STUDENT DBMS

```
#include <stdio.h>
#include <string.h>
#define MAX 2  // Maximum number of students

struct Student {
    int rollNo;
    char name[50];
    float marks;
};

struct Student students[MAX];
int count = 0;

// Function prototypes
void addStudent();
void updateStudent();
void searchStudent();
void sortStudents();
void displayStudents();
void maxStudent();

int main() {
    int choice;
    do {
        printf("\n--- Student Database Management System ---\n");
        printf("1. Add New Entry\n2. Update Record\n3. Search Student\n");
        printf("4. Sort by Marks\n5. Display All\n6. Max Marks\n7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addStudent(); break;
            case 2: updateStudent(); break;
            case 3: searchStudent(); break;
            case 4: sortStudents(); break;
            case 5: displayStudents(); break;
            case 6: maxStudent(); break;
            case 7: printf("Exiting program...\n"); break;
            default: printf("Invalid choice!\n");
        }
    } while (choice != 7);

    return 0;
}

// Add student
void addStudent() {
    if (count >= MAX) {
        printf("Database full!\n");
        return;
    }
    printf("Enter Roll No: ");
    scanf("%d", &students[count].rollNo);
    printf("Enter Name: ");
    scanf(" %[^\n]", students[count].name);
    printf("Enter Marks: ");
    scanf("%f", &students[count].marks);
    count++;
    printf("Added successfully!\n");
}

// Update record
void updateStudent() {
    int roll, found = 0;
    printf("Enter Roll No to update: ");
    scanf("%d", &roll);
    for (int i = 0; i < count; i++) {
        if (students[i].rollNo == roll) {
            printf("Enter new Name: ");
            scanf(" %[^\n]", students[i].name);
            printf("Enter new Marks: ");
            scanf("%f", &students[i].marks);
            printf("Updated!\n");
```

```
            found = 1;
            break;
        }
    }
    if (!found) printf("Record not found!\n");
}

// Search student
void searchStudent() {
    int roll, found = 0;
    printf("Enter Roll No to search: ");
    scanf("%d", &roll);
    for (int i = 0; i < count; i++) {
        if (students[i].rollNo == roll) {
            printf("Found: Roll No: %d, Name: %s, Marks: %.2f\n",
                   students[i].rollNo, students[i].name, students[i].marks);
            found = 1;
            break;
        }
    }
    if (!found) printf("Record not found!\n");
}

// Sort by marks (descending)
void sortStudents() {
    struct Student temp;
    for (int i = 0; i < count - 1; i++)
        for (int j = i + 1; j < count; j++)
            if (students[i].marks < students[j].marks) {
                temp = students[i];
                students[i] = students[j];
                students[j] = temp;
            }
    printf("Sorted by marks!\n");
}

// Display all
void displayStudents() {
    if (count == 0) {
        printf("No records found!\n");
        return;
    }
    printf("\n--- Student Records ---\n");
    for (int i = 0; i < count; i++)
        printf("Roll No: %d, Name: %s, Marks: %.2f\n",
               students[i].rollNo, students[i].name, students[i].marks);
}

// Find student with max marks
void maxStudent() {
    if (count == 0) {
        printf("No records found!\n");
        return;
    }
    int maxIndex = 0;
    for (int i = 1; i < count; i++)
        if (students[i].marks > students[maxIndex].marks)
            maxIndex = i;

    printf("\nTopper:\nRoll No: %d\nName: %s\nMarks: %.2f\n",
           students[maxIndex].rollNo, students[maxIndex].name, students[maxIndex].marks); }
```

STACK AND QUEUE USING STATIC ARRAYS

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

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

void push() {
   int x;
   if (top == MAX - 1) printf("Stack Full!\n");
   else {
      printf("Enter parcel ID: ");
      scanf("%d", &x);
      stack[++top] = x;
      printf("Added!\n");
   }
}

void pop() {
   if (top == -1) printf("Stack Empty!\n");
   else printf("Removed parcel %d\n", stack[top--]);
}

void displayStack() {
   if (top == -1) printf("No parcels!\n");
   else {
      printf("Stack: ");
      for (int i = top; i >= 0; i--) printf("%d ", stack[i]);
      printf("\n");
   }
}
void add() {
   int x;
   if (rear == MAX - 1) printf("Queue Full!\n");
   else {
      printf("Enter parcel ID: ");
      scanf("%d", &x);
      if (front == -1) front = 0;
      queue[++rear] = x;
      printf("Added!\n");
   }
}
void del() {
   if (front == -1 || front > rear) printf("Queue Empty!\n");
   else printf("Delivered parcel %d\n", queue[front++]);
}
void displayQueue() {
   if (front == -1 || front > rear) printf("No parcels!\n");
   else {
      printf("Queue: ");
      for (int i = front; i <= rear; i++) printf("%d ", queue[i]);
      printf("\n");
   }
}

