Codekata Report:

Name: Deepu Pandey

Email: deepu.24scse1011405@galgotiasuniversity.ac.in

Specialization: School of Computer Science & Engineering

Completion Year: 2028-3rd Sem

Section: Section-7

1. Title: Maximum Difference in Array

Problem Statement:You are given an array of integers, and you need to find the maximum difference between any two elements in the array. The difference is defined as the absolute difference between the two elements. The array may contain both positive and negative numbers, and your task is to ensure the difference calculation considers all possible pairs within the array.

Input:

The first line contains an integer N (1 ≤ N ≤ 1000) representing the number of elements in the array.The second line contains N space-separated integers representing the elements of the array A[i] (-10^5 ≤ A[i] ≤ 10^5).

Output:

A single integer, the maximum difference between any two elements in the array.

Constraints:

1 ≤ N ≤ 1000-10^5 ≤ A[i] ≤ 10^5

Example 1:Input:51 -3 2 10 6

Output:13

Explanation:The maximum difference is between -3 and 10, which is |10 - (-3)| = 13.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h> // for abs()
int main() {
int n;
scanf("%d", &n);
int arr[n];
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
int max = arr[0];
int min = arr[0];
// Find maximum and minimum element
for (int i = 1; i < n; i++) {
if (arr[i] > max)
max = arr[i];
if (arr[i] < min)
min = arr[i];
}
int maxDiff = abs(max - min);
printf("%d\n", maxDiff);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

20

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

25

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0

Compilation Status: Passed

Execution Time:

0.001s

2. Problem Statement: Warehouse Inventory OptimizationYou are the manager of a large warehouse that stores different types of products. The warehouse is organized in a 3D grid (length, width,

height), where each position in the grid represents a storage bin. Each bin will contain a specific number of the products. Your goal to find the maximum number of products stored within a rectangular block of bins in the grid, defined by any two opposite corners.

Given a 3D grid of integers representing the number of products in each bin, write a C program to calculate the maximum number of products stored within any rectangular sub-block of bins.

Input:The first line will contain three integers named L, W, and H which are representing the length, width, and height of the warehouse grid.The next L * W * H lines contain integers representing the number of products stored in each bin.

Output:A single integer representing the maximum number of products stored within any sub-block of bins.

Constraints:2 <= L, W, H <= 50 (Warehouse dimensions are between 2x2x2 and 50x50x50).0 <= products[i][j][k] <= 1000 (Number of products in each bin is between 0 and 1000).

Explanation:To solve the problem, we need to efficiently compute the sum of products within any sub-block of bins. This can be done by utilizing a 3D prefix sum array to quickly compute the sum of any sub-block using inclusion-exclusion principles.

The 3D prefix sum array prefix[i][j][k] stores the sum of all products in the sub-block from the origin (0,0,0) to the current position (i,j,k). Once the prefix sums are calculated, finding the sum of products for any arbitrary sub-block defined by two opposite corners is fast.

Example:Input:3 3 31 2 34 5 67 8 910 11 1213 14 1516 17 1819 20 2122 23 2425 26 27

Output:351

Explanation: The largest sub-block (the entire grid) has a sum of 351, which is the sum of all bins in the grid.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <limits.h>
int main() {
int L, W, H;
scanf("%d %d %d", &L, &W, &H);
int grid[51][51][51] = {0};
```

```
int prefix[51][51][51] = {0};
// Reading input
for (int i = 0; i < L; i++) {
for (int j = 0; j < W; j++) {
for (int k = 0; k < H; k++) {
scanf("%d", &grid[i][j][k]);
}
}
}
// Building 3D prefix sum
for (int i = 0; i < L; i++) {
for (int j = 0; j < W; j++) {
for (int k = 0; k < H; k++) {
prefix[i][j][k] = grid[i][j][k];
if (i > 0) prefix[i][j][k] += prefix[i-1][j][k];
if (j > 0) prefix[i][j][k] += prefix[i][j-1][k];
if (k > 0) prefix[i][j][k] += prefix[i][j][k-1];
if (i > 0 && j > 0) prefix[i][j][k] -= prefix[i-1][j-1][k];
if (i > 0 && k > 0) prefix[i][j][k] -= prefix[i-1][j][k-1];
if (j > 0 && k > 0) prefix[i][j][k] -= prefix[i][j-1][k-1];
if (i > 0 && j > 0 && k > 0) prefix[i][j][k] += prefix[i-1][j-1][k-1];
}
}
}
int maxSum = INT_MIN;
// Iterate over all possible sub-blocks
for (int x1 = 0; x1 < L; x1++) {
for (int y1 = 0; y1 < W; y1++) {
for (int z1 = 0; z1 < H; z1++) {
for (int x2 = x1; x2 < L; x2++) {
for (int y2 = y1; y2 < W; y2++) {
for (int z2 = z1; z2 < H; z2++) {
int sum = prefix[x2][y2][z2];
if (x1 > 0) sum -= prefix[x1-1][y2][z2];
if (y1 > 0) sum -= prefix[x2][y1-1][z2];
if (z1 > 0) sum -= prefix[x2][y2][z1-1];
if (x1 > 0 && y1 > 0) sum += prefix[x1-1][y1-1][z2];
if (x1 > 0 && z1 > 0) sum += prefix[x1-1][y2][z1-1];
if (y1 > 0 && z1 > 0) sum += prefix[x2][y1-1][z1-1];
if (x1 > 0 && y1 > 0 && z1 > 0) sum -= prefix[x1-1][y1-1][z1-1];
if (sum > maxSum) maxSum = sum;
}
}
}
}
}
}
printf("%d\n",maxSum);
return
0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

378

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

360

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

82

Compilation Status: Passed

Execution Time:

0.001s

3. Problem Statement: Warehouse Inventory ManagementA warehouse is organized into a grid of rows and columns where each cell represents a storage unit for products. The goal is to calculate the total number of products in the warehouse after a series of stock updates. Each stock update specifies a rectangular region within the grid and the number of products to be added to each cell in that region.

You need to write a program to manage these updates and calculate the total number of products in the entire warehouse after all updates.

