1. Stack

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

#define SIZE 100

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

void push(char item)
{
    if (top >= SIZE - 1)
    {
        printf("Stack Overflow.\n");
    }
    else
    {
        top = top + 1;
        stack[top] = item;
    }
}
char pop()
{
    char item;

    if (top < 0)
    {
        printf("stack under flow\n");
        exit(1);
    }
    else
    {
        item = stack[top];
        top = top - 1;
        return (item);
    }
```

```
}

int is_operator(char symbol)
{
    if (symbol == '^' || symbol == '*' || symbol == '/' ||
symbol == '+' || symbol == '-')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int precedence(char symbol)
{
    if (symbol == '^')
    {
        return 3;
    }
    else if (symbol == '*' || symbol == '/')
    {
        return 2;
    }
    else if (symbol == '+' || symbol == '-')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
    int i, j;
```

```
    char item;
    char x;
    push('(');
    strcat(infix_exp, ")");
    i = 0;
    j = 0;
    item = infix_exp[i];

    while (item != '\0')
    {
        if (item == '(')
        {
            push(item);
        }
        else if (isdigit(item) || isalpha(item))
        {
            postfix_exp[j] = item;
            j++;
        }
        else if (is_operator(item) == 1)
        {
            x = pop();
            while (is_operator(x) == 1 && precedence(x) >=
precedence(item))
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
            push(x);
            push(item);
        }
        else if (item == ')')
        {
            x = pop();
            while (x != '(')
            {
                postfix_exp[j] = x;
```

```
                j++;
                x = pop();
            }
        }
        else
        {
            printf("Invalid infix Expression.\n");
            exit(1);
        }
        i++;
        item = infix_exp[i];
    }
    if (top > 0)
    {
        printf("Invalid infix Expression.\n");
        exit(1);
    }
    postfix_exp[j] = '\0';
}

void EvalPostfix(char postfix[])
{
    int i;
    char ch;
    int val, num;
    int A, B;
    printf("Enter values \n");

    /* evaluate postfix expression */
    for (i = 0; postfix[i] != '\0'; i++)
    {
        ch = postfix[i];

        if (ch == '+' || ch == '-' || ch == '*' || ch == '/')
        {
            A = pop();
            printf("Stack val %d\n", A);
            B = pop();
```

```
            printf("Stack val %d\n", B);

            switch (ch) /* ch is an operator */
            {
            case '*':
                val = B * A;
                break;

            case '/':
                val = B / A;
                break;

            case '+':
                val = B + A;
                break;

            case '-':
                val = B - A;
                break;
            }

            /* push the value obtained above onto the stack */
            push(val);
        }
        else
        {

            printf("Enter Value of %c = ", ch);
            scanf("%d", &num);
            push(num);
        }
    }
    printf(" \n Result of expression evaluation : %d \n",
pop());
}

int main()
{
```

```
    char infix[SIZE], postfix[SIZE];
    printf("Enter Infix expression : ");
    scanf("%s", infix);
    InfixToPostfix(infix, postfix);
    printf("Postfix Expression: %s\n", postfix);
    // printf("%d", top);
    EvalPostfix(postfix);
    return 0;
}
```

Enter Infix expression : (a+b)/c
Postfix Expression: ab+c/
Enter values
Enter Value of a = 3
Enter Value of b = 4
Stack val 4
Stack val 3
Enter Value of c = 5
Stack val 5
Stack val 7

Result of expression evaluation : 1

2. Queuessss

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

// 500 copies of structures
#define MAXSIZE 500

// queue ka initialization
int front = -1, rear = -1;

// structure ka creation to hold int num and int memory
struct request
{
    int num;
```

```
    int mem;
};

struct request req[500]; // creating 500 copies of struct req
type variable

// insert function for queue
void insert(int number, int memory)
{

    // checking for full/empty/insertion condition to insert
    if (rear == MAXSIZE - 1)
    {
        printf("Queue is full");
        exit(0);
    }
    else if (front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;

        req[rear].num = number;
        req[rear].mem = memory;
    }
    else
    {
        rear++;
        req[rear].num = number;
        req[rear].mem = memory;
    }
}

// delete function for queue
struct request rem()
{
    // check if empty or not
    if (front == -1 || front > rear)
    {
```

```
        printf("NO ELEMENTS");
    }
    else
    {
        return req[front++];
    }
}

// main function
int main()
{
    int a_m = 3000;
    int num_in, mem_in;
    int no_req;
    struct request temp;
    printf("Enter number of requests\n");
    scanf("%d", &no_req);

    // take in all requeuests and allocating num_in and mem_in
    for (int i = 0; i < no_req; i++)
    {
        num_in = 1000 + rand() % 9000; // to get num_in >1000
and <100000
        mem_in = 100 + rand() % 900;

        // passing into function
        insert(num_in, mem_in);
        printf("The process %d has been allocated %d bytes\n",
num_in, mem_in);
    }

    // request processing using remeove function and checking
memeory storage avaiavalbitly while front<=rear

    while (front <= rear)
    {
        temp = rem();
        if (a_m >= temp.mem)
```

```
        {
            a_m = a_m - temp.mem;
            printf("Request number %d successfullly processed,
Available cap is %d\n", temp.num, a_m);
        }
        else
        {
            printf("Request number %d cannot be processed due
to shortage of memory,avaiable free space is %d\n", temp.num,
a_m);
            break;
        }
    }
    return 0;
}
```

