1. **Develop a Program in C for the following:**
   a) **Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).**
   b) **Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.**

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

// Structure to represent a day
struct Day {
    char *dayName;
    int date;
    char *activity;
};

// Function to create a day in the calendar
void create(struct Day calendar[], int index) {
    char temp[50];

    printf("Enter the name of the day: ");
    fgets(temp, sizeof(temp), stdin);
    temp[strcspn(temp, "\n")] = '\0'; // Remove newline character if present

    calendar[index].dayName = (char *)malloc(strlen(temp) + 1);
    if (calendar[index].dayName == NULL) {
        perror("Failed to allocate memory");
        exit(EXIT_FAILURE);
    }
    strcpy(calendar[index].dayName, temp);

    printf("Enter the date of the day: ");
    scanf("%d", &calendar[index].date);
    getchar(); // Consume the newline character left by scanf

    printf("Enter the activity for the day: ");
    fgets(temp, sizeof(temp), stdin);
    temp[strcspn(temp, "\n")] = '\0'; // Remove newline character if present

    calendar[index].activity = (char *)malloc(strlen(temp) + 1);
    if (calendar[index].activity == NULL) {
        perror("Failed to allocate memory");
        exit(EXIT_FAILURE);
    }
```

```
    strcpy(calendar[index].activity, temp);
}

// Function to read data from the keyboard for all days
void read(struct Day calendar[]) {
    for (int i = 0; i < 7; i++) {
        printf("Enter details for Day %d:\n", i + 1);
        create(calendar, i);
    }
}

// Function to display the calendar
void display(struct Day calendar[]) {
    printf("\nDay\tDate\tActivity\n");
    for (int i = 0; i < 7; i++) {
        printf("%s\t%d\t%s\n", calendar[i].dayName, calendar[i].date, calendar[i].activity);
    }
}

int main() {
    // Declare an array of Day structures
    struct Day calendar[7];

    // Read data for each day
    read(calendar);

    // Display the calendar
    display(calendar);

    // Free dynamically allocated memory
    for (int i = 0; i < 7; i++) {
        free(calendar[i].dayName);
        free(calendar[i].activity);
    }

    return 0;
}
```

OUTPUT

```
Enter details for Day 1:
Enter the name of the day: Monday
Enter the date of the day: 1
Enter the activity for the day: Gym
Enter details for Day 2:
Enter the name of the day: Tuesday
Enter the date of the day: 2
Enter the activity for the day: Work
Enter details for Day 3:
Enter the name of the day: Wednesday
Enter the date of the day: 3
Enter the activity for the day: Study
Enter details for Day 4:
Enter the name of the day: Thursday
Enter the date of the day: 4
Enter the activity for the day: Shopping
Enter details for Day 5:
Enter the name of the day: Friday
Enter the date of the day: 5
Enter the activity for the day: Movie
Enter details for Day 6:
Enter the name of the day: Saturday
Enter the date of the day: 6
Enter the activity for the day: Rest
Enter details for Day 7:
Enter the name of the day: Sunday
Enter the date of the day: 7
Enter the activity for the day: Family

Day    Date    Activity
Monday 1       Gym
Tuesday 2      Work
Wednesday 3    Study
Thursday 4     Shopping
Friday 5       Movie
Saturday 6     Rest
Sunday 7       Family
```

2) Develop a Program in C for the following operations on Strings.

a) Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)

b) Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR

Support the program with functions for each of the above operations. Don't use Built-in functions.

```
#include <stdio.h>
void main() {
   char s[20], pat[20], rep[20], ans[30];
   int i, j, k, l, flag;

   printf("\nEnter string:");
   scanf("%s", s);

   printf("\nEnter pattern:");
   scanf("%s", pat);

   printf("\nEnter replacement:");
   scanf("%s", rep);

   for(i = 0, k = 0; s[i] != '\0'; i++) {
      flag = 1;

      for(j = 0; pat[j] != '\0'; j++) {
         if(s[i + j] != pat[j]) {
            flag = 0;
         }
      }

      l = j;

      if(flag) {
         for(j = 0; rep[j] != '\0'; j++, k++) {
            ans[k] = rep[j];
         }
         i += l - 1;
      } else {
         ans[k++] = s[i];
      }
   }

   ans[k] = '\0';
   printf("%s", ans);
}
```

OUTPUT

Example 1:

Input:

Enter string: Program
Enter pattern: am
Enter replacement: ams

Output:

Programs

Example 2:

Input:

Enter string: abcef
Enter pattern: ce
Enter replacement: cde

Output:

Abcdef

3) Develop a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

a) Push an Element on to Stack
b) Pop an Element from Stack
c) Demonstrate how Stack can be used to check Palindrome
d) Demonstrate Overflow and Underflow situations on Stack
e) Display the status of Stack
f) Exit

Support the program with appropriate functions for each of the above operations

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

int s[5], top = -1;

void push() {
   if (top == 4)
      printf("\nStack overflow!!!!");
   else {
      printf("\nEnter element to insert:");
      scanf("%d", &s[++top]);
   }
}

void pop() {
   if (top == -1)
      printf("\nStack underflow!!!");
   else
      printf("\nElement popped is: %d", s[top--]);
}

void disp() {
   int t = top;
   if (t == -1)
      printf("\nStack empty!!");
   else
      printf("\nStack elements are:\n");
   while (t >= 0)
      printf("%d ", s[t--]);
}

void pali() {
   int num[5], rev[5], i, t;
   for (i = 0, t = top; t >= 0; i++, t--)
      num[i] = rev[t] = s[t];
   for (i = 0; i <= top; i++)
      if (num[i] != rev[i])
         break;
   /*printf(" num     rev\n");
```

```
    for (t = 0; t <= top; t++)
        printf("%4d   %4d\n", num[t], rev[t]);*/ //remove /* */ to display num and rev
    if (i == top + 1)
        printf("\nIt is a palindrome");
    else
        printf("\nIt is not a palindrome");
}

int main() {
    int ch;
    do {
        printf("\n...Stack operations.....\n");
        printf("1.PUSH\n");
        printf("2.POP\n");
        printf("3.Palindrome\n");
        printf("4.Display\n");
        printf("5.Exit\n_____\n");
        printf("Enter choice:");
        scanf("%d", &ch);
        switch (ch) {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: pali(); break;
            case 4: disp(); break;
            case 5: exit(0);
            default: printf("\nInvalid choice");
        }
    } while (1);
    return 0;
}
```