int main() {
   int type, ch;
   printf("1.Stack(LIFO)  2.Queue(FIFO): ");
   scanf("%d", &type);
   do {
      if (type == 1)
         printf("\n1.Push 2.Pop 3.Display 4.Exit: ");
      else
         printf("\n1.Add 2.Delete 3.Display 4.Exit: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1: if (type==1) push(); else add(); break;
         case 2: if (type==1) pop(); else del(); break;
         case 3: if (type==1) displayStack(); else displayQueue(); break;
      }
   } while (ch != 4);
   return 0;
}
```

TEXT EDITOR USING SINGLY LINKED LIST

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

struct Node {
    char line[100];
    struct Node *next;
};

struct Node *head = NULL;

// Insert at front
void insertFront() {
    struct Node *newNode = malloc(sizeof(struct Node));
    printf("Enter line: ");
    scanf(" %[^\n]", newNode->line);
    newNode->next = head;
    head = newNode;
    printf("Inserted at front!\n");
}

// Insert at end
void insertEnd() {
    struct Node *newNode = malloc(sizeof(struct Node));
    printf("Enter line: ");
    scanf(" %[^\n]", newNode->line);
    newNode->next = NULL;
    if (head == NULL) head = newNode;
    else {
        struct Node *temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newNode;
    }
    printf("Inserted at end!\n");
}

// Delete from front
void deleteFront() {
    if (head == NULL) printf("List empty!\n");
    else {
        struct Node *temp = head;
        head = head->next;
        free(temp);
        printf("Deleted front line!\n");
    }
}

// Delete from end
void deleteEnd() {
    if (head == NULL) printf("List empty!\n");
    else if (head->next == NULL) {
        free(head);
        head = NULL;
    } else {
        struct Node *temp = head;
        while (temp->next->next) temp = temp->next;
        free(temp->next);
        temp->next = NULL;
    }
    printf("Deleted end line!\n");
}

// Display all lines
void display() {
    if (head == NULL) printf("No text!\n");
    else {
        struct Node *temp = head;
        printf("\n--- Text Content ---\n");
        while (temp) {
            printf("%s\n", temp->line);
            temp = temp->next;
        }
    }
```

```
}

// Display in reverse (recursion)
void displayReverse(struct Node *node) {
    if (node == NULL) return;
    displayReverse(node->next);
    printf("%s\n", node->line);
}

// Reverse the list (links)
void reverseList() {
    struct Node *prev = NULL, *curr = head, *next = NULL;
    while (curr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
    printf("List reversed!\n");
}

int main() {
    int ch;
    do {
        printf("\n1.Insert Front\n2.Insert End\n3.Delete Front\n4.Delete End\n5.Display\n6.Display Reverse\n7.Reverse List\n8.Exit\nChoice:
");
        scanf("%d", &ch);
        switch (ch) {
            case 1: insertFront(); break;
            case 2: insertEnd(); break;
            case 3: deleteFront(); break;
            case 4: deleteEnd(); break;
            case 5: display(); break;
            case 6: displayReverse(head); break;
            case 7: reverseList(); break;
        }
    } while (ch != 8);
    return 0;
}
```

ONLINE DIRECTORY SYSTEM USING BST

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

struct Node {
    char name[50];
    struct Node *left, *right;
};

struct Node* createNode(char name[]) {
    struct Node* newNode = malloc(sizeof(struct Node));
    strcpy(newNode->name, name);
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, char name[]) {
    if (root == NULL) return createNode(name);
    if (strcmp(name, root->name) < 0)
        root->left = insert(root->left, name);
    else if (strcmp(name, root->name) > 0)
        root->right = insert(root->right, name);
    return root;
}

void inorder(struct Node* root) {
    if (root) {
        inorder(root->left);
        printf("%s ", root->name);
        inorder(root->right);
    }
}

void preorder(struct Node* root) {
    if (root) {
        printf("%s ", root->name);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        printf("%s ", root->name);
    }
}

void search(struct Node* root, char key[]) {
    if (root == NULL)
        printf("Name not found!\n");
    else if (strcmp(key, root->name) == 0)
        printf("Name found: %s\n", root->name);
    else if (strcmp(key, root->name) < 0)
        search(root->left, key);
    else
        search(root->right, key);
}

int main() {
    struct Node* root = NULL;
    int ch;
    char name[50];

    do {
        printf("\n1. Insert Name\n2. Inorder\n3. Preorder\n4. Postorder\n5. Search\n6. Exit\nChoice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter name: ");
                scanf(" %[^\n]", name);
                root = insert(root, name);
                break;
```