3. Polynomial addition

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

struct node
{
    int coef;
    int power;
    struct node *link;
};

struct node *add_node(int c, int p, struct node *poly)
{
    struct node *new;
    new = (struct node *)malloc(sizeof(struct node));
    new->coef = c;
    new->power = p;
    new->link = NULL;
    if (poly == NULL)
    {
```

```
        poly = new;
    }
    else
    {
        struct node *temp;
        temp = poly;
        while (temp->link != NULL)
        {
            temp = temp->link;
        }
        temp->link = new;
    }
    return poly;
}

void display(struct node *poly)
{
    while (poly != NULL)
    {
        printf("%dx^%d", poly->coef, poly->power);
        if (poly->link != NULL)
            printf(" + ");
        poly = poly->link;
    }
}
struct node *add(struct node *poly1, struct node *poly2,
struct node *result)
{
    struct node *t1;
    struct node *t2;
    t1 = poly1;
    t2 = poly2;
    int c, p;
    while (t1 != NULL && t2 != NULL)
    {
        if (t1->power == t2->power)
        {
            c = t1->coef + t2->coef;
```

```
            p = t1->power;
            t1 = t1->link;
            t2 = t2->link;
        }
        else if (t1->power > t2->power)
        {
            c = t1->coef;
            p = t1->power;
            t1 = t1->link;
        }
        else
        {
            c = t2->coef;
            p = t2->power;
            t2 = t2->link;
        }
        result = add_node(c, p, result);
    }
    // this is needed if there are unequal number of terms
    while (t1 != NULL)
    {
        c = t1->coef;
        p = t1->power;
        t1 = t1->link;
        result = add_node(c, p, result);
    }
    while (t2 != NULL)
    {
        c = t2->coef;
        p = t2->power;
        t2 = t2->link;
        result = add_node(c, p, result);
    }

    return result;
}

int main()
```

```c
{
    struct node *p1 = NULL, *p2 = NULL, *result_add = NULL;
    printf("Enter coefficient and power for polynomial 1\n");
    int c, p;
    while (1)
    {
        scanf("%d %d", &c, &p);
        if (p == -1)
            break;
        p1 = add_node(c, p, p1);
    }
    printf("Enter coefficient and power for polynomial 2\n");
    while (1)
    {
        scanf("%d %d", &c, &p);
        if (p == -1)
            break;
        p2 = add_node(c, p, p2);
    }
    printf("Polynomial1: ");
    display(p1);
    printf("\nPolynomial2: ");
    display(p2);
    result_add = add(p1, p2, result_add);
    printf("initial result is \n");
    display(result_add);