OUTPUT

```
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____
Enter choice:1

Enter element to insert:10

...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____
Enter choice:1

Enter element to insert:20

...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____
Enter choice:1

Enter element to insert:30

...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____
Enter choice:1

Enter element to insert:40

...Stack operations.....
1.PUSH
2.POP
```

```
3.Palindrome
4.Display
5.Exit

_____________
Enter choice:1

Enter element to insert:50

...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____________
Enter choice:4

Stack elements are:
50 40 30 20 10
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____________
Enter choice:1

Stack overflow!!!!
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____________
Enter choice:60

Invalid choice
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____________
Enter choice:3
```

It is not a palindrome
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____
Enter choice:2

Element popped is: 50
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____
Enter choice:2

Element popped is: 40
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____
Enter choice:2

Element popped is: 30
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____
Enter choice:2

Element popped is: 20
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit

_____

```
Enter choice:2

Element popped is: 10
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit
_________________
Enter choice:2

Stack underflow!!!
...Stack operations.....
1.PUSH
2.POP
3.Palindrome
4.Display
5.Exit
_________________
Enter choice:5
```

4) Develop a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

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

int F(char symbol) {
   switch (symbol) {
      case '+':
      case '-': return 2;
      case '*':
      case '/':
      case '%': return 4;
      case '^':
      case '$': return 5;
      case '(': return 0;
      case '#': return -1;
      default: return 8;
   }
}

int G(char symbol) {
   switch (symbol) {
      case '+':
      case '-': return 1;
      case '*':
      case '/':
      case '%': return 3;
      case '^':
      case '$': return 6;
      case '(': return 3;
      case ')': return 0;
      default: return 7;
   }
}

void infix_postfix(char infix[], char postfix[]) {
   int top = -1, j = 0, i;
   char s[30], symbol;
   s[++top] = '#';
   for (i = 0; i < strlen(infix); i++) {
      symbol = infix[i];
      while (F(s[top]) > G(symbol)) {
         postfix[j++] = s[top--];
      }
      if (F(s[top]) != G(symbol)) {
         s[++top] = symbol;
      } else {
         top--;
      }
   }
   while (s[top] != '#') {
      postfix[j++] = s[top--];
   }
```

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

void main() {
    char infix[20], postfix[20];
    printf("\nEnter a valid infix expression\n");
    scanf("%s", infix);
    infix_postfix(infix, postfix);
    printf("\nThe infix expression is:\n");
    printf("%s", infix);
    printf("\nThe postfix expression is:\n");
    printf("%s", postfix);
}
```

OUTPUT

```
Enter a valid infix expression
a+b-x*d+a

The infix expression is:
a+b-x*d+a

The postfix expression is:
ab+xd*-a+
```

5) Develop a Program in C for the following Stack Applications

a) Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %,^

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <ctype.h> // For isdigit()

// Function to compute the result based on the operator and two operands
float compute(char symbol, float op1, float op2) {
   switch (symbol) {
      case '+': return op1 + op2;
      case '-': return op1 - op2;
      case '*': return op1 * op2;
      case '/': return op1 / op2;
      case '^': // Power operation
      case '$': return pow(op1, op2);
      default : return 0;
   }
}

int main() {
   float stack[20], result, op1, op2;
   int top = -1, i;
   char postfix[20], symbol;

   // Input the postfix expression
   printf("\nEnter the postfix expression:\n");
   scanf("%s", postfix);

   // Evaluate the postfix expression
   for (i = 0; i < strlen(postfix); i++) {
      symbol = postfix[i];

      // If the symbol is a digit, push it onto the stack
      if (isdigit(symbol)) {
         stack[++top] = symbol - '0';  // Convert character to integer
      } else {
         // If the symbol is an operator, pop two operands from the stack
         op2 = stack[top--];
         op1 = stack[top--];

         // Compute the result and push it back to the stack
         result = compute(symbol, op1, op2);
         stack[++top] = result;
      }
   }