```
            case 2:
                printf("Inorder: ");
                inorder(root);
                printf("\n");
                break;
            case 3:
                printf("Preorder: ");
                preorder(root);
                printf("\n");
                break;
            case 4:
                printf("Postorder: ");
                postorder(root);
                printf("\n");
                break;
            case 5:
                printf("Enter name to search: ");
                scanf(" %[^\n]", name);
                search(root, name);
                break;
        }
    } while (ch != 6);
    return 0;
}
```

CAMPUS NAVIGATION USING BFS AND DFS

```
#include <stdio.h>
#define MAX 10
int adj[MAX][MAX], visited[MAX], n;

// DFS Function
void DFS(int v) {
   printf("%d ", v);
   visited[v] = 1;
   for (int i = 0; i < n; i++) {
      if (adj[v][i] == 1 && !visited[i])
         DFS(i);
   }
}
// BFS Function
void BFS(int start) {
   int queue[MAX], front = 0, rear = 0;
   for (int i = 0; i < n; i++) visited[i] = 0;

   printf("%d ", start);
   visited[start] = 1;
   queue[rear++] = start;

   while (front < rear) {
      int v = queue[front++];
      for (int i = 0; i < n; i++) {
         if (adj[v][i] == 1 && !visited[i]) {
            printf("%d ", i);
            visited[i] = 1;
            queue[rear++] = i;
         }
      }
   }
}

int main() {
   int i, j, start, choice;

   printf("Enter number of buildings (nodes): ");
   scanf("%d", &n);

   printf("Enter adjacency matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         scanf("%d", &adj[i][j]);

   printf("Enter starting building: ");
   scanf("%d", &start);

   do {
      printf("\n1. DFS\n2. BFS\n3. Exit\nEnter choice: ");
      scanf("%d", &choice);

      for (i = 0; i < n; i++) visited[i] = 0;

      switch (choice) {
         case 1:
            printf("DFS Traversal: ");
            DFS(start);
            printf("\n");
            break;
         case 2:
            printf("BFS Traversal: ");
            BFS(start);
            printf("\n");
            break;
         case 3:
            printf("Exiting...\n");
            break;
         default:
            printf("Invalid choice!\n");
      }
   } while (choice != 3);

   return 0; }
```

Pattern Printing using Loops

```
#include <stdio.h>

int main() {
    int n, i, j, space;

    printf("Enter number of rows: ");
    scanf("%d", &n);

    // 1. Right-angled triangle with numbers
    printf("\nRight-angled Triangle:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= i; j++)
            printf("%d ", j);
        printf("\n");
    }

    // 2. Diamond shape with numbers
    printf("\nDiamond Shape:\n");
    for (i = 1; i <= n; i++) {
        for (space = i; space < n; space++) printf(" ");
        for (j = 1; j <= (2*i - 1); j++) printf("%d", j);
        printf("\n");
    }
    for (i = n-1; i >= 1; i--) {
        for (space = n; space > i; space--) printf(" ");
        for (j = 1; j <= (2*i - 1); j++) printf("%d", j);
        printf("\n");
    }

    // 3. Pyramid with asterisks
    printf("\nPyramid with Asterisks:\n");
    for (i = 1; i <= n; i++) {
        for (space = i; space < n; space++) printf(" ");
        for (j = 1; j <= (2*i - 1); j++) printf("*");
        printf("\n");
    }