Input:The first line contains two integers R and C (1 ≤ R, C ≤ 100), representing the number of rows and columns of the warehouse grid.The next line contains an integer U (1 ≤ U ≤ 1000), representing the number of stock updates.Each of the following U lines contains five integers r1, c1, r2, c2, and value (1 ≤ r1 ≤ r2 ≤ R, 1 ≤ c1 ≤ c2 ≤ C, 0 ≤ value ≤ 100), representing a stock update where r1, c1 is the top-left corner and r2, c2 is the bottom-right corner of the rectangular region, and value is the number of products to be added to each cell in that region.

Output:Print a single integer: the total number of products in the warehouse after applying all updates.

ExampleInput:4 521 1 2 2 52 3 4 5 10

Output:75

Explanation:Initially, the grid is all zeros.

After the first update (1 1 2 2 5), the grid looks like:5 5 0 0 05 5 0 0 00 0 0 0 00 0 0 0 0After the second update (2 3 4 5 10), the grid looks like:5 5 0 0 05 5 10 10 100 0 10 10 100 0 10 10 10

The total number of products is calculated as: 5*5 + 5*5 + 10*3*2 = 75.

Constraints:The number of rows and columns are between 1 and 100.The number of updates is between 1 and 1000.The update values are between 0 and 100.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
int main() {
```

```
int R, C, U;
scanf("%d %d", &R, &C);
scanf("%d", &U);
int grid[102][102] = {0}; // Padding to simplify boundary updates
// Apply updates using 2D difference array
for (int i = 0; i < U; i++) {
int r1, c1, r2, c2, value;
scanf("%d %d %d %d %d", &r1, &c1, &r2, &c2, &value);
grid[r1][c1] += value;
grid[r1][c2 + 1] -= value;
grid[r2 + 1][c1] -= value;
grid[r2 + 1][c2 + 1] += value;
}
// Convert to actual grid using prefix sums
for (int i = 1; i <= R; i++) {
for (int j = 1; j <= C; j++) {
grid[i][j] += grid[i - 1][j] + grid[i][j - 1] - grid[i - 1][j - 1];
}
}
// Compute total products
int total = 0;
for (int i = 1; i <= R; i++) {
for (int j = 1; j <= C; j++) {
total += grid[i][j];
}
}
// Labeled output for GUVI
printf("Total number of products: %d\n", total);
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Total number of products: 63

Compilation Status: Passed

Execution Time:

0.002s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Total number of products: 22

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Total number of products: 50

Compilation Status: Passed

Execution Time:

0.001s

4. Problem Statement: Student Records ManagementObjective:

Write a C program to manage a set of student records using arrays, pointers, structures, and unions. The program should allow storing and displaying information about each student, including their grades and status.

Specifications:

Structures:

Please define a structure named Student with these following members:Name (Its an array of characters with a maximum length = 50)id (an integer)grades (an array of five floating-point numbers)average (a floating-point number)status (a union that can hold either a character to represent pass/fail or an integer for a grade category)Union:

The union status can either hold:char passFail ('P' for pass, 'F' for fail) if the average is greater than or equal to 50.int gradeCategory (1 for excellent, 2 for good, 3 for average, 4 for poor) based on the average score:Excellent (>= 85)Good (>= 70 and <

85)Average (>= 50 and < 70)Poor (< 50)Functions:

Implement the following functions:void inputStudentData(Student *s, int n) - to inpu data for n students.void calculateAverage(Student *s, int n) - to calculate the averag grades for each student.void determineStatus(Student *s, int n) - to set the status c each student based on their average.void displayStudents(const Student *s, int n) - to display the information of each student.Input/Output:

Your code should first read an integer n = the number of students.Then, it should read the name, ID, and the five grades for each student.Calculate the average, determine the status, and display the details of each student.ExampleInput:2Alice 101 75.0 82.0 90.0 70.0 88.0Bob 102 45.0 55.0 65.0 35.0 60.0

Output:Student: AliceID: 101Grades: 75.0, 82.0, 90.0, 70.0, 88.0Average: 81.0Status: Grade Category - 2 (Good)

Student: BobID: 102Grades: 45.0, 55.0, 65.0, 35.0, 60.0Average: 52.0Status: Pass/Fail - P

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
typedef union {
char passFail;
int gradeCategory;
} Status;
typedef struct {
char name[50];
int id;
float grades[5];
float average;
Status status;
int usePassFail;
} Student;
void inputStudentData(Student *s, int n) {
for (int i = 0; i < n; i++) {
scanf("%s", s[i].name);
scanf("%d", &s[i].id);
for (int j = 0; j < 5; j++) {
scanf("%f", &s[i].grades[j]);
}
s[i].usePassFail = 0;
}
```

```
}
void calculateAverage(Student *s, int n) {
for (int i = 0; i < n; i++) {
float sum = 0;
for (int j = 0; j < 5; j++) {
sum += s[i].grades[j];
}
s[i].average = sum / 5.0;
}
}
void determineStatus(Student *s, int n) {
for (int i = 0; i < n; i++) {
if (s[i].average >= 85) {
s[i].status.gradeCategory = 1;
s[i].usePassFail = 0;
} else if (s[i].average >= 70) {
s[i].status.gradeCategory = 2;
s[i].usePassFail = 0;
} else if (s[i].average >= 50) {
s[i].status.passFail = 'P';
s[i].usePassFail = 1;
} else {
s[i].status.passFail = 'F';
s[i].usePassFail = 1;
}
}
}
void displayStudents(const Student *s, int n) {
for (int i = 0; i < n; i++) {
printf("%s\n", s[i].name);
printf("%d\n", s[i].id);
for (int j = 0; j < 5; j++) {
printf("%.1f ", s[i].grades[j]);
}
printf("\n%.1f\n", s[i].average);

if (s[i].usePassFail) {
printf("%c\n", s[i].status.passFail);
} else {
printf("%d\n", s[i].status.gradeCategory);
}
}
}
int main() {
int n;
scanf("%d", &n);
Student students[n];

inputStudentData(students, n);
calculateAverage(students, n);
determineStatus(students, n);
displayStudents(students, n);