   // The final result will be at the top of the stack
   result = stack[top--];
   printf("\nThe result is: %f\n", result);

   return 0;
}
```

OUTPUT:

Output 1:
Enter the postfix expression:
23+3*45+

The result is : 29.000000

Output 2:
Enter the postfix expression:
25+8-64*

The result is: -16.000000

b) Solving Tower of Hanoi problem with n disks

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

// Function to solve Tower of Hanoi recursively
void towerOfHanoi(int n, char source, char temp, char destination) {
   if (n == 0) {
      return;
   }
   // Move n-1 discs from source to temp using destination as auxiliary
   towerOfHanoi(n - 1, source, destination, temp);

   // Move nth disc from source to destination
   printf("\nMove disc %d from %c to %c", n, source, destination);

   // Move n-1 discs from temp to destination using source as auxiliary
   towerOfHanoi(n - 1, temp, source, destination);
}

int main() {
   int n;

   // Input the number of discs
   printf("\nEnter the number of discs: \n\n");
   scanf("%d", &n);

   // Output the sequence of moves
   printf("\nThe sequence of moves involved in the Tower of Hanoi are:\n");
   towerOfHanoi(n, 'A', 'B', 'C');

   // Output the total number of moves
   printf("\n\nTotal Number of moves are: %d\n", (int)pow(2, n) - 1);

   return 0;
}
```

OUTPUT:

Enter the number of discs:

3

The sequence of moves involved in the Tower of Hanoi are:

Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C

Total Number of moves are: 7

6) Develop a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)
- **a) Insert an Element on to Circular QUEUE**
- **b) Delete an Element from Circular QUEUE**
- **c) Demonstrate Overflow and Underflow situations on Circular QUEUE**
- **d) Display the status of Circular QUEUE**
- **e) Exit**

Support the program with appropriate functions for each of the above operations

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

#define MAX 5  // Define the maximum size of the queue

int queue[MAX], front = -1, rear = -1;  // Initialize the queue, front, and rear

// Function to insert an element into the circular queue
void insert() {
   if (front == (rear + 1) % MAX) {
      printf("\nQueue overflow");
   } else {
      if (front == -1) {  // If the queue is empty, set front to 0
         front++;
      }
      rear = (rear + 1) % MAX;  // Circular increment of rear
      printf("\nEnter element to be inserted: ");
      scanf("%d", &queue[rear]);
   }
}

// Function to delete an element from the circular queue
void delete() {
   if (front == -1) {
      printf("\nQueue underflow");
   } else {
      printf("\nElement deleted is: %d", queue[front]);
      if (front == rear) {  // If the queue has only one element
         front = rear = -1;
      } else {
         front = (front + 1) % MAX;  // Circular increment of front
      }
   }
}

// Function to display the elements of the circular queue
void display() {
   if (front == -1) {
      printf("\nQueue is empty");
   } else {
      int i;
      printf("\nQueue elements are:\n");
      for (i = front; i != rear; i = (i + 1) % MAX) {
         printf("%d\t", queue[i]);
      }
      printf("%d", queue[i]);  // Print the last element at rear
```

```
        printf("\nFront is at: %d\nRear is at: %d", queue[front], queue[rear]);
    }
}

// Main function to handle circular queue operations
int main() {
    int choice;
    printf("\nCircular Queue operations");
    printf("\n1. Insert");
    printf("\n2. Delete");
    printf("\n3. Display");
    printf("\n4. Exit");

    do {
        printf("\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: insert(); break;   // Insert element
            case 2: delete(); break;   // Delete element
            case 3: display(); break;  // Display elements
            case 4: exit(0);           // Exit the program
            default: printf("\nInvalid choice...!");
        }
    } while (1);

    return 0;
}
```

OUTPUT:

```
Circular Queue operations
1. Insert
2. Delete
3. Display
4. Exit
Enter choice: 1

Enter element to be inserted: 10

Enter choice: 1

Enter element to be inserted: 20

Enter choice: 1

Enter element to be inserted: 30

Enter choice: 1

Enter element to be inserted: 40

Enter choice: 1

Enter element to be inserted: 50

Enter choice: 1

Queue overflow
Enter choice: 2

Element deleted is: 10
Enter choice: 1

Enter element to be inserted: 60

Enter choice: 2

Element deleted is: 20
Enter choice: 2

Element deleted is: 30
Enter choice: 2

Element deleted is: 40
Enter choice: 2
Element deleted is: 50
Enter choice: 2
Element deleted is: 60
Enter choice: 2
Queue underflow
Enter choice: 2