    printf("\nFinal Polynomial after addition: ");
    display(result_add);
}

// 1 1 2 2 3 3 -1 -1
// 3 3 4 4 1 1 -1 -1
```

4. Text editor

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

struct node
{
    char line[80];
    struct node *rlink;
    struct node *llink;
};

char buffer[80];
char newline[80];
int n, d;

struct node *first, *temp, *rlink, *llink;

void insert(char l[])
{

    struct node *newnode = (struct node *)malloc(sizeof(struct
node));
    newnode->rlink = NULL;
    newnode->llink = NULL;
    strcpy(newnode->line, l);

    if (first == NULL)
    {
        first = newnode;
        temp = first;
    }

    else
    {

        newnode->llink = temp;
```

```
        //   if (temp->rlink != NULL)

        temp->rlink = newnode;
        temp = newnode;
    }
}

void insertnew(char nl[], int p)
{

    struct node *newnode = (struct node *)malloc(sizeof(struct
node));
    newnode->rlink = NULL;
    newnode->llink = NULL;
    temp = first;
    strcpy(newnode->line, nl);

    if (p == 1)
    {
        first->llink = newnode;
        newnode->rlink = first;
        first = newnode;
        temp = first;
    }

    else
    {
        for (int i = 0; i < p - 2; i++)
        {
            temp = temp->rlink;
        }
        {
            newnode->llink = temp;
            newnode->rlink = temp->rlink;
            newnode->rlink->llink = newnode;
            temp->rlink = newnode;
            //   p = p + 1;
        }
```

```
    }
}

void display()
{

    struct node *temp1 = first;
    printf("\nDisplaying:\n");
    while (temp1->rlink != NULL)
    {

        printf("%s\n", temp1->line);
        temp1 = temp1->rlink;
    }
}

void Delete(int p)
{
    int i;
    // struct node *temp;
    temp = first;

    if (p == 1)
    {
        first = temp->rlink;
        temp->rlink = NULL;
    }
    else
    {
        for (int i = 0; i <= p - 2; i++)
        {
            temp = temp->rlink;
        }
        {
            temp->llink->rlink = temp->rlink;
            temp->rlink->llink = temp->llink;
        }
    }
```

```
}

void main()
{

    first = NULL;
    temp = NULL;
    rlink = NULL;
    llink = NULL;

    printf("Enter few lines:");
    scanf(" %[^\n]", newline);
    insert(newline);
    char end[4] = "end";
    int flag = 1;

    while (flag != 0)
    {
        scanf(" %[^\n]", newline);
        flag = strcmp(newline, end);
        insert(newline);
    }
    display();

    printf("\nEnter the Line no. to insert:");
    scanf("%d", &n);
    printf("\nEnter the Line to be inserted:");
    scanf(" %[^\n]", buffer);
    insertnew(buffer, n);
    display();

    printf("\nEnter the line to be deleted:");
    scanf("%d", &d);
    Delete(d);
    display();
}
```

5. Binary tree

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
struct node
{
    char info;
    struct node *left, *right;
};

struct stack
{
    struct node *top;
    struct stack *link;
};

struct node *newnode(char ch)
{
    struct node *temp = (struct node *)malloc(sizeof(struct
node));
    temp->info = ch;
    temp->left = temp->right = NULL;
    return temp;
}

void push(struct stack **s, struct node *x)
{
    struct stack *temp = (struct stack *)malloc(sizeof(struct
stack));
    temp->top = x;
    temp->link = *s;
    *s = temp;
}
```

```
struct node *pop(struct stack **s)
{
    if (*s == NULL)
    {
        printf("Stack underflow\n");
        return NULL;
    }
    struct node *temp = (*s)->top;
    *s = (*s)->link;
    return temp;
}

struct node *ExpTree(char postfix[])
{
    struct stack *s = NULL;
    struct node *t, *t1, *t2;
    for (int i = 0; i < strlen(postfix); i++)
    {
        t = newnode(postfix[i]);
        if (isalnum(postfix[i]))
            push(&s, t);
        else
        {
            t1 = pop(&s);
            t2 = pop(&s);
            t->right = t1;
            t->left = t2;
            push(&s, t);
        }
    }
    return pop(&s);
}

void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
```

```
        printf("%c", root->info);
        inorder(root->right);
    }
}

void preorder(struct node *root)
{
    if (root != NULL)
    {
        printf("%c", root->info);
        preorder(root->left);
        preorder(root->right);
    }
}