return 0;
```

}

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Alice
101
90.0 88.0 92.0 87.0 91.0
89.6
1

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Bob
102
72.0 75.0 70.0 74.0 71.0
72.4
2

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Charlie
103
50.0 55.0 60.0 52.0 58.0
55.0
P

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

David
104
30.0 40.0 35.0 38.0 45.0
37.6
F

Compilation Status: Passed

Execution Time:

0.001s

5. Problem Statement: Survivor GridYou are given a grid representing a battle zone, where each cell has a certain number of soldiers. Some of the cells are designated as "safe zones" (represented by 0 soldiers), which are protected and cannot be attacked. Your task is to simulate a strategic attack where you can choose to attack a group of connected cells in the grid.

Cells are considered connected if they share a side (left, right, up, or down) with another cell containing soldiers. When you attack a connected group of cells, all the soldiers in that group are eliminated, and the total number of soldiers eliminated is

counted.

Your goal is to find the maximum number of soldiers you can eliminate in a single attack on a connected group of cells.

Input Format:The first line contains two integers n and m (where 2 ≤ n, m ≤ 500), representing the number of rows and columns in the grid.The next n lines each contain m integers, representing the number of soldiers in each cell of the grid. A value of 0 in the grid represents a "safe zone" that cannot be attacked or crossed.

Output Format:Output a single integer, the maximum number of soldiers you can eliminate in a single attack.

Constraints:2 ≤ n, m ≤ 500The number of soldiers in any cell will be between 1 and 1000, except for safe zones which have 0 soldiers.The grid contains at least one cell with soldiers (≥ 1).

Example:

Input:5 510 0 0 10 010 10 0 10 00 0 0 10 1010 10 0 0 010 0 10 10 10

Output:50

Explanation:In the given 5x5 grid, the largest connected group of cells with soldiers is in the bottom right corner:0 0 1010 10 10The total number of soldiers in this group is 50.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#define MAX 500
int grid[MAX][MAX];
int visited[MAX][MAX];
int n, m;
// Directions: up, down, left, right
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};
// DFS to explore connected group
int dfs(int x, int y) {
if (x < 0 || x >= n || y < 0 || y >= m || visited[x][y] || grid[x][y] == 0)
return 0;
visited[x][y] = 1;
int total = grid[x][y];
for (int dir = 0; dir < 4; dir++) {
int nx = x + dx[dir];
int ny = y + dy[dir];
```

```
total += dfs(nx, ny);
}
return total;
}
int main() {
scanf("%d %d", &n, &m);
for (int i = 0; i < n; i++)
for (int j = 0; j < m; j++)
scanf("%d", &grid[i][j]);
int maxSoldiers = 0;
for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++) {
if (!visited[i][j] && grid[i][j] != 0) {
int eliminated = dfs(i, j);
if (eliminated > maxSoldiers)
maxSoldiers = eliminated;
}     }
}
printf("%d\n", maxSoldiers);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

70

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

30

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

60

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

120

Compilation Status: Passed

Execution Time:

0.001s

6. Problem Statement: City Grid Traffic OptimizationYou are given a city represented as a 2D grid of intersections, where each element in the grid represents the time (in minutes) it takes for traffic to pass through that intersection. The city mayor wants to optimize traffic flow by finding the fastest route from the top-left corner of the city to the bottom-right corner. You are allowed to move only

right or down at any intersection.

Given a matrix M x N representing the city grid, where each value at grid[i][j] represents the time required at the (i, j) intersection, write a C program that finds th minimum total time required to move from the top-left corner (0,0) to the bottom-rig corner (M-1, N-1).

Input:The first line contains two integers M and N — the number of rows and columns of the grid.The next M lines each contain N integers representing the city grid, where each integer is the time in minutes for that particular intersection.

Output:A single integer, the minimum time required to travel from the top-left to the bottom-right corner.

Constraints:2 <= M, N <= 100 (grid size between 2x2 and 100x100)1 <= grid[i][j] <= 100 (time taken at each intersection is between 1 and 100 minutes)

Explanation:The idea is to use a dynamic programming approach to compute the minimum time at each intersection. From the top-left corner (0, 0), for any intersection (i, j), you can only arrive from either the intersection directly above (i-1, j) or the one directly to the left (i, j-1).

We need to compute the minimum time required to reach each intersection (i, j) using the relation:min_time[i][j] = grid[i][j] + min(min_time[i-1][j], min_time[i][j-1])The value at min_time[M-1][N-1] will give the answer.

Example:Input:3 34 7 86 5 93 2 1

Output:18

Explanation: The best route is to move like this:(0,0) -> (0,1) -> (0,2) -> (1,2) -> (2,2)The total time will be 4 + 7 + 8 + 9 + 1 = 18.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#define MAX 100
int min(int a, int b) {
return (a < b) ? a : b;
}
int main() {
int M, N;
int grid[MAX][MAX], min_time[MAX][MAX];
scanf("%d %d", &M, &N);
// Input grid
for (int i = 0; i < M; i++)
```

```
for (int j = 0; j < N; j++)
scanf("%d", &grid[i][j]);
// Initialize top-left corner
min_time[0][0] = grid[0][0];
// Fill first row
for (int j = 1; j < N; j++)
min_time[0][j] = grid[0][j] + min_time[0][j - 1];
// Fill first column
for (int i = 1; i < M; i++)
min_time[i][0] = grid[i][0] + min_time[i - 1][0];
// Fill rest of the grid
for (int i = 1; i < M; i++) {
for (int j = 1; j < N; j++) {
min_time[i][j] = grid[i][j] + min(min_time[i - 1][j], min_time[i][j - 1]);
}
}
// Output result
printf("%d\n", min_time[M - 1][N - 1]);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

16

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

10

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

70

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5

Compilation Status: Passed

Execution Time:

0.001s

7. Problem StatementTitle: Warehouse Management System Optimization

Problem Description:You are tasked with optimizing a warehouse management system. The warehouse stores items in containers, and each container has a specific weight. The system should dynamically determine the most suitable container for the next operation based on the weights of the containers and the operation type.

Operations include adding more items to a container (increasing weight) or removi items (decreasing weight).

You must use the sizeof, comma, and conditional (?:) operators to achieve the following:

Determine the Container with Minimum Weight:

Use the ?: operator to select the container with the minimum weight.Calculate the Size of the Selected Container:

Use the sizeof operator to calculate the size of the selected container.Sequence the Operations Efficiently:

Use the comma operator to sequence operations, ensuring the selection and size calculation is efficient and correctly ordered.

Input:An integer n (3 ≤ n ≤ 100) representing the number of containers.An array weights of n integers (1 ≤ weights[i] ≤ 1000) representing the weights of each container.An integer operation (0 or 1), where 0 indicates an "add" operation and 1 indicates a "remove" operation.

Output:The weight of the selected container after performing the specified operation.The size of the selected container (in bytes).

Constraints:The weights of containers are always positive.The operation should be performed on the container with the minimum weight.Assume that the sizeof(int) is equal to 4 bytes for this problem.You cannot use loops; all operations must be performed using the sizeof, comma, and conditional operators.

Explanation:Use the conditional operator to identify the container with the minimum weight.Calculate the size of the container using the sizeof operator.Sequence the operations using the comma operator to ensure proper execution order.

Example Input 1:5100 200 50 400 1500Example Output 1:Selected Container Weight: 51Size of Container: 4

Example Input 2:4250 500 120 3001Example Output 2:Selected Container Weight: 119Size of Container: 4