Queue underflow
Enter choice: 4
```

7. Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Programme, Sem, PhNo*

a) Create a SLL of N Students Data by using *front insertion*.
b) Display the status of SLL and count the number of nodes in it
c) Perform Insertion / Deletion at End of SLL
d) Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
e) Exit

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

// Define a structure for the student node
struct Student {
    char usn[15];
    char name[30];
    char programme[20];
    int sem;
    char phone[15];
    struct Student* next;
};

// Initialize the head pointer
struct Student* head = NULL;

// Function to create a new student node using front insertion
void createStudentAtFront() {
    struct Student* new_student = (struct Student*)malloc(sizeof(struct Student));

    printf("\nEnter USN: ");
    scanf("%s", new_student->usn);
    printf("Enter Name: ");
    scanf("%s", new_student->name);
    printf("Enter Programme: ");
    scanf("%s", new_student->programme);
    printf("Enter Semester: ");
    scanf("%d", &new_student->sem);
    printf("Enter Phone Number: ");
    scanf("%s", new_student->phone);

    new_student->next = head;
    head = new_student;
    printf("\nStudent record inserted at the front!\n");
}

// Function to display the list and count nodes
void displayList() {
    struct Student* temp = head;
    int count = 0;

    if (temp == NULL) {
        printf("\nThe list is empty.\n");
        return;
    }
```

```
    printf("\nThe student list is:\n");
    while (temp != NULL) {
        printf("USN: %s, Name: %s, Programme: %s, Sem: %d, Phone: %s\n",
               temp->usn, temp->name, temp->programme, temp->sem, temp->phone);
        temp = temp->next;
        count++;
    }
    printf("\nTotal number of students: %d\n", count);
}

// Function to insert a student node at the end
void insertAtEnd() {
    struct Student* new_student = (struct Student*)malloc(sizeof(struct Student));
    struct Student* temp = head;

    printf("\nEnter USN: ");
    scanf("%s", new_student->usn);
    printf("Enter Name: ");
    scanf("%s", new_student->name);
    printf("Enter Programme: ");
    scanf("%s", new_student->programme);
    printf("Enter Semester: ");
    scanf("%d", &new_student->sem);
    printf("Enter Phone Number: ");
    scanf("%s", new_student->phone);
    new_student->next = NULL;

    if (head == NULL) {
        head = new_student;
    } else {
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = new_student;
    }
    printf("\nStudent record inserted at the end!\n");
}

// Function to delete a student node from the end
void deleteAtEnd() {
    struct Student* temp = head;
    struct Student* prev = NULL;

    if (head == NULL) {
        printf("\nList is empty, nothing to delete.\n");
        return;
    }

    if (head->next == NULL) { // Only one node in the list
        printf("\nStudent record deleted: %s\n", head->usn);
        free(head);
        head = NULL;
    } else {
        while (temp->next != NULL) {
            prev = temp;
```

```
            temp = temp->next;
        }
        printf("\nStudent record deleted: %s\n", temp->usn);
        free(temp);
        prev->next = NULL;
    }
}

// Function to delete a student node from the front (demonstration of stack)
void deleteAtFront() {
    struct Student* temp = head;

    if (head == NULL) {
        printf("\nList is empty, nothing to delete.\n");
        return;
    }

    printf("\nStudent record deleted: %s\n", head->usn);
    head = head->next;
    free(temp);
}

// Menu-driven main function
int main() {
    int choice;

    while (1) {
        printf("\nMenu: ");
        printf("\n1. Create SLL using Front Insertion");
        printf("\n2. Display SLL and Count Nodes");
        printf("\n3. Insert at End of SLL");
        printf("\n4. Delete from End of SLL");
        printf("\n5. Delete from Front of SLL (Demonstration of Stack)");
        printf("\n6. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createStudentAtFront();
                break;
            case 2:
                displayList();
                break;
            case 3:
                insertAtEnd();
                break;
            case 4:
                deleteAtEnd();
                break;
            case 5:
                deleteAtFront();
                break;
            case 6:
                exit(0);
```

```
            break;
        default:
            printf("\nInvalid choice, please try again.\n");
    }
  }
  return 0;
}
```

OUTPUT:

```
Menu:
1. Create SLL using Front Insertion
2. Display SLL and Count Nodes
3. Insert at End of SLL
4. Delete from End of SLL
5. Delete from Front of SLL (Demonstration of Stack)
6. Exit
Enter your choice: 1

Enter USN: 1RV20CS001
Enter Name: John
Enter Programme: B.E
Enter Semester: 5
Enter Phone Number: 9876543210

Student record inserted at the front!

Menu:
1. Create SLL using Front Insertion
2. Display SLL and Count Nodes
3. Insert at End of SLL
4. Delete from End of SLL
5. Delete from Front of SLL (Demonstration of Stack)
6. Exit
Enter your choice: 2

The student list is:
USN: 1RV20CS001, Name: John, Programme: B.E, Sem: 5, Phone: 9876543210

Total number of students: 1
```

8) Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo

a) Create a DLL of N Employees Data by using end insertion.
b) Display the status of DLL and count the number of nodes in it.
c) Perform Insertion and Deletion at End of DLL.
d) Perform Insertion and Deletion at Front of DLL.
e) Demonstrate how this DLL can be used as Double Ended Queue.
f) Exit.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Enode {
   char ssn[15];
   char name[20];
   char dept[5];
   char designation[10];
   int salary;
   long long int phno;
   struct Enode *left;
   struct Enode *right;
} *head = NULL, *tail = NULL, *temp1 = NULL, *temp2 = NULL;

void create(char s[], char n[], char dpt[], char des[], int sal, long long int p);
void ins_beg(char s[], char n[], char dpt[], char des[], int sal, long long int p);
void ins_end(char s[], char n[], char dpt[], char des[], int sal, long long int p);
void del_beg();
void del_end();
void display();

int count = 0;

int main() {
   int choice;
   char s[15], n[20], dpt[5], des[10];
   int sal;
   long long int p;

   printf("1. Create\n2. Display\n3. Insert at beginning\n4. Insert at End\n5. Delete at beginning\n6.
Delete at End\n7. Exit\n");

   while (1) {
      printf("\nEnter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone): ");
            scanf("%s %s %s %s %d %lld", s, n, dpt, des, &sal, &p);
            create(s, n, dpt, des, sal, p);
            break;
         case 2:
            display();
```

