Major Paper- 5

DMS LAB MANUAL

Write the following programs using C/ C++

1. **To find the power set for a given set.**

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

void printPowerSet(char* set, int set_size) {
    int pow_set_size = pow(2, set_size);
    for (int counter = 0; counter < pow_set_size; counter++) {
        printf("{ ");
        for (int j = 0; j < set_size; j++) {
            if (counter & (1 << j))
                printf("%c ", set[j]);
        }
        printf("}\n");
    }
}

int main() {
    char set[] = {'a', 'b', 'c'};
    printPowerSet(set, 3);
    return 0;
}
```

2. Take two sets as input and display their **Union, Intersection, and Difference**.

```
#include <stdio.h>

void Union(int a[], int b[], int n, int m) {
    int i, j, k = 0, c[100];
    for (i = 0; i < n; i++) c[k++] = a[i];
    for (i = 0; i < m; i++) {
        int found = 0;
        for (j = 0; j < n; j++)
            if (b[i] == a[j]) found = 1;
        if (!found) c[k++] = b[i];
    }
    printf("Union: ");
    for (i = 0; i < k; i++) printf("%d ", c[i]);
    printf("\n");
}
```

```
void Intersection(int a[], int b[], int n, int m) {
    int i, j;
    printf("Intersection: ");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            if (a[i] == b[j])
                printf("%d ", a[i]);
    printf("\n");
}

void Difference(int a[], int b[], int n, int m) {
    int i, j, found;
    printf("A - B: ");
    for (i = 0; i < n; i++) {
        found = 0;
        for (j = 0; j < m; j++)
            if (a[i] == b[j]) found = 1;
        if (!found) printf("%d ", a[i]);
    }
    printf("\n");
}

int main() {
    int a[] = {1,2,3,4};
    int b[] = {3,4,5,6};
    int n = 4, m = 4;
    Union(a,b,n,m);
    Intersection(a,b,n,m);
    Difference(a,b,n,m);
    return 0;
}
```

3. Read three sets and output the elements that belong to **exactly one**, **exactly two**, or **all three** sets.

```
#include <stdio.h>

int main() {
    int A[] = {1,2,3,4};
    int B[] = {3,4,5,6};
    int C[] = {4,5,6,7};
    int U[100], k = 0, i, j, found;
```

```
// Build union
for (i = 0; i < 4; i++) U[k++] = A[i];
for (i = 0; i < 4; i++) {
    found = 0;
    for (j = 0; j < k; j++)
        if (B[i] == U[j]) found = 1;
    if (!found) U[k++] = B[i];
}
for (i = 0; i < 4; i++) {
    found = 0;
    for (j = 0; j < k; j++)
        if (C[i] == U[j]) found = 1;
    if (!found) U[k++] = C[i];
}

printf("Exactly 1: ");
for (i = 0; i < k; i++) {
    int count = 0;
    for (j = 0; j < 4; j++) if (U[i] == A[j]) count++;
    for (j = 0; j < 4; j++) if (U[i] == B[j]) count++;
    for (j = 0; j < 4; j++) if (U[i] == C[j]) count++;
    if (count == 1) printf("%d ", U[i]);
}

printf("\nExactly 2: ");
for (i = 0; i < k; i++) {
    int count = 0;
    for (j = 0; j < 4; j++) if (U[i] == A[j]) count++;
    for (j = 0; j < 4; j++) if (U[i] == B[j]) count++;
    for (j = 0; j < 4; j++) if (U[i] == C[j]) count++;
    if (count == 2) printf("%d ", U[i]);
}

printf("\nAll 3: ");
for (i = 0; i < k; i++) {
    int count = 0;
    for (j = 0; j < 4; j++) if (U[i] == A[j]) count++;
    for (j = 0; j < 4; j++) if (U[i] == B[j]) count++;
    for (j = 0; j < 4; j++) if (U[i] == C[j]) count++;
    if (count == 3) printf("%d ", U[i]);
}
printf("\n");
return 0;
```

```
}
```

4. Compute the **Cartesian Product** of two sets.

```
#include <stdio.h>

int main() {
    int A[] = {1, 2};
    int B[] = {3, 4, 5};
    int n = 2, m = 3;

    printf("Cartesian Product A x B:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            printf("(%d, %d) ", A[i], B[j]);
        }
        printf("\n");
    }
    return 0;
}
```

5. Check whether a given function (input as pairs) is one-to-one, onto, or both.
   Here, we assume the function is given as pairs (x, f(x)).
   We check:

- One-to-one (injective): no two *x* map to the same *f(x)*
- Onto (surjective): *f(x)* covers the whole target set.

```
#include <stdio.h>

int isOneToOne(int x[], int fx[], int n) {
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (fx[i] == fx[j] && x[i] != x[j])
                return 0;
    return 1;
}

int isOnto(int fx[], int codomain[], int n, int m) {
    for (int i = 0; i < m; i++) {
        int found = 0;
        for (int j = 0; j < n; j++)
            if (codomain[i] == fx[j]) found = 1;
        if (!found) return 0;
    }
    return 1;
}

int main() {
    int x[] = {1, 2, 3};
    int fx[] = {4, 5, 6};
    int codomain[] = {4, 5, 6};

    int n = 3, m = 3;

    if (isOneToOne(x, fx, n))
        printf("The function is One-to-One.\n");
    else
        printf("Not One-to-One.\n");

    if (isOnto(fx, codomain, n, m))
        printf("The function is Onto.\n");
    else
        printf("Not Onto.\n");

    return 0;
}
```

6. Find the **composition** of two functions entered as arrays.