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
int main() {
int n;
int weights[5];
```

```
int operation;
scanf("%d", &n);
for(int i = 0; i < n; i++)
scanf("%d", &weights[i]);
scanf("%d", &operation);
int minIndex = (weights[0] <= weights[1] ? 0 : 1);
minIndex = (weights[minIndex] <= weights[2] ? minIndex : 2);
minIndex = (weights[minIndex] <= weights[3] ? minIndex : 3);
minIndex = (weights[minIndex] <= weights[4] ? minIndex : 4);
int selectedWeight = ((weights[minIndex] = (operation == 0 ? weights[minIndex]+1 :
weights[minIndex]-1)), weights[minIndex]);
int sizeOfContainer = sizeof(weights[minIndex]);
printf("Selected Container Weight: %d\n", selectedWeight);
printf("Size of Container: %d\n", sizeOfContainer);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Selected Container Weight: 51
Size of Container: 4

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Selected Container Weight: 119
Size of Container: 4

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Selected Container Weight: 1
Size of Container: 4

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Selected Container Weight: 54
Size of Container: 4

Compilation Status: Passed

Execution Time:

0.001s

8. Problem StatementYou have to create a program to calculate the factorial of a number using the recursion concept. Factorial is a mathematical operation that multiplies a number by all positive integers less than itself. So, you need to:

Implement a recursive function to compute the factorial of a given non-negative

integer.Handle user input and output the result of the factorial calculation.Function Requiredint factorial(int n): Computes the factorial of the integer n using recursion.

InputA single integer n, where 0 ≤ n ≤ 12.

OutputPrint the factorial of n.

Constraints0 ≤ n ≤ 12

ExampleInput5

Output120

ExplanationFor the input 5, the factorial is computed as 5! = 5 * 4 * 3 * 2 * 1 = 120.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
int factorial(int n) {
if (n == 0) return 1;
return n * factorial(n - 1);
}
int main() {
int n;
scanf("%d", &n);
printf("%d\n", factorial(n));
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

6

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5040

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3628800

Compilation Status: Passed

Execution Time:

0.001s

9. Problem Statement:Write a C program to generate the Fibonacci series using recursion.The Fibonacci sequence is defined as:

F(0)=0, F(1)=1, F(n) = F(n−1) + F(n−2) for n≥2

The program should take an integer n from the user and print the first n terms of the Fibonacci sequence.

Implement the logic using recursion only.

Do not use arrays or strings.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
int fibonacci(int n) {
if (n == 0) return 0;
if (n == 1) return 1;
return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
int n;
scanf("%d", &n);
for (int i = 0; i < n; i++) {
printf("%d", fibonacci(i));
if (i != n - 1) printf(" ");
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0 1 1 2 3

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0 1 1 2 3 5 8

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0 1

Compilation Status: Passed

Execution Time:

0.001s

TestCase5:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0 1 1 2 3 5 8 13 21 34

Compilation Status: Passed

Execution Time:

0.001s

10. Problem Statement: Masked Phone NumbersIn a call center application, employees need to mask the personal information of users while displaying their phone numbers for privacy protection. The goal is to create a program that takes a string representing a list of phone numbers (with spaces separating them) and masks all but the last four digits of each phone number with asterisks ('*').

The numbers may contain non-numeric characters such as spaces, dashes ('-'), or parentheses ('()'), but these should be ignored while processing. The output should retain any formatting, but only the last four digits of each phone number should remain visible, and the rest should be replaced with asterisks.

Input:A string phone_numbers containing multiple phone numbers separated by spaces. The length of the string is n (1 ≤ n ≤ 10^4). Each phone number is between 7

and 15 digits, and can include non-numeric characters for formatting.

Output:A string with each phone number masked except for the last four digits, retaining any formatting.

ExampleInput:+1 (555) 123-4567 123-456-7890 800-555-1212

Output:+1 (***) ***-**67 ***-***-7890 ***-***-1212

Explanation:The first phone number +1 (555) 123-4567 becomes +1 (***) ***-**67.The second phone number 123-456-7890 becomes ***-***-7890.The third phone number 800-555-1212 becomes ***-***-1212.

Constraints:Each phone number contains between 7 and 15 digits.Non-numeric characters (spaces, dashes, parentheses, etc.) should not affect the masking process.The input will only contain valid phone numbers and formatting.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
/**
* @brief Masks a single string token representing a full or partial phone number.
* It masks all digits except for the last four, preserving non-digit characters.
* @param token The string to be masked in-place.
*/
void mask_phone_token(char *token) {
int digit_count = 0;
int len = strlen(token);
// 1. First pass: Count the total number of digits in the token.
for (int i = 0; i < len; i++) {
if (isdigit(token[i])) {
digit_count++;
}
}
// 2. Only apply masking if the token has more than 4 digits.
if (digit_count > 4) {
int digits_to_mask = digit_count - 4;
int digits_seen = 0;
// 3. Second pass: Replace the first 'digits_to_mask' digits with '*'.
for (int i = 0; i < len; i++) {
if (isdigit(token[i])) {
digits_seen++;
```

```
if (digits_seen <= digits_to_mask) {
token[i] = '*';
}
}
}
}
}
int main() {
char input_line[10005];
// Read the entire line of input, which may contain multiple numbers.
if (fgets(input_line, sizeof(input_line), stdin) == NULL) {
return 1; // Exit if there's an error reading input.
}

// Remove the trailing newline character that fgets() might add.
input_line[strcspn(input_line, "\n\r")] = 0;
// Use strtok to split the input string by spaces.
char *token = strtok(input_line, " ");

bool is_first_token = true;
// Process each token until there are no more.
while (token != NULL) {
// Print a space separator before each token except the first one.
if (!is_first_token) {
printf(" ");
}

// Apply the masking logic to the current token.
mask_phone_token(token);

// Print the modified token.
printf("%s", token);

is_first_token = false;

// Get the next token from the string.
token = strtok(NULL, " ");
}

printf("\n"); // Print a final newline for clean output.
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

--7890 ***-***-1212

Compilation Status: Passed

Execution Time:

0.002s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

******3333 ***-***-6666 ****-8888

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

***4321

Compilation Status: Passed

Execution Time:

0.001s

11. Problem Statement:Write a C program to solve the Tower of Hanoi problem for N disks using recursion. Take N from user input.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
if (n == 0)
return;
towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
printf("%d %c %c\n", n, from_rod, to_rod);
towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
int main() {
int n;
scanf("%d", &n);
towerOfHanoi(n, 'A', 'C', 'B');
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 A C

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 A B
2 A C
1 B C

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 A C
2 A B
1 C B
3 A C
1 B A
2 B C
1 A C

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 A B
2 A C
1 B C

3 A B
1 C A
2 C B
1 A B
4 A C
1 B C
2 B A
1 C A
3 B C
1 A B
2 A C
1 B C

Compilation Status: Passed

Execution Time:

0.001s

12. Problem Statement:Write a C program using head recursion to print numbers from 1 to N. Take N as user input.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
void printNumbers(int n) {
if (n == 0)
return;
printNumbers(n - 1);        // head recursion
if (n > 1)                  // print a space before every number except the first
printf(" ");
printf("%d", n);
}
int main() {
int N;
if (scanf("%d", &N) != 1)
return 0;
if (N <= 0)
return 0;
printNumbers(N);
printf("\n");
return 0;
```

