,&z);
                    sum = evaluate(x,y,z,head1);

                    printf("Result : %f\n",sum);

                    break;

            case 3: exit(0);
```

```
            default:
                    printf("Invalid choice !!\n");
        }
    }
}
```

```
1.Add
2.Evaluate
3.Exit
Enter your choice :
1
Enter the terms of 1st Polynomial :
Enter the coefficient and power of x,y,z : 3 2 1 0
2 1 2 1
4 0 0 2
0
Enter the terms of 2nd Polynomial :
Enter the coefficient and power of x,y,z : 1 2 1 0
-2 1 2 1
3 0 0 2
0
Polynomial 1 : +3x^2 y^1 z^0 +2x^1 y^2 z^1 +4x^0 y^0 z^2

Polynomial 2 : +1x^2 y^1 z^0 -2x^1 y^2 z^1 +3x^0 y^0 z^2

Addition of 2 polynomials is : +4x^2 y^1 z^0 +7x^0 y^0 z^2
1.Add
2.Evaluate
3.Exit
Enter your choice : 2
Enter a polynomial : Enter the coefficient and power of x,y,z : 3 2 1 0
2 1 2 1
4 0 0 2
0
Enter values of x,y and z : 1 2 3
Result : 66.000000
1.Add
2.Evaluate
3.Exit
Enter your choice :
```

Program 9. CALENDER

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

#define NO_OF_DAYS 7

typedef struct
{
    char *day;
    int date;
    char *activity;
}CALENDER;

void create_calender(CALENDER a[], int i, char day[], int date, char activity[])
{
    a[i].day = (char*)malloc(strlen(day) + 1);
    a[i].activity = (char*)malloc(strlen(activity) + 1);

    strcpy(a[i].day, day);
    a[i].date = date;
    strcpy(a[i].activity, activity);
}

void read_calender(CALENDER a[])
{
    int i, date;

    char day[10], activity[10];

    for(i=0; i<NO_OF_DAYS; i++)
    {
        scanf("%s",day);
        scanf("%d",&date);
        scanf("%s",activity);

        create_calender(a, i, day, date, activity);
    }
}

void print_weeks_activity(CALENDER a[])
{
    printf("Weeks Activity\n");

    for(int i=0; i<NO_OF_DAYS; i++)
        printf("%-10s : %s\n",a[i].day, a[i].activity);
```

```
}

void main()
{
    CALENDER a[NO_OF_DAYS];

    printf("DAY\tDATE\tACTIVITY\n");

    read_calender(a);

    print_weeks_activity(a);
}
```

```
DAY     DATE    ACTIVITY
MONDAY  15      SWIMMING
TUESDAY 16      GYM
WEDNESDAY 17    EXERCISE
THURSDAY 18     SLEEPING
FRIDAY   19     SLEEPING
SATURDAY 20     SLEEPING
SUNDAY   21     MOVIE
Weeks Activity
MONDAY      : SWIMMING
TUESDAY     : GYM
WEDNESDAY   : EXERCISE
THURSDAY    : SLEEPING
FRIDAY      : SLEEPING
SATURDAY    : SLEEPING
SUNDAY      : MOVIE
```