```
#include <stdio.h>

int main() {
    // f: X -> Y
    int x[] = {1, 2, 3};
    int fx[] = {2, 3, 4};

    // g: Y -> Z
    int y[] = {2, 3, 4};
    int gy[] = {5, 6, 7};

    int n = 3;

    printf("Composition g(f(x)):\n");
    for (int i = 0; i < n; i++) {
        int f = fx[i];
        int g;
        for (int j = 0; j < n; j++) {
            if (y[j] == f) {
                g = gy[j];
                break;
            }
        }
        printf("x: %d, f(x): %d, g(f(x)): %d\n", x[i], f, g);
    }
    return 0;
}
```

7. Implement an **inverse function** (for a bijection).

```
#include <stdio.h>

int main() {
    int x[] = {1, 2, 3};
    int fx[] = {4, 5, 6};
    int n = 3;

    printf("Inverse Function f⁻¹(y):\n");
    for (int i = 0; i < n; i++) {
        printf("f(%d) = %d => f⁻¹(%d) = %d\n", x[i], fx[i], fx[i], x[i]);
    }
    return 0;
}
```

8. Calculate a **summation** (e.g., sum of first *n* squares).

```
#include <stdio.h>

int main() {
    int n = 5;
    int sum = 0;

    for (int i = 1; i <= n; i++)
        sum += i * i;

    printf("Sum of squares of first %d natural numbers: %d\n", n, sum);
    return 0;
}
```

9. **To find Permutation and Combination result for a given pair of values n and r.**

```
#include <stdio.h>

int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial(n - 1);
}

int main() {
    int n = 5, r = 2;
    int perm = factorial(n) / factorial(n - r);
    int comb = factorial(n) / (factorial(r) * factorial(n - r));

    printf("Permutation: %d\n", perm);
    printf("Combination: %d\n", comb);
    return 0;
}
```

10. To find Binomial coefficients

```
#include <stdio.h>

int binomialCoeff(int n, int k) {
    if (k == 0 || k == n)
        return 1;
    return binomialCoeff(n - 1, k - 1) + binomialCoeff(n - 1, k);
}

int main() {
    int n = 5, k = 2;
    printf("C(%d,%d) = %d\n", n, k, binomialCoeff(n, k));
    return 0;
}
```

11. To check a number is prime or not.

```
#include <stdio.h>

int isPrime(int n) {
    if (n <= 1) return 0;
    for (int i = 2; i <= n / 2; i++)
        if (n % i == 0)
            return 0;
    return 1;
}

int main() {
    int num = 29;
    if (isPrime(num))
        printf("%d is Prime\n", num);
    else
```

```
        printf("%d is Not Prime\n", num);
    return 0;
}
```

12. Generate the first *n* terms of a sequence defined by a **recurrence relation**, e.g., Fibonacci or custom user-defined relation.

```
#include <stdio.h>

int fib(int n) {
    if (n <= 1) return n;
    else return fib(n-1) + fib(n-2);
}

int main() {
    int n = 10;
    printf("Fibonacci Sequence up to %d terms:\n", n);
    for (int i = 0; i < n; i++) {
        printf("%d ", fib(i));
    }
    printf("\n");
    return 0;
}
```

13. Graph representation using Adjacency List.

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

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

struct List {
    struct Node* head;
};

struct Graph {
    int V;
    struct List* array;
};

struct Node* newNode(int dest) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->dest = dest;
    node->next = NULL;
    return node;
}

struct Graph* createGraph(int V) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->V = V;
    graph->array = (struct List*)malloc(V * sizeof(struct List));
    for (int i = 0; i < V; ++i)
        graph->array[i].head = NULL;
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* node = newNode(dest);
    node->next = graph->array[src].head;
    graph->array[src].head = node;

    node = newNode(src);
    node->next = graph->array[dest].head;
    graph->array[dest].head = node;
}

void printGraph(struct Graph* graph) {
    for (int v = 0; v < graph->V; ++v) {
        struct Node* pCrawl = graph->array[v].head;
        printf("\n Adjacency list of vertex %d\n head ", v);
        while (pCrawl) {
            printf("-> %d", pCrawl->dest);
            pCrawl = pCrawl->next;
        }
        printf("\n");
    }
}

int main() {
    int V = 5;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);
    printGraph(graph);
    return 0;
}
```

14. Graph representation using Adjacency Matrix.

```
#include <stdio.h>

#define V 5

void printMatrix(int graph[V][V]) {
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int graph[V][V] = {0};

    graph[0][1] = 1;
    graph[1][0] = 1;
    graph[0][4] = 1;
    graph[4][0] = 1;
    graph[1][2] = 1;
    graph[2][1] = 1;
    graph[1][3] = 1;
    graph[3][1] = 1;
    graph[1][4] = 1;
    graph[4][1] = 1;
    graph[2][3] = 1;
    graph[3][2] = 1;
    graph[3][4] = 1;
    graph[4][3] = 1;

    printMatrix(graph);
    return 0;
}
```

15. **Find the shortest path pair in a plane.**

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

typedef struct {
    double x, y;
} Point;

double distance(Point p1, Point p2) {
    return sqrt((p2.x - p1.x)*(p2.x - p1.x) + (p2.y - p1.y)*(p2.y - p1.y));
}

int main() {
    Point p1 = {1.0, 2.0}, p2 = {4.0, 6.0};
    printf("Shortest path distance: %.2f\n", distance(p1, p2));
    return 0;
}
```