}

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 4 5

Compilation Status: Passed

Execution Time:

0.002s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 4 5 6 7

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 4 5 6 7 8 9 10

Compilation Status: Passed

Execution Time:

0.002s

13. Problem Statement:Write a C program to find the factorial of a number using tail recursion. The number should be taken from user input.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
int factorialTail(int n, int acc) {
if(n==0) return acc;
return factorialTail(n-1, n*acc);
}
int main() {
int N;
scanf("%d", &N);
printf("%d\n", factorialTail(N, 1));
```

```
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.002s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

120

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3628800

Compilation Status: Passed

Execution Time:

0.001s

14. Problem StatementTitle: Vectorized Operations on Large Data Sets Using restrict

In a large-scale scientific computation system, you are given two large data arrays that represent measurements from different sensors. Your task is to perform a vectorized operation on these data sets to calculate the sum of corresponding elements from the two arrays and store the result in a third array. The restrict qualifier is crucial here to optimize memory access and prevent any aliasing issues that would otherwise degrade performance.

You need to implement a C program that uses the restrict keyword to ensure that the input arrays and the output array are distinct, allowing the compiler to optimize the vectorized operation for maximum performance.

Input:

The first input is an integer n (1 <= n <= 1000), representing the size of the arrays.The second input consists of n integers representing the first array.The third input consists of n integers representing the second array.

Output:

The program should output the resulting array after performing the element-wise sum of the two arrays.

Constraints:

Use the restrict qualifier to optimize memory access for the two input arrays and the output array.Input will be provided through standard input methods.The size of the arrays will not exceed 1000 elements.The arrays are guaranteed to have no overlapping memory regions.

Example Input/OutputExample 1:

Input:51 2 3 4 55 4 3 2 1Output:6 6 6 6 6

Example 2:

Input:310 20 301 2 3Output:11 22 33

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
void vectorSum(int n, int *restrict a, int *restrict b, int *restrict c) {
for(int i = 0; i < n; i++)
c[i] = a[i] + b[i];
}
int main() {
int n;
scanf("%d", &n);
int a[1000], b[1000], c[1000];
for(int i = 0; i < n; i++) scanf("%d", &a[i]);
for(int i = 0; i < n; i++) scanf("%d", &b[i]);
vectorSum(n, a, b, c);
for(int i = 0; i < n; i++)
printf("%d%c", c[i], i==n-1?'\n':' ');
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

6 6 6 6 6

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

11 22 33

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

500 500 500 500

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2000 4000

Compilation Status: Passed

Execution Time:

0.001s

15. Problem StatementWrite a C program that manages and analyzes a dynamically sized array of floating-point numbers. The program should:

Take(read) the size of the array from the user.Declare and initialize an array of the specified size.Populate(fill) the array with user-input floating-point numbers.Perform these following operations and display the results:Calculate the average of the

numbers.Compute the maximum and minimum numbers present in the array.Compute the standard deviation of the numbers.

Input:An integer n (size of the array, where 1 <= n <= 100).n floating-point numbers.

Output:The average of the numbers with two decimal precision.The maximum number in the array.The minimum number in the array.The standard deviation of the numbers with two decimal precision.

Constraints:The array size n must be between 1 and 100.The input floating-point numbers should be in the range of -1e6 to 1e6.The calculations should be done with precision.ExplanationThis problem emphasizes the understanding of variable declaration and initialization, including handling user input for dynamically sized data. The program should efficiently handle floating-point arithmetic and use appropriate data types and methods to ensure accuracy and precision.