```
            break;
        case 3:
            printf("Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone): ");
            scanf("%s %s %s %s %d %lld", s, n, dpt, des, &sal, &p);
            ins_beg(s, n, dpt, des, sal, p);
            break;
        case 4:
            printf("Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone): ");
            scanf("%s %s %s %s %d %lld", s, n, dpt, des, &sal, &p);
            ins_end(s, n, dpt, des, sal, p);
            break;
        case 5:
            del_beg();
            break;
        case 6:
            del_end();
            break;
        case 7:
            exit(0);
            break;
        default:
            printf("Invalid choice. Please enter a number between 1 and 7.\n");
            break;
    }
  }

  return 0;
}

void create(char s[], char n[], char dpt[], char des[], int sal, long long int p) {
  if (head == NULL) {
    head = (struct Enode *)malloc(sizeof(struct Enode));
    if (head == NULL) {
      printf("Memory allocation failed.\n");
      exit(1);
    }
    strcpy(head->ssn, s);
    strcpy(head->name, n);
    strcpy(head->dept, dpt);
    strcpy(head->designation, des);
    head->salary = sal;
    head->phno = p;
    head->left = NULL;
    head->right = NULL;
    tail = head;
  } else {
    temp1 = (struct Enode *)malloc(sizeof(struct Enode));
    if (temp1 == NULL) {
      printf("Memory allocation failed.\n");
      exit(1);
    }
    strcpy(temp1->ssn, s);
    strcpy(temp1->name, n);
    strcpy(temp1->dept, dpt);
    strcpy(temp1->designation, des);
```

```
        temp1->salary = sal;
        temp1->phno = p;
        tail->right = temp1;
        temp1->right = NULL;
        temp1->left = tail;
        tail = temp1;
    }
}

void display() {
    temp1 = head;
    if (temp1 == NULL) {
        printf("No employee details to display.\n");
        return;
    }

    printf("Employee Details:\n");
    while (temp1 != NULL) {
        printf("\nSSN: %s\nName: %s\nDept: %s\nDesignation: %s\nSalary: %d\nPhone: %lld\n",
            temp1->ssn, temp1->name, temp1->dept, temp1->designation, temp1->salary, temp1-
>phno);
        temp1 = temp1->right;
    }
}

void ins_beg(char s[], char n[], char dpt[], char des[], int sal, long long int p) {
    temp1 = (struct Enode *)malloc(sizeof(struct Enode));
    if (temp1 == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    strcpy(temp1->ssn, s);
    strcpy(temp1->name, n);
    strcpy(temp1->dept, dpt);
    strcpy(temp1->designation, des);
    temp1->salary = sal;
    temp1->phno = p;
    temp1->right = head;
    if (head != NULL) {
        head->left = temp1;
    }
    head = temp1;
    temp1->left = NULL;
}

void ins_end(char s[], char n[], char dpt[], char des[], int sal, long long int p) {
    temp1 = (struct Enode *)malloc(sizeof(struct Enode));
    if (temp1 == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    strcpy(temp1->ssn, s);
    strcpy(temp1->name, n);
    strcpy(temp1->dept, dpt);
    strcpy(temp1->designation, des);
```

```
    temp1->salary = sal;
    temp1->phno = p;
    temp1->right = NULL;
    if (tail != NULL) {
        tail->right = temp1;
        temp1->left = tail;
    }
    tail = temp1;
    if (head == NULL) {
        head = temp1;
    }
}

void del_beg() {
    if (head == NULL) {
        printf("List is empty, nothing to delete.\n");
        return;
    }
    temp1 = head->right;
    free(head);
    head = temp1;
    if (head != NULL) {
        head->left = NULL;
    } else {
        tail = NULL;
    }
}

