Program 1:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define NUM_DAYS_IN_WEEK 7
typedef struct
{
char *acDayName;
int iDate;
char *acActivity;
} DAYTYPE;
void fnFreeCal (DAYTYPE *);
void fnDispCal (DAYTYPE *);
void fnReadCal (DAYTYPE *);
DAYTYPE *fnCreateCal();
int main()
{
DAYTYPE *weeklyCalendar = fnCreateCal();
fnReadCal (weeklyCalendar);
fnDispCal (weeklyCalendar);
fnFreeCal (weeklyCalendar);
return 0;
}
DAYTYPE *fnCreateCal ()
{
DAYTYPE *calendar = (DAYTYPE *)malloc( NUM_DAYS_IN_WEEK *sizeof(DAYTYPE));
for (int i = 0; i < NUM_DAYS_IN_WEEK; i++)
{
```

```
calendar[i].acDayName = NULL;
calendar[i].iDate = 0;
calendar[i].acActivity = NULL;
}
return calendar;
}
void fnReadCal (DAYTYPE *calendar)
{
char cChoice;
for (int i = 0; i < NUM_DAYS_IN_WEEK; i++)
{
printf("Do you want to enter details for day %d [Y/N]: ", i + 1);
scanf("%c", &cChoice);
getchar();
if (tolower(cChoice) == 'n')
continue;
printf("Day Name: ");
char nameBuffer[50];
scanf("%s", &nameBuffer);
calendar[i].acDayName = strdup (nameBuffer);
printf("Date: ");
scanf("%d", &calendar[i].iDate);
printf("Activity: ");
char activityBuffer[100];
scanf("%S", &activityBuffer);
calendar[i].acActivity = strdup (activityBuffer);
printf("\n");
getchar();
}
}
```

```
void fnDispCal (DAYTYPE *calendar)
{
printf("\nWeek's Activity Details:\n");
for (int i = 0; i < NUM_DAYS_IN_WEEK; i++)
{
printf("Day %d:\n", i + 1);
if (calendar[i].iDate == 0)
{
printf("No Activity\n\n");
continue;
}
printf(" Day Name: %s\n", calendar[i].acDayName);
printf(" Date: %d\n", calendar [i].iDate);
printf(" Activity: %s\n\n", calendar[i].acActivity);
}
}
void fnFreeCal (DAYTYPE *calendar)
{
for(int i = 0; i < NUM_DAYS_IN_WEEK; i++)
{
free (calendar[i].acDayName);
free (calendar[i].acActivity);
}
free(calendar);
}
```

Program 2 :

```
#include <stdio.h>

#include <string.h>


int main()

{

   char st[200], srch[30], rep[30], res[200], cpy[200];

   int i=0, j=0 ,k=0, l, mtch, iStop, len, nom=0;


   printf("\nEnter the main string\n");

            scanf(" %[^\n]", st);

   printf("\nEnter the Pattern string\n");

            scanf(" %[^\n]", srch);

   printf("\nEnter the Replace string\n");

            scanf(" %[^\n]", rep);

   strcpy(cpy, st);

   for(i=0;i<(strlen(st)-strlen(srch)+1);i++)

   {

      mtch = 0;

      for(j=0;j<strlen(srch);j++)

      {

       if(st[i+j] == srch[j])

         {

            mtch++;

         } else {

            break;

         }

         if(mtch == strlen(srch))