ExampleInput:510.5 20.8 30.1 40.2 50.6

Output:Average: 30.44Maximum: 50.60Minimum: 10.50Standard Deviation: 14.86

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
double my_sqrt(double x) {
if (x <= 0.0) return 0.0;
double r = x;
for (int i = 0; i < 60; ++i) {
double nr = 0.5 * (r + x / r);
if (nr == r) break;
r = nr;
}
return r;
}
int main(void) {
int n;
if (scanf("%d", &n) != 1) return 0;
if (n < 1 || n > 100) return 0;
double *a = (double*)malloc(n * sizeof(double));
if (!a) return 0;
for (int i = 0; i < n; ++i) {
if (scanf("%lf", &a[i]) != 1) {
free(a);
return 0;
}   }
```

```
double sum = 0.0;
double maxv = a[0];
double minv = a[0];
for (int i = 0; i < n; ++i) {
sum += a[i];
if (a[i] > maxv) maxv = a[i];
if (a[i] < minv) minv = a[i];
}
double mean = sum / n;
double sqsum = 0.0;
for (int i = 0; i < n; ++i) {
double diff = a[i] - mean;
sqsum += diff * diff;
}
double variance = sqsum / n;
double stddev = my_sqrt(variance);
printf("Average: %.2f\n", mean);
printf("Maximum: %.2f\n", maxv);
printf("Minimum: %.2f\n", minv);
printf("Standard Deviation: %.2f\n", stddev);
free(a);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Average: 2.00
Maximum: 3.00
Minimum: 1.00
Standard Deviation: 0.82

Compilation Status: Passed

Execution Time:

0.002s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Average: 10.00
Maximum: 30.00
Minimum: -10.00
Standard Deviation: 14.14

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Average: 45.50
Maximum: 75.50
Minimum: 15.50
Standard Deviation: 20.00

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Average: 0.00
Maximum: 1000000.00
Minimum: -1000000.00
Standard Deviation: 1000000.00

Compilation Status: Passed

Execution Time:

0.001s

16. Problem StatementYou are designing a system for an e-commerce platform where you need to calculate the total cost of items in a shopping cart with multiple discount tiers. Each discount tier provides a different discount rate depending on the total number of items in the cart. You will use recursion to calculate the total cost with discounts applied.

Discount Tiers:

Tier 1: 5% discount for 1 to 5 items.Tier 2: 10% discount for 6 to 10 items.Tier 3: 15% discount for more than 10 items.Your task is to:

Calculate the total cost of items before applying the discount.Apply the discount based on the total number of items.Return the final cost after applying the specific amount of discount.

Functions RequiredcalculateTotalCost(int items[], int n): Recursively calculates the total cost of the items in the cart.applyDiscount(float totalCost, int itemCount): Determines the amount of discount basically based on the number of items and then calculates the final price after applying the discount amount.

InputThe first line contains an integer N representing the number of items in the cart.The second line contains N integers where each integer represents the price of an item.

OutputOutput the final cost of the items in the cart after applying the discount, rounded to two decimal places.

Constraints1 ≤ N ≤ 1000 < items[i] ≤ 500

ExampleInput430 20 50 40

Output124.50

ExplanationThe total cost of items is 30 + 20 + 50 + 40 = 140.The number of items is 4, which falls under Tier 1 (5% discount).Discount = 5% of 140 = 7.Final cost = 140 - 7 = 133.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
double calculateTotalCost(int items[], int n) {
if (n == 0) return 0.0;
return items[n-1] + calculateTotalCost(items, n-1);
}
double applyDiscount(double totalCost, int itemCount) {
double rate = 0.0;
if (itemCount >= 1 && itemCount <= 5) rate = 0.05;
else if (itemCount >= 6 && itemCount <= 10) rate = 0.10;
else if (itemCount > 10) rate = 0.15;
return totalCost * (1.0 - rate);
}
int main(void) {
int N;
if (scanf("%d", &N) != 1) return 0;
if (N < 1 || N > 100) return 0;
int items[100];
for (int i = 0; i < N; ++i) {
if (scanf("%d", &items[i]) != 1) return 0;
}
double total = calculateTotalCost(items, N);
double finalCost = applyDiscount(total, N);
long long v2 = (long long)(finalCost * 100.0 + 0.5);
long long whole = v2 / 100;
int rem = (int)(v2 % 100);
if (rem == 0) {
printf("%lld\n", whole);
}
else if (rem % 10 == 0) {
printf("%lld.%d\n", whole, rem / 10);
}
else {
if (rem < 10) printf("%lld.0%d\n", whole, rem);
else printf("%lld.%02d\n", whole, rem);
}
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

142.5

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

288

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3867.5

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

65.55

Compilation Status: Passed

Execution Time:

0.001s

17. Problem Statement:You need to develop a program that processes and analyzes large datasets of integers. Your task is to manage data using arrays and pointers and perform operations like sorting, searching, and finding the median.

RequirementsData Input: There will be an Input of the array of integers dynamically.Sorting: Sort the array using the quicksort algorithm with pointers.Searching: Implementing the binary search methodology using pointers to find a specific element.Median Calculation: Calculate the median of the sorted array using pointers.

Input FormatThe first line will contain an integer n, such that 1 <= n <= 100000, representing the number of elements in the array.The second line contains n space-separated integers representing the array elements.The third line contains an integer x, representing the element to search in the array.

Output FormatPrint the sorted array of integers on a single line, space-separated.Print the median value of the array on the next line.Print the result of the binary search on the next line: the index of the element x if found, otherwise -1.

ConstraintsAll integers are within the range -1000000 to 1000000.Efficient use of pointers for sorting, searching, and calculating the median.Consider both odd and even lengths for median calculation.

ExampleInput:712 4 5 3 8 7 15

Output:1 3 4 5 7 8 1253

Explanation:Sorting: The sorted array is [1, 3, 4, 5, 7, 8, 12].Median: The median value of this sorted array is 5 since it is the middle element.Binary Search: The element 5 is found at index 3 (0-based index) in the sorted array.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *a, int *b) {
int tmp = *a;
*a = *b;
*b = tmp;
}
int* partition(int *low, int *high) {
int pivot = *high;
int *i = low - 1;
for (int *j = low; j < high; j++) {
if (*j <= pivot) {
i++;
swap(i, j);
}
}swap(i + 1, high);
return i + 1;
}
void quicksort(int *low, int *high) {
if (low < high) {
int *pi = partition(low, high);
quicksort(low, pi - 1);
quicksort(pi + 1, high);
}
}
int binarySearch(int *arr, int n, int x) {
int *low = arr;
int *high = arr + n - 1;
while (low <= high) {
int *mid = low + (high - low) / 2;
if (*mid == x)
return (int)(mid - arr);
else if (*mid < x)
low = mid + 1;
else
high = mid - 1;
}
return -1;
}
double findMedian(int *arr, int n) {
if (n % 2 == 1) {
return *(arr + n / 2);
}
else {
return (*(arr + n / 2 - 1) + *(arr + n / 2)) / 2.0;
}
}
int main() {
int n, x;
scanf("%d", &n);
int *arr = (int*)malloc(n * sizeof(int));
if (!arr) {
return 1;
}
```