void del_end() {
    if (tail == NULL) {
        printf("List is empty, nothing to delete.\n");
        return;
    }
    temp1 = tail->left;
    free(tail);
    tail = temp1;
    if (tail != NULL) {
        tail->right = NULL;
    } else {
        head = NULL;
    }
}
```

OUTPUT

1. Create
2. Display
3. Insert at beginning
4. Insert at End
5. Delete at beginning
6. Delete at End
7. Exit

Enter your choice: 1

Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone): 12345 John IT Manager 50000 9876543210
Enter your choice: 2
Employee Details:

SSN: 12345
Name: John
Dept: IT
Designation: Manager
Salary: 50000
Phone: 9876543210

Enter your choice: 3
Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone): 67890 Alice HR Executive 45000 1234567890

Enter your choice: 2
Employee Details:

SSN: 67890
Name: Alice
Dept: HR
Designation: Executive
Salary: 45000
Phone: 1234567890

SSN: 12345
Name: John
Dept: IT
Designation: Manager
Salary: 50000
Phone: 9876543210

Enter your choice: 4
Enter the required data (Emp no, Name, Dept, Desig, Sal, Phone): 54321 Bob Marketing Specialist 55000 1112233445

Enter your choice: 2
Employee Details:

SSN: 67890
Name: Alice
Dept: HR
Designation: Executive
Salary: 45000
Phone: 1234567890

SSN: 12345
Name: John
Dept: IT
Designation: Manager
Salary: 50000
Phone: 9876543210

SSN: 54321

Name: Bob
Dept: Marketing
Designation: Specialist
Salary: 55000
Phone: 1112233445

Enter your choice: 5

Enter your choice: 2
Employee Details:

SSN: 12345
Name: John
Dept: IT
Designation: Manager
Salary: 50000
Phone: 9876543210

SSN: 54321
Name: Bob
Dept: Marketing
Designation: Specialist
Salary: 55000
Phone: 1112233445
Enter your choice: 7

9) Develop a Program in C for the following operationson Singly Circular Linked List (SCLL) with header nodes

a) Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3

b) Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)

Support the program with appropriate functions for each of the above operations

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

struct node {
   int coeff;
   int expo;
   struct node *ptr;
};

struct node *head1 = NULL, *head2 = NULL, *head3 = NULL;
struct node *list1 = NULL, *list2 = NULL, *list3 = NULL;
struct node *dummy1 = NULL, *dummy2 = NULL;
int n, ch, c, e, i;

void create_poly1(int, int);
void create_poly2(int, int);
void display();
void add_poly();
void eval_poly(int);

int main() {
   int x;

   printf("1. Create first polynomial\n");
   printf("2. Create second polynomial\n");
   printf("3. Display both polynomials\n");
   printf("4. Add polynomials\n");
   printf("5. Evaluate a polynomial\n");
   printf("6. Exit\n");

   while(1) {
      printf("Enter choice\n");
      scanf("%d", &ch);

      switch(ch) {
         case 1:
            printf("Enter the number of terms\n");
            scanf("%d", &n);
            printf("Enter coefficient and power of each term\n");
            for(i = 0; i < n; i++) {
               scanf("%d%d", &c, &e);
               create_poly1(c, e);
```

```
            }
            break;

        case 2:
            printf("Enter the number of terms\n");
            scanf("%d", &n);
            printf("Enter coefficient and power of each term\n");
            for(i = 0; i < n; i++) {
               scanf("%d%d", &c, &e);
               create_poly2(c, e);
            }
            break;

        case 3:
            display();
            break;

        case 4:
            add_poly();
            break;

        case 5:
            printf("Enter the value for x\n");
            scanf("%d", &x);
            eval_poly(x);
            break;

        case 6:
            exit(0);
      }
   }

   return 0;
}

void create_poly1(int c, int e) {
   struct node *temp = (struct node*)malloc(sizeof(struct node));

   temp->coeff = c;
   temp->expo = e;
   temp->ptr = NULL;

   if(list1 == NULL) {
      list1 = temp;
      head1 = list1;
   } else {
      head1->ptr = temp;
      head1 = temp;
```

```
    }

    dummy1 = (struct node*)malloc(sizeof(struct node));
    dummy1->coeff = 0;
    dummy1->expo = 0;
    dummy1->ptr = list1;
    head1->ptr = dummy1;
}

void create_poly2(int c, int e) {
    struct node *temp = (struct node*)malloc(sizeof(struct node));

    temp->coeff = c;
    temp->expo = e;
    temp->ptr = NULL;

    if(list2 == NULL) {
        list2 = temp;
        head2 = list2;
    } else {
        head2->ptr = temp;
        head2 = temp;
    }

    dummy2 = (struct node*)malloc(sizeof(struct node));
    dummy2->coeff = 0;
    dummy2->expo = 0;
    dummy2->ptr = list2;
    head2->ptr = dummy2;
}

void add_poly() {
    struct node *temp1 = list1, *temp2 = list2, *temp;

    while((temp1 != dummy1) && (temp2 != dummy2)) {
        temp = (struct node*)malloc(sizeof(struct node));

        if(list3 == NULL) {
            list3 = temp;
            head3 = list3;
        }

        if(temp1->expo == temp2->expo) {
            temp->coeff = temp1->coeff + temp2->coeff;
            temp->expo = temp1->expo;
            temp1 = temp1->ptr;
            temp2 = temp2->ptr;
        } else if(temp1->expo > temp2->expo) {
```

```
            temp->coeff = temp1->coeff;
            temp->expo = temp1->expo;
            temp1 = temp1->ptr;
        } else {
            temp->coeff = temp2->coeff;
            temp->expo = temp2->expo;
            temp2 = temp2->ptr;
        }

        temp->ptr = NULL;
        head3->ptr = temp;
        head3 = temp;
    }

    while(temp1 != dummy1) {
        temp = (struct node*)malloc(sizeof(struct node));
        temp->coeff = temp1->coeff;
        temp->expo = temp1->expo;
        temp->ptr = NULL;
        head3->ptr = temp;
        head3 = temp;
        temp1 = temp1->ptr;
    }

    while(temp2 != dummy2) {
        temp = (struct node*)malloc(sizeof(struct node));
        temp->coeff = temp2->coeff;
        temp->expo = temp2->expo;
        temp->ptr = NULL;
        head3->ptr = temp;
        head3 = temp;
        temp2 = temp2->ptr;
    }
}

void display() {
    struct node *temp1 = list1, *temp2 = list2, *temp3 = list3;

    printf("\nPOLYNOMIAL 1: ");
    while(temp1 != dummy1) {
        printf("%dX^%d + ", temp1->coeff, temp1->expo);
        temp1 = temp1->ptr;
    }
    printf("\b\b ");  // Remove the last "+" character

    printf("\nPOLYNOMIAL 2: ");
    while(temp2 != dummy2) {
        printf("%dX^%d + ", temp2->coeff, temp2->expo);
```

```
        temp2 = temp2->ptr;
    }
    printf("\b\b ");

    printf("\n\nSUM OF POLYNOMIALS:\n");
    while(temp3 != NULL && temp3->ptr != list3) {
        printf("%dX^%d + ", temp3->coeff, temp3->expo);
        temp3 = temp3->ptr;
    }
    if(temp3 != NULL) {
        printf("%dX^%d\n", temp3->coeff, temp3->expo);
    }
}

void eval_poly(int x) {
    int result = 0;
    struct node *temp1 = list1, *temp2 = list2;

    while(temp1 != dummy1) {
        result += (temp1->coeff) * pow(x, temp1->expo);
        temp1 = temp1->ptr;
    }
    printf("Polynomial 1 Evaluation: %d\n", result);

    result = 0;
    while(temp2 != dummy2) {
        result += (temp2->coeff) * pow(x, temp2->expo);
        temp2 = temp2->ptr;
    }
    printf("Polynomial 2 Evaluation: %d\n", result);
}
```