         {

            nom++;
            for(k=0;k<i;k++)
            {
               res[k] = st[k];
            }
            iStop = k + strlen(srch);
            res[k] = '\0';
            strcat(res, rep);
            len = strlen(res);
            for(k=iStop, l=0; st[k] != '\0';k++, l++)
            {
               res[len+l] = st[k];
            }
            res[len+l] = '\0';
            strcpy(st,res);
         }
      }
   }
   printf("\nInput Text\n");
   printf("%s\n",cpy);
   if(nom > 0)
   {
      printf("\n%d matches occured\n\nText after replacing matched patterns is shown below\n",
nom);
      printf("\n%s\n",res);
   } else
{
      printf("\nPattern String not found in Text\n");
   }
   return 0;
}
```

Program 3:

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

int stack[MAX], top = -1;

int isFull() {
  return top == MAX - 1;
}

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

void push() {
  if (isFull()) {
    printf("Stack Overflow!\n");
    return;
  }
  int element;
  printf("Enter element to push: ");
  scanf("%d", &element);
  stack[++top] = element;
  printf("%d pushed onto the stack.\n", element);
}

void pop() {
  if (isEmpty()) {
    printf("Stack Underflow!\n");
```

```
        return;
    }
    printf("%d popped from the stack.\n", stack[top--]);
}

void checkPalindrome() {
    if (isEmpty()) {
        printf("Stack is empty, cannot check palindrome.\n");
        return;
    }
    int i, isPalin = 1;
    for (i = 0; i <= top / 2; i++) {
        if (stack[i] != stack[top - i]) {
            isPalin = 0;
            break;
        }
    }
    if (isPalin)
        printf("The stack contents form a palindrome.\n");
    else
        printf("The stack contents do not form a palindrome.\n");
}

void display() {
    if (isEmpty()) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack contents: ");
    for (int i = 0; i <= top; i++)
```

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

int main() {
    int choice;
    do {
        printf("\n--- Stack Menu ---\n");
        printf("1. Push\n2. Pop\n3. Check Palindrome\n4. Display\n5. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1: push(); break;
        case 2: pop(); break;
        case 3: checkPalindrome(); break;
        case 4: display(); break;
        case 5: printf("Exiting...\n"); break;
        default: printf("Invalid choice!\n");
        }
    } while (choice != 5);

    return 0;
}
```

Program 4 :

```
#include <stdio.h>

#include <ctype.h>

#include <string.h>


#define MAX 100

char stack[MAX];

int top = -1;


void push(char c) { stack[++top] = c; }

char pop() { return stack[top--]; }

char peek() { return (top == -1) ? -1 : stack[top]; }

int precedence(char c) { return (c == '^') ? 3 : (c == '*' || c == '/' || c == '%') ? 2 : (c == '+' || c == '-') ? 1 :
0; }


void infixToPostfix(char* in, char* post)

{

  int i = 0, j = 0;

  char c;

  while ((c = in[i++]) != '\0') {

    if (isalnum(c))

    {

      post[j++] = c;

    }

    else if (c == '(')

    {

      push(c);

    }

    else if (c == ')')

    {
            while (peek() != '(') {
                post[j++] = pop();
            }
            pop(); // Remove '('
        } else
        {
            while (top != -1 && precedence(peek()) >= precedence(c))
             {
                post[j++] = pop();
             }
            push(c);
        }
    }
    while (top != -1) {
        post[j++] = pop();
    }
    post[j] = '\0';
}

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

Program 5 a:

```
#include <stdio.h>
void push(int [], int*, int);
int pop(int [], int*);
int main()
{
int iastack[50], i, op1, op2, res;
char expr[50], symb;
int top = -1;
printf("\nEnter a valid postfix expression : \n");
scanf("%s", expr);
for(i=0; i<strlen(expr); i++)
{ symb = expr[i];
if(isdigit(symb))
{
push(iastack, &top, symb-'0');
}
else
{
op2 = pop(iastack, &top);
op1 = pop(iastack, &top);
switch(symb)
{ case '+' : res = op1 + op2;
break;
case '-' : res = op1 - op2;
break;
case '*' : res = op1 * op2;
break;
case '/' : res = op1 / op2;
break;
case '%' : res = op1 % op2;
break;
case '^' : res = (int)pow(op1 , op2);
break;
}
push(iastack, &top, res);
}
}
res = pop(iastack, &top);
printf("\nValue of %s expression is : %d\n", expr, res);
return 0;
}
void push(int Stack[], int *t , int elem)
{
*t = *t + 1;
Stack[*t] = elem;
}
.
int pop(int Stack[], int *t)
{
int elem;
elem = Stack[*t];
*t = *t -1;
return elem;
}
```