```
for (int i = 0; i < n; i++) {
scanf("%d", arr + i);
}
scanf("%d", &x);
quicksort(arr, arr + n - 1);
printf("Sorted array: ");
for (int i = 0; i < n; i++) {
printf("%d", *(arr + i));
if (i != n - 1) printf(" ");
}
printf(" \n");
double median = findMedian(arr, n);
printf("Median: %.2f\n", median);
printf("Binary Search Result: %d\n", binarySearch(arr, n, x));
free(arr);
return 0;}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sorted array: 1 1 3 4 5
Median: 3.00
Binary Search Result: 3

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sorted array: 3 5 7 8 9 10
Median: 7.50

Binary Search Result: 3

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sorted array: -5 -3 -1 0
Median: -2.00
Binary Search Result: 3

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sorted array: 10 20 30 40 50 60 70 80
Median: 45.00
Binary Search Result: -1

Compilation Status: Passed

Execution Time:

0.001s

18. Problem Statement:You need to develop a program that processes and analyzes large datasets of integers. Your task is to manage data using arrays and pointers and perform operations like

sorting, searching, and finding the median.

RequirementsData Input: There will be an Input of the array of integers dynamically.Sorting: Sort the array using the quicksort algorithm with pointers.Searching: Implementing the binary search methodology using pointers to find a specific element.Median Calculation: Calculate the median of the sorted array using pointers.

Input FormatThe first line will contain an integer n, such that 1 <= n <= 100000, representing the number of elements in the array.The second line contains n space-separated integers representing the array elements.The third line contains an integer x, representing the element to search in the array.

Output FormatPrint the sorted array of integers on a single line, space-separated.Print the median value of the array on the next line.Print the result of the binary search on the next line: the index of the element x if found, otherwise -1.

ConstraintsAll integers are within the range -1000000 to 1000000.Efficient use of pointers for sorting, searching, and calculating the median.Consider both odd and even lengths for median calculation.

ExampleInput:712 4 5 3 8 7 15

Output:1 3 4 5 7 8 1253

Explanation:Sorting: The sorted array is [1, 3, 4, 5, 7, 8, 12].Median: The median value of this sorted array is 5 since it is the middle element.Binary Search: The element 5 is found at index 3 (0-based index) in the sorted array.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int partition(int *arr, int low, int high) {
int pivot = arr[high];
int i = low - 1;
for(int j = low; j < high; j++) {
if(arr[j] <= pivot) {
i++;
int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
}
}
int temp = arr[i+1];
```

```
arr[i+1] = arr[high];
arr[high] = temp;
return i + 1;
}
void quicksort(int *arr, int low, int high) {
if(low < high) {
int pi = partition(arr, low, high);
quicksort(arr, low, pi - 1);
quicksort(arr, pi + 1, high);
}
}
int* binary_search(int *arr, int size, int x) {
int low = 0, high = size - 1;
while(low <= high) {
int mid = low + (high - low)/2;if(arr[mid] == x)
return &arr[mid];
else if(arr[mid] < x)
low = mid + 1;
else
high = mid - 1;
}
return NULL;
}
double find_median(int *arr, int n) {
if(n % 2 != 0)
return (double)arr[n/2];
else
return (arr[n/2 - 1] + arr[n/2]) / 2.0;
}
int main() {
int n;
scanf("%d", &n);
int *arr = (int *)malloc(n * sizeof(int));
for(int i = 0; i < n; i++)
scanf("%d", &arr[i]);
int x;
scanf("%d", &x);
quicksort(arr, 0, n - 1);
printf("Sorted array: ");
for(int i = 0; i < n; i++)
printf("%d ", arr[i]);
printf("\n");
double median = find_median(arr, n);
printf("Median: %.2lf\n", median);
int *found = binary_search(arr, n, x);
if(found)
printf("Binary Search Result: %d\n", *found);
else
printf("Binary Search Result: -1\n");
free(arr);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sorted array: 1 1 3 4 5
Median: 3.00
Binary Search Result: 4

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sorted array: 3 5 7 8 9 10
Median: 7.50
Binary Search Result: 8

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sorted array: -5 -3 -1 0
Median: -2.00
Binary Search Result: 0

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sorted array: 10 20 30 40 50 60 70 80
Median: 45.00
Binary Search Result: -1

Compilation Status: Passed

Execution Time:

0.001s

19. Problem Statement:Scenario:You are tasked with developing a menu-driven program for a small library management system. The program should allow users to perform various operations such as adding a book, searching for a book by its ID, deleting a book by its ID, displaying all books, and exiting the program.

Input:The program will repeatedly display a menu with the following options:

Add a bookSearch for a book by IDDelete a book by IDDisplay all booksExitFor Option 1 (Add a book): The user will input a book ID (integer) and a book title (string).For Option 2 (Search for a book by ID): The user will input the book ID to search for.For Option 3 (Delete a book by ID): The user will input the book ID to delete.For Option 4 (Display all books): The program will display all the books currently in the system.For Option 5 (Exit): The program will terminate.

Output:

For Option 1: The program should confirm that the book was added.For Option 2: The program should display the book details if found or notify the user if the book is not

found.For Option 3: The program should confirm that the book was deleted or notif
the user if the book ID is not found.For Option 4: The program should list all books i
the system.For Option 5: The program will exit with a message.

Constraints:

The system can hold a maximum of 100 books.The book ID must be unique.The title should not exceed 50 characters.Input validation should be performed for book ID to ensure it is a positive integer.

Example:Input:1. Add a book2. Search for a book by ID3. Delete a book by ID4. Display all books5. Exit

Choose an option: 1Enter Book ID: 101Enter Book Title: "C Programming Language"

Choose an option: 1Enter Book ID: 102Enter Book Title: "Data Structures in C"

Choose an option: 2Enter Book ID: 101

Choose an option: 4

Choose an option: 3Enter Book ID: 102

Choose an option: 5

Output:Book added successfully.Book added successfully.Book found: ID=101, Title="C Programming Language"Book ID=101, Title="C Programming Language"Book deleted successfully.Exiting the program...