OUTPUT:

```
<< MENU >>
Polynomial Operations:
1. Add
2. Evaluate
3. Exit
---------------------------
Enter your choice ==> 1
Enter number of terms of polynomial ==> 3
Enter coefficient & exponent:
4 3
Enter coefficient & exponent:
2 2
Enter coefficient & exponent:
5 1

The polynomial is ==> 5x^(1) + 2x^(2) + 4x^(3)

Enter number of terms of polynomial ==> 3
Enter coefficient & exponent:
4 1
Enter coefficient & exponent:
3 2
Enter coefficient & exponent:
5 3

The polynomial is ==> 4x^(1) + 3x^(2) + 5x^(3)

Addition of polynomials:
The resultant polynomial is ==> 9x^(1) + 5x^(2) + 9x^(3)
---------------------------
Enter your choice ==> 2

Enter number of terms of polynomial ==> 3
Enter coefficient & exponent:
1 1
Enter coefficient & exponent:
4 2
Enter coefficient & exponent:
5 4
The polynomial is ==> 1x^(1) + 4x^(2) + 5x^(4)

Enter the value of x: 2
Value of polynomial = 102

---------------------------
Enter your choice ==> 3
Exiting...
```

10) Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers

a) Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2

b) Traverse the BST in Inorder, Preorder and Post Order

c) Search the BST for a given element (KEY) and report the appropriate message

d) Delete an element(ELEM) from BST

e) Exit

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

struct BST {
   int data;
   struct BST *left;
   struct BST *right;
};

typedef struct BST NODE;
NODE* createtree(NODE *node, int data) {
   if (node == NULL) {
      NODE *temp = (NODE*)malloc(sizeof(NODE));
      temp->data = data;
      temp->left = temp->right = NULL;
      return temp;
   }
   if (data < node->data) {
      node->left = createtree(node->left, data);
   } else if (data > node->data) {
      node->right = createtree(node->right, data);
   }
   return node;
}

NODE* search(NODE *node, int data) {
   if (node == NULL) {
      printf("\nElement not found");
   } else if (data < node->data) {
      node->left = search(node->left, data);
   } else if (data > node->data) {
      node->right = search(node->right, data);
   } else {
      printf("\nElement found: %d", node->data);
   }
   return node;
}

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

```
        printf("%d\t", node->data);
        inorder(node->right);
    }
}

void preorder(NODE *node) {
    if (node != NULL) {
        printf("%d\t", node->data);
        preorder(node->left);
        preorder(node->right);
    }
}

void postorder(NODE *node) {
    if (node != NULL) {
        postorder(node->left);
        postorder(node->right);
        printf("%d\t", node->data);
    }
}

NODE* findMin(NODE *node) {
    if (node == NULL) {
        return NULL;
    }
    if (node->left != NULL) {
        return findMin(node->left);
    }
    return node;
}

NODE* del(NODE *node, int data) {
    NODE *temp;
    if (node == NULL) {
        printf("\nElement not found");
    } else if (data < node->data) {
        node->left = del(node->left, data);
    } else if (data > node->data) {
        node->right = del(node->right, data);
    } else {
        // Node found
        if (node->right && node->left) {
            // Replace with the minimum element in the right subtree
            temp = findMin(node->right);
            node->data = temp->data;
            node->right = del(node->right, temp->data);
        } else {
            // Node with one child or no child
```

```
            temp = node;
            if (node->left == NULL) {
               node = node->right;
            } else if (node->right == NULL) {
               node = node->left;
            }
            free(temp);
         }
      }
      return node;
}

int main() {
      int data, ch, i, n;
      NODE *root = NULL;

      while (1) {
         printf("\n1. Insert in Binary Search Tree");
         printf("\n2. Search Element in Binary Search Tree");
         printf("\n3. Delete Element in Binary Search Tree");
         printf("\n4. Inorder Traversal");
         printf("\n5. Preorder Traversal");
         printf("\n6. Postorder Traversal");
         printf("\n7. Exit");
         printf("\nEnter your choice: ");
         scanf("%d", &ch);

         switch (ch) {
            case 1:
               printf("\nEnter N value: ");
               scanf("%d", &n);
               printf("\nEnter the values to create BST (e.g., 6, 9, 5, 2, 8, ...):\n");
               for (i = 0; i < n; i++) {
                  scanf("%d", &data);
                  root = createtree(root, data);
               }
               break;
            case 2:
               printf("\nEnter the element to search: ");
               scanf("%d", &data);
               search(root, data);
               break;
            case 3:
               printf("\nEnter the element to delete: ");
               scanf("%d", &data);
               root = del(root, data);
               break;
            case 4:
```

```
            printf("\nInorder Traversal: \n");
            inorder(root);
            break;
        case 5:
            printf("\nPreorder Traversal: \n");
            preorder(root);
            break;
        case 6:
            printf("\nPostorder Traversal: \n");
            postorder(root);
            break;
        case 7:
            exit(0);
        default:
            printf("\nInvalid choice! Please try again.");
    }
  }

  return 0;
}
```