    // 4. Pyramid with alphabets
    printf("\nPyramid with Alphabets:\n");
    for (i = 1; i <= n; i++) {
        char ch = 'A';
        for (space = i; space < n; space++) printf(" ");
        for (j = 1; j <= (2*i - 1); j++) {
            printf("%c", ch);
            if (j < i) ch++;
            else ch--;
        }
        printf("\n");
    }

    return 0;
}
```

Sequential and Binary Search in Contact Manager

```
#include <stdio.h>
#include <string.h>
#define MAX 10

int sequentialSearch(char names[][50], int n, char key[]) {
    for (int i = 0; i < n; i++)
        if (strcmp(names[i], key) == 0)
            return i;
    return -1;
}

int binarySearch(char names[][50], int n, char key[]) {
    int low = 0, high = n - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        int cmp = strcmp(key, names[mid]);
        if (cmp == 0) return mid;
        else if (cmp < 0) high = mid - 1;
        else low = mid + 1;
    }
    return -1;
}

int main() {
    char names[MAX][50], key[50];
    int n, i, choice, pos;

    printf("Enter number of contacts: ");
    scanf("%d", &n);

    printf("Enter contact names (in sorted order for binary search):\n");
    for (i = 0; i < n; i++)
        scanf("%s", names[i]);

    printf("Enter name to search: ");
    scanf("%s", key);

    printf("\n1. Sequential Search\n2. Binary Search\nEnter choice: ");
    scanf("%d", &choice);

    if (choice == 1)
        pos = sequentialSearch(names, n, key);
    else
        pos = binarySearch(names, n, key);

    if (pos != -1)
        printf("\nContact found at position %d\n", pos + 1);
    else
        printf("\nContact not found!\n");

    return 0;
}
```

Sorting Product Prices (Bubble, Selection, Insertion Sort)

```
#include <stdio.h>

#define MAX 5

void bubbleSort(int a[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (a[j] > a[j + 1]) {
                int temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
}

void selectionSort(int a[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++)
            if (a[j] < a[min])
                min = j;
        int temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
}

void insertionSort(int a[], int n) {
    for (int i = 1; i < n; i++) {
        int key = a[i], j = i - 1;
        while (j >= 0 && a[j] > key) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }
}

void display(int a[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

int main() {
    int a[MAX], n, choice;

    printf("Enter number of products (max %d): ", MAX);
    scanf("%d", &n);

    printf("Enter product prices:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);

    do {
        printf("\n1. Bubble Sort\n2. Selection Sort\n3. Insertion Sort\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        int temp[MAX];
        for (int i = 0; i < n; i++) temp[i] = a[i]; // copy array

        switch (choice) {
            case 1:
                bubbleSort(temp, n);
                printf("Sorted Prices (Bubble Sort): ");
                display(temp, n);
                break;
            case 2:
                selectionSort(temp, n);
                printf("Sorted Prices (Selection Sort): ");
                display(temp, n);
                break;
            case 3:
                insertionSort(temp, n);
                printf("Sorted Prices (Insertion Sort): ");
```

```
            display(temp, n);
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
    }
  } while (choice != 4);

  return 0;
}
```

Stack & Queue using Linked List (Dynamic Implementation)

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

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

struct Node *top = NULL, *front = NULL, *rear = NULL;

void push(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = top;
    top = newNode;
}

void pop() {
    if (top == NULL) printf("Stack is empty!\n");
    else {
        printf("Popped: %d\n", top->data);
        struct Node* temp = top;
        top = top->next;
        free(temp);
    }
}

void displayStack() {
    struct Node* temp = top;
    printf("Stack: ");
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void enqueue(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = NULL;
    if (rear == NULL)
        front = rear = newNode;
    else {
        rear->next = newNode;
        rear = newNode;
    }
}

void dequeue() {
    if (front == NULL) printf("Queue is empty!\n");
    else {
        printf("Deleted: %d\n", front->data);
        struct Node* temp = front;
        front = front->next;
        if (front == NULL) rear = NULL;
        free(temp);
    }
}

void displayQueue() {
    struct Node* temp = front;
    printf("Queue: ");
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, val, mode;
    printf("Choose system:\n1. Stack (LIFO - Browser History)\n2. Queue (FIFO - Service Window)\nEnter choice: ");
    scanf("%d", &mode);
```

```
    do {
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &val);
                if (mode == 1) push(val);
                else enqueue(val);
                break;
            case 2:
                if (mode == 1) pop();
                else dequeue();
                break;
            case 3:
                if (mode == 1) displayStack();
                else displayQueue();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

BINARY TO DECIMAL

```
#include <stdio.h>
#define MAX 20

int stack[MAX];
int top = -1;

void push(int val) {
   stack[++top] = val;
}

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

int main() {
   int num, rem;
   printf("Enter a decimal number: ");
   scanf("%d", &num);

   while (num > 0) {
      rem = num % 2;
      push(rem);
      num = num / 2;
   }

   printf("Binary = ");
   while (top != -1)
      printf("%d", pop());

   return 0;
}
```