int main()
{
    char postfix[20];
    printf("Enter postfix expression: ");
    scanf("%s", postfix);
    struct node *root = ExpTree(postfix);
    printf("Infix Expression is: ");
    inorder(root);
    printf("\nPrefix expression is: ");
    preorder(root);
    return 0;
}
```

6. Dictionary

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

struct node
{
    char word[500];
    struct node *left, *right;
};

struct node *add(struct node *head, char s[])
{

    struct node *new = (struct node *)malloc(sizeof(struct
node));
    strcpy(new->word, s);
    new->left = NULL;
    new->right = NULL;
    if (head == NULL)
    {
        head = new;
    }

    if (strcmp(s, head->word) > 0)
    {
        head->right = add(head->right, s);
    }
    else if (strcmp(s, head->word) < 0)
    {
        head->left = add(head->left, s);
    }
    return head;
}

void inorder(struct node *head)
```

```
{
    if (head != NULL)
    {
        inorder(head->left);
        printf("%s\n", head->word);
        inorder(head->right);
    }
}

struct node *delete(struct node *head, char s[])
{
    if (head == NULL)
    {
        return NULL;
    }
    if (strcmp(s, head->word) > 0)
    {
        head->right = delete (head->right, s);
    }
    else if (strcmp(s, head->word) < 0)
    {
        head->left = delete (head->left, s);
    }
    else
    {
        if (head->right == NULL)
        {
            struct node *tmp = head->left;
            free(head);
            return tmp;
        }
        else if (head->left == NULL)
        {
            struct node *tmp = head->right;
            free(head);
            return tmp;
        }
        else
```

```
        {
            struct node *tmp = head->right;
            while (tmp->left != NULL)
            {
                tmp = tmp->left;
            }
            strcpy(head->word, tmp->word);
            head->right = delete (head->right, head->word);
        }
    }
    return head;
}

int main()
{
    int choice;
    char a[100];
    struct node *head = NULL;
    while (1)
    {
        printf("Select an option:\n");
        printf("1. Enter words\n");
        printf("2. Delete words\n");
        printf("3. Display words\n");
        printf("4. Exit\n");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            while (1)
            {
                printf("Enter a word (-1 to exit): ");
                scanf("%s", a);
                if (strcmp(a, "-1") == 0)
                {
                    break;
                }
```

```
                head = add(head, a);
            }
            break;

        case 2:
            printf("Enter a word to delete: ");
            scanf("%s", a);
            head = delete (head, a);
            break;
        case 3:
            printf("Words in alphabetical order:\n");
            inorder(head);
            break;
        case 4:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

7. Dickstara

```
#include <stdio.h>
// #include <math.h>
int main()
{
    int n, g[100][100], i, j, s, visited[100], count, next;
    int distance[100];
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjaceny matrix: \n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
```

```
        {
            scanf("%d", &g[i][j]);
            if (g[i][j] == 0)
            {
                g[i][j] = 999;
            }
        }
    }

    printf("Enter the starting node: ");
    scanf("%d", &s);

    for (i = 0; i < n; i++)
    {
        distance[i] = g[s][i];

        visited[i] = 0;
    }
    distance[s] = 0;
    visited[s] = 1;
    count = 1;
    while (count < n - 1)
    {
        int mindistance = 999;
        for (i = 0; i < n; i++)
        {
            if (distance[i] < mindistance && !visited[i])
            {
                mindistance = distance[i];
                next = i;
            }
        }
        visited[next] = 1;
        for (i = 0; i < n; i++)
        {
            if (!visited[i])
            {
                if ((mindistance + g[next][i]) < distance[i])
```

```
                {
                    distance[i] = mindistance + g[next][i];
                }
            }
        }
        count++;
    }
    for (i = 0; i < n; i++)
    {
        if (i != s)
        {
            printf("\nDistance of %d from %d = %d\n", i, s,
distance[i]);
        }
    }
    return 0;
}
```

8. Rabin karp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char line[100], pat[100];
int n, m;
float hasher(char a[])
{
    int i;
    float val = 1000;

    for (i = 0; i < n; i++)
    {
        val = val / a[i];
        printf("%f\t", val);
    }
    return val;
```

```
}
int main()
{
    char new[100];
    printf("enter the line:");
    scanf("%s", line);
    printf("enter the pattern:");
    scanf("%s", pat);
    m = strlen(line);
    n = strlen(pat);
    int i, j;
    for (i = 0; i < m; i++)
    {
        if (line[i] == pat[0])
        {
            for (j = 0; j < n; j++)
            {
                new[j] = line[i + j];
            }
            if (hasher(pat) == hasher(new))
            {
                printf("position is at %d\n", i);
            }
        }
    }
    return 0;
}
```

9. Heapify

```
#include <stdio.h>
void Insert(int H[], int n)
{
    int i = n, temp;
    temp = H[i];
```

```
    while (i > 1 && temp > H[i / 2])
    {
        H[i] = H[i / 2];
        i = i / 2;
    }
    H[i] = temp;
}
int main()
{
    // int H[10] = {0, 14, 15, 5, 20, 30, 8, 40};
    int n, i;
    int H[10];
    H[0] = 0;
    printf("Enter number of elements ");
    scanf("%d", &n);

    printf("enter numbers now");

    for (i = 1; i <= n; i++)
    {
        scanf("%d", &H[i]);
    }
    for (i = 2; i <= n; i++)
        Insert(H, i);

    for (i = 1; i <= n; i++)
        printf("%d ", H[i]);
    printf("\n");

    return 0;
}
```