OUTPUT:

```
Program for Binary Search Tree
    1. Create
    2. Search
    3. Recursive Traversals
    4. Exit
Enter your choice: 1
Enter the element: 15
Do you want to enter more elements? (1 for yes, 0 for no): 1
Enter the element: 25
Do you want to enter more elements? (1 for yes, 0 for no): 1
Enter the element: 35
Do you want to enter more elements? (1 for yes, 0 for no): 1
Enter the element: 45
Do you want to enter more elements? (1 for yes, 0 for no): 1
Enter the element: 5
Do you want to enter more elements? (1 for yes, 0 for no): 1
Enter the element: 7
Do you want to enter more elements? (1 for yes, 0 for no): 0
Enter your choice: 2
Enter the element to be searched: 7
The element 7 is present.
Parent of node 7 is 5.
    1. Create
    2. Search
    3. Recursive Traversals
    4. Exit
Enter your choice: 2
Enter the element to be searched: 88
The element 88 is not present.
Enter your choice: 3
In-order traversal: 5 7 15 25 35 45
Pre-order traversal: 15 5 7 25 35 45
Post-order traversal: 7 5 45 35 25 15
Enter your choice: 4
Exiting the program...
```

11) Develop a Program in C for the following operations on Graph(G) of Cities

a) Create a Graph of N cities using Adjacency Matrix.

b) Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

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

int a[20][20], q[20], visited[20], reach[10], n, i, j, f = 0, r = -1, count = 0;

void bfs(int v) {
   for (i = 1; i <= n; i++) {
      if (a[v][i] && !visited[i])
         q[++r] = i;
   }
   if (f <= r) {
      visited[q[f]] = 1;
      bfs(q[f++]);
   }
}

void dfs(int v) {
   int i;
   reach[v] = 1;
   for (i = 1; i <= n; i++) {
      if (a[v][i] && !reach[i]) {
         printf("\n %d -> %d", v, i);
         count++;
         dfs(i);
      }
   }
}

int main() {
   int v, choice;

   printf("\n Enter the number of vertices: ");
   scanf("%d", &n);

   // Initialize the queue and visited array
   for (i = 1; i <= n; i++) {
      q[i] = 0;
      visited[i] = 0;
   }

   // Initialize the reach array
   for (i = 1; i <= n; i++)
      reach[i] = 0;
```

```
    printf("\n Enter the graph data in matrix form:\n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &a[i][j]);

    // Main menu
    while (1) {
        printf("\n1. BFS\n2. DFS\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("\n Enter the starting vertex: ");
                scanf("%d", &v);

                if (v < 1 || v > n) {
                    printf("\n BFS is not possible");
                } else {
                    bfs(v);
                    printf("\n The nodes reachable from vertex %d:\n", v);
                    for (i = 1; i <= n; i++) {
                        if (visited[i])
                            printf("%d\t", i);
                    }
                }
                break;

            case 2:
                dfs(1);
                if (count == n - 1)
                    printf("\n Graph is connected");
                else
                    printf("\n Graph is not connected");
                break;

            case 3:
                exit(0);
                break;

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

    return 0;
}
```

OUTPUT:

Output 1:

```
Enter the number of vertices: 4

Enter graph data in matrix form:
1 2 1 3
2 4 1 3
5 4 1 3
3 2 4 1

1. BFS
2. DFS
3. Exit
1

Enter the starting vertex: 1

The nodes which are reachable from 1:
1 2 3 4
```

Output 2:

```
Enter the number of vertices: 4
Enter graph data in matrix form:
3 2 4 1
4 3 5 1
2 1 3 2
3 4 2 5

1. BFS
2. DFS
3. Exit
2

1->2
2->3
3->4
Graph is connected
```

12) Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Develop a Program in C that uses Hash function H: K →L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

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

#define MAX_KEYS 20

int key[MAX_KEYS], n, m;
int *ht, index;
int count = 0;

// Function to insert a key into the hash table using linear probing
void insert(int key) {
   index = key % m;
   while (ht[index] != -1) {
      index = (index + 1) % m;
   }
   ht[index] = key;
   count++;
}

// Function to display the contents of the hash table
void display() {
   int i;
   if (count == 0) {
      printf("\nHash Table is empty");
      return;
   }

   printf("\nHash Table contents are:\n");
   for (i = 0; i < m; i++) {
      printf("T[%d] --> %d\n", i, ht[i]);
   }
}

int main() {
   int i;

   // Input number of employee records
   printf("\nEnter the number of employee records (N): ");
   scanf("%d", &n);

   // Input size of the hash table
```

```
    printf("\nEnter the size of the hash table (m): ");
    scanf("%d", &m);

    // Allocate memory for the hash table and initialize it
    ht = (int *)malloc(m * sizeof(int));
    if (ht == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    for (i = 0; i < m; i++) {
        ht[i] = -1;
    }

    // Input key values for employee records
    printf("\nEnter the key values (four-digit) for employee records:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &key[i]);
    }

    // Insert keys into the hash table
    for (i = 0; i < n; i++) {
        if (count == m) {
            printf("\n~~~Hash table is full. Cannot insert record %d key~~~\n", i + 1);
            break;
        }
        insert(key[i]);
    }

    // Display the hash table
    display();

    // Free allocated memory
    free(ht);

    return 0;
}
```

OUTPUT:

```
Enter the number of employee records (N): 12

Enter the size of the hash table (m): 15

Enter the key values (four-digit) of 'N' Employee Records:
1234
5678
3456
2345
6799
1235
7890
3214
3456
1235
5679
2346

Hash Table contents are:

T[0] --> 7890
T[1] --> -1
T[2] --> -1
T[3] --> -1
T[4] --> 1234
T[5] --> 2345
T[6] --> 3456
T[7] --> 6799
T[8] --> 5678
T[9] --> 1235
T[10] --> 3214
T[11] --> 3456
T[12] --> 1235
T[13] --> 5679
T[14] --> 2346
```