Program 5 b :

```
#include <stdio.h>
void towers(int, char, char, char);
int main()
{
int num;
printf("Enter the number of disks : ");
scanf("%d", &num);
printf("The sequence of moves involved in the Tower of Hanoi are :\n");
towers(num, 'A', 'C', 'B');
printf("\n");
return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
if (num == 1)
{
printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
return;
}
towers(num - 1, frompeg, auxpeg, topeg);
printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
towers(num - 1, auxpeg, topeg, frompeg);
}
```

Program 6:

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

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

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

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

void insert() {
  char element;
  if (isFull()) {
    printf("Queue Overflow! Cannot insert more elements.\n");
    return;
  }
  printf("Enter the element to insert: ");
  scanf(" %c", &element);
  if (isEmpty()) {
    front = rear = 0;
  } else {
    rear = (rear + 1) % MAX;
  }
  queue[rear] = element;
  printf("Inserted '%c' into the queue.\n", element);
```

```
}

void delete() {
    if (isEmpty()) {
        printf("Queue Underflow! No elements to delete.\n");
        return;
    }
    printf("Deleted '%c' from the queue.\n", queue[front]);
    if (front == rear) {
        front = rear = -1;
    } else {
        front = (front + 1) % MAX;
    }
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue contents: ");
    int i = front;
    do {
        printf("%c ", queue[i]);
        i = (i + 1) % MAX;
    } while (i != (rear + 1) % MAX);
    printf("\n");
}

int main() {
    int choice;
    do {
        printf("\n--- Circular Queue Menu ---\n");
        printf("1. Insert an Element\n");
        printf("2. Delete an Element\n");
        printf("3. Display Queue\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 program...\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Program 7:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Node {

   char usn[20], name[30], prog[30];

   int sem;

   long phNo;

   struct Node *link;

};


typedef struct Node *NODE;

NODE head = NULL;

int count = 0;


NODE createNode() {

   NODE temp = (NODE)malloc(sizeof(struct Node));

   printf("Enter USN, Name, Programme, Sem, Phone: ");

   scanf("%s %s %s %d %ld", temp->usn, temp->name, temp->prog, &temp->sem, &temp->phNo);

   temp->link = NULL;

   count++;

   return temp;

}


void insertFront() {

   NODE temp = createNode();

   temp->link = head;

   head = temp;

}
```

```
void insertEnd() {
    NODE temp = createNode();
    if (!head) {
        head = temp;
        return;
    }
    NODE cur = head;
    while (cur->link) cur = cur->link;
    cur->link = temp;
}

void deleteFront() {
    if (!head) {
        printf("List is empty.\n");
        return;
    }
    NODE temp = head;
    head = head->link;
    printf("Deleted: %s\n", temp->usn);
    free(temp);
    count--;
}

void deleteEnd() {
    if (!head) {
        printf("List is empty.\n");
        return;
    }
    if (!head->link) {
        printf("Deleted: %s\n", head->usn);
        free(head);
        head = NULL;
    } else {
        NODE cur = head, prev = NULL;
        while (cur->link) {
            prev = cur;
            cur = cur->link;
        }
        printf("Deleted: %s\n", cur->usn);
        free(cur);
        prev->link = NULL;
    }
    count--;
}

void display() {
    if (!head) {
        printf("List is empty.\n");
        return;
    }
    NODE cur = head;
    printf("SLL Contents:\n");
    while (cur) {
        printf("USN: %s, Name: %s, Programme: %s, Sem: %d, PhNo: %ld\n",
               cur->usn, cur->name, cur->prog, cur->sem, cur->phNo);
        cur = cur->link;
    }
    printf("Total nodes: %d\n", count);
}
```

```
int main() {
  int choice;
  do {
    printf("\n1. Insert at Front\n2. Insert at End\n3. Delete from Front\n4. Delete from End\n5.
Display\n6. Exit\nEnter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
    case 1: insertFront(); break;
    case 2: insertEnd(); break;
    case 3: deleteFront(); break;
    case 4: deleteEnd(); break;
    case 5: display(); break;
    case 6: printf("Exiting...\n"); break;
    default: printf("Invalid choice!\n");
    }
  } while (choice != 6);
  return 0;
}
```

Program 8 :

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Node {

   char ssn[20], name[30], dept[20], desg[20];

   float sal;

   long phNo;

   struct Node *prev, *next;

};


typedef struct Node *NODE;

NODE head = NULL;

int count = 0;


NODE createNode() {

   NODE temp = (NODE)malloc(sizeof(struct Node));

   printf("Enter SSN, Name, Dept, Designation, Salary, Phone: ");

   scanf("%s %s %s %s %f %ld", temp->ssn, temp->name, temp->dept, temp->desg, &temp->sal,
&temp->phNo);

   temp->prev = temp->next = NULL;

   count++;

   return temp;

}


void insertEnd() {

   NODE temp = createNode();

   if (!head) {

      head = temp;
```

```
  } else {
    NODE cur = head;
    while (cur->next) cur = cur->next;
    cur->next = temp;
    temp->prev = cur;
  }
}

void insertFront() {
  NODE temp = createNode();
  if (!head) {
    head = temp;
  } else {
    temp->next = head;
    head->prev = temp;
    head = temp;
  }
}

void deleteEnd() {
  if (!head) {
    printf("List is empty.\n");
    return;
  }
  NODE temp = head;
  if (!head->next) {
    printf("Deleted: %s\n", head->ssn);
    free(head);
    head = NULL;
  } else {
        while (temp->next) temp = temp->next;
        printf("Deleted: %s\n", temp->ssn);
        temp->prev->next = NULL;
        free(temp);
    }
    count--;
}

void deleteFront() {
    if (!head) {
        printf("List is empty.\n");
        return;
    }
    NODE temp = head;
    printf("Deleted: %s\n", temp->ssn);
    head = head->next;
    if (head) head->prev = NULL;
    free(temp);
    count--;
}

void display() {
    if (!head) {
        printf("List is empty.\n");
        return;
    }
    NODE cur = head;
    printf("DLL Contents:\n");
    while (cur) {
        printf("SSN: %s, Name: %s, Dept: %s, Designation: %s, Salary: %.2f, PhNo: %ld\n",
            cur->ssn, cur->name, cur->dept, cur->desg, cur->sal, cur->phNo);
        cur = cur->next;
    }
    printf("Total nodes: %d\n", count);
}

int main() {
    int choice;
    do {
        printf("\n1. Create DLL (Insert at End)\n2. Display DLL and Count Nodes\n");
        printf("3. Insert at Front\n4. Delete from Front\n5. Insert/Delete at End\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            insertEnd();
            break;
        case 2:
            display();
            break;
        case 3:
            insertFront();
            break;
        case 4:
            deleteFront();
            break;
        case 5:
            printf("1. Insert at End\n2. Delete from End\n");
            int subChoice;
            scanf("%d", &subChoice);
            if (subChoice == 1)
                insertEnd();
            else if (subChoice == 2)
                deleteEnd();
            else
                printf("Invalid choice!\n");
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 6);

    return 0;
}
```

Program 9 :

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

// Define the structure for a node
struct Node {
  int coeff, x, y, z;  // Coefficient and powers of x, y, z
  struct Node *next;
};

typedef struct Node *NODE;

// Create a new node
NODE createNode(int coeff, int x, int y, int z) {
  NODE temp = (NODE)malloc(sizeof(struct Node));
  temp->coeff = coeff;
  temp->x = x;
  temp->y = y;
  temp->z = z;
  temp->next = temp; // Circular link
  return temp;
}

// Insert a term at the end
NODE insertEnd(NODE head, int coeff, int x, int y, int z) {
  NODE temp = createNode(coeff, x, y, z);
  if (!head) return temp;
  NODE cur = head;
  while (cur->next != head) cur = cur->next;
  cur->next = temp;
```

```
    temp->next = head;
    return head;
}

// Display the polynomial
void display(NODE head) {
    if (!head) {
        printf("Polynomial is empty.\n");
        return;
    }
    NODE cur = head;
    do {
        printf("%+dx^%dy^%dz^%d ", cur->coeff, cur->x, cur->y, cur->z);
        cur = cur->next;
    } while (cur != head);
    printf("\n");
}

// Add two polynomials
NODE addPolynomials(NODE poly1, NODE poly2) {
    NODE sum = NULL;
    NODE p1 = poly1, p2;
    do {
        sum = insertEnd(sum, p1->coeff, p1->x, p1->y, p1->z);
        p1 = p1->next;
    } while (p1 != poly1);

    p2 = poly2;
    do {
        NODE temp = sum;
        int found = 0;
        do {
            if (temp->x == p2->x && temp->y == p2->y && temp->z == p2->z) {
                temp->coeff += p2->coeff;
                found = 1;
                break;
            }
            temp = temp->next;
        } while (temp != sum);

        if (!found) sum = insertEnd(sum, p2->coeff, p2->x, p2->y, p2->z);
        p2 = p2->next;
    } while (p2 != poly2);

    return sum;
}

int main() {
    NODE poly1 = NULL, poly2 = NULL, polySum = NULL;

    // Create POLY1
    printf("Creating POLY1...\n");
    poly1 = insertEnd(poly1, 6, 2, 2, 1);  // 6x^2y^2z
    poly1 = insertEnd(poly1, -4, 0, 1, 5); // -4yz^5
    poly1 = insertEnd(poly1, 3, 3, 1, 1);  // 3x^3yz
    poly1 = insertEnd(poly1, 2, 1, 5, 1);  // 2xy^5z
    poly1 = insertEnd(poly1, -2, 1, 1, 3); // -2xyz^3

    // Create POLY2
    printf("Creating POLY2...\n");
    poly2 = insertEnd(poly2, 1, 2, 2, 1);  // 1x^2y^2z

    poly2 = insertEnd(poly2, 5, 0, 1, 5);  // 5yz^5

    poly2 = insertEnd(poly2, -3, 3, 1, 1); // -3x^3yz


    // Display POLY1 and POLY2

    printf("POLY1: ");

    display(poly1);

    printf("POLY2: ");

    display(poly2);


    // Add POLY1 and POLY2

    printf("Adding POLY1 and POLY2...\n");

    polySum = addPolynomials(poly1, poly2);


    // Display POLYSUM

    printf("POLYSUM: ");

    display(polySum);


    return 0;

}
```

Program 10 :

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

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

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

struct Node* insert(struct Node* root, int data) {
   if (root == NULL) return createNode(data);
   if (data < root->data)
      root->left = insert(root->left, data);
   else if (data > root->data)
      root->right = insert(root->right, data);
   return root;
}

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 search(struct Node* root, int key) {
    if (root == NULL) {
        printf("Key %d not found.\n", key);
        return;
    }
    if (root->data == key) {
        printf("Key %d found in the BST.\n", key);
        return;
    }
    if (key < root->data)
        search(root->left, key);
    else
        search(root->right, key);
}

int main() {
    struct Node* root = NULL;
    int choice, key, i;
    int elements[] = {6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2};
    int n = sizeof(elements) / sizeof(elements[0]);

    while (1) {
        printf("\n--- BST Menu ---\n");
        printf("1. Create BST\n");
        printf("2. Traverse BST (Inorder, Preorder, Postorder)\n");
        printf("3. Search for a key\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Creating BST with elements: ");
                for (i = 0; i < n; i++) {
                    printf("%d ", elements[i]);
                    root = insert(root, elements[i]);
                }
                printf("\nBST created successfully.\n");
                break;
            case 2:
                printf("Inorder: ");
                inorder(root);
                printf("\nPreorder: ");
                preorder(root);
                printf("\nPostorder: ");
                postorder(root);
                printf("\n");
                break;
            case 3:
                printf("Enter key to search: ");
                scanf("%d", &key);
                search(root, key);
                break;
            case 4:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
    return 0;
}
```