Explanation:

The user adds two books with IDs 101 and 102. Program usually confirms the addition of each book.The user searches for the book with ID 101, and the program displays its details.The user displays all books, and the program lists the available book.The user deletes the book with ID 102, and the program confirms the deletion.The user exits the program.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
#define MAX_BOOKS 100
#define MAX_TITLE_LENGTH 50
typedef struct {
int id;
char title[MAX_TITLE_LENGTH];
} Book;
```

```
Book library[MAX_BOOKS];
int bookCount = 0;
void addBook() {
if (bookCount >= MAX_BOOKS) {
printf("Library is full, cannot add more books.\n");
return;
}
int id;
char title[MAX_TITLE_LENGTH];
if (scanf("%d", &id) != 1) return;
for (int i = 0; i < bookCount; i++) {
if (library[i].id == id) {
printf("Book ID must be unique. Book not added.\n");
return;
}
}
getchar();if (fgets(title, MAX_TITLE_LENGTH, stdin) == NULL) return;
title[strcspn(title, "\n")] = 0;
library[bookCount].id = id;
strcpy(library[bookCount].title, title);
bookCount++;
printf("Book added successfully.\n");
}
void searchBookById() {
int id;
if (scanf("%d", &id) != 1) return;
for (int i = 0; i < bookCount; i++) {
if (library[i].id == id) {
printf("Book found: ID=%d, Title=\"%s\"\n", library[i].id, library[i].title);
return;
}
}
printf("Book not found.\n");
}
void deleteBookById() {
int id;
if (scanf("%d", &id) != 1) return;
for (int i = 0; i < bookCount; i++) {
if (library[i].id == id) {
for (int j = i; j < bookCount - 1; j++) {
library[j] = library[j + 1];
}
bookCount--;
printf("Book deleted successfully.\n");
return;
}
}
printf("Book ID not found.\n");
}
void displayAllBooks() {
if (bookCount == 0) {
printf("No books in the library.\n");
return;
}
```

```
for (int i = 0; i < bookCount; i++) {
printf("Book ID=%d, Title=\"%s\"\n", library[i].id, library[i].title);
}
}
int main() {
int choice;
while (scanf("%d", &choice) == 1) {
switch (choice) {
case 1:
addBook();
break;
case 2:
searchBookById();
break;
case 3:
deleteBookById();break;
case 4:
displayAllBooks();
break;
case 5:
return 0;
default:
printf("Invalid option.\n");
}
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Book added successfully.
Book found: ID=201, Title="Algorithms"

Compilation Status: Passed

Execution Time:

0.002s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Book added successfully.
Book found: ID=101, Title="Data Structures"

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Book added successfully.
Book ID must be unique. Book not added.

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Book added successfully.
Book deleted successfully.
Book not found.

Compilation Status: Passed

Execution Time:

0.001s

20. Problem Statement:You are given a sorted array arr[] of size n which may contain duplicate elements.Your task is to write a C program to find the first and last occurrence indices of a given element x.If the element is not present, return -1 -1.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
int n, x;
scanf("%d", &n);

int arr[n];
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}

scanf("%d", &x);

int first = -1, last = -1;

// Find first occurrence
for (int i = 0; i < n; i++) {
if (arr[i] == x) {
first = i;
break;
}
}

// Find last occurrence
for (int i = n - 1; i >= 0; i--) {
if (arr[i] == x) {
last = i;
break;
}
}

printf("%d %d\n", first, last);
```

```
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 3

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0 1

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3 3

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

-1 -1

Compilation Status: Passed

Execution Time:

0.001s

21. Problem Statement: Dynamic Aggregation of Different Types of ValuesIn a financial application, you need to aggregate different types of values. These values could be monetary amounts, item counts, or percentages. The challenge is that the number of values and their types vary with each call. Depending on the type of aggregation, your task is to either sum, find the maximum, or average values of a specific type, all while working with a variable number of arguments.

Input:The first input is a string that defines the aggregation operation. It can be one of three values:"sum": Sum the values of a specified type."max": Find the maximum value of a specified type."average": Calculate the average of the values of a specified type.The second input is a string that defines the type of value being aggregated. It can be:"monetary": Represents monetary amounts."count": Represents item counts."percentage": Represents percentage values.The third input is an integer n (number of values to be processed).The fourth input is a list of n integers representing the values of the specified type.

Output:Print the result of the specified aggregation operation on the specified type of values.

Problem Statement:Given an aggregation operation ("sum", "max", "average") and a type of value ("monetary", "count", "percentage"), your task is to implement a function

that performs the specified operation using variable argument handling in C.

Example:Input:summonetary4100 200 300 400Output:1000

Explanation:The operation is "sum", and the values are monetary amounts 100, 200 300, 400. The sum of these values is 1000.

Constraints:1 <= n <= 100-10^6 <= value of each integer <= 10^6The input operation is guaranteed to be one of "sum", "max", or "average".The type of value is guaranteed to be one of "monetary", "count", or "percentage".

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
#include <limits.h>
int main() {
char operation[16];
char valtype[16];
int n;
if (scanf("%15s", operation) != 1) return 0;
if (scanf("%15s", valtype) != 1) return 0;
if (scanf("%d", &n) != 1) return 0;
long long sum = 0;
int x;
int maximum = INT_MIN;
for (int i = 0; i < n; i++) {
if (scanf("%d", &x) != 1) return 0;
sum += x;
if (x > maximum) maximum = x;
}
if (strcmp(operation, "sum") == 0) {
printf("%lld\n", sum);
}
else if (strcmp(operation, "max") == 0) {
printf("%d\n", maximum);
}
else if (strcmp(operation, "average") == 0) {
printf("%lld\n", sum / n);
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1000

Compilation Status: Passed

Execution Time:

0.002s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

70

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

500

Compilation Status: Passed

Execution Time:

0.001s

22. Problem Statement:You are given an array of integers (with possible duplicates). Your task is to sort the array in ascending order using the Bubble Sort algorithm. Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order. The process is repeated until the array is completely sorted.

You need to implement Bubble Sort using C programming.

The program should accept:

The number of elements (n)

The array elements

The output should be the sorted array.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int n;
if (scanf("%d", &n) != 1) return 0;
if (n <= 0) {
printf("\n");
return 0;
}
int *arr = (int*)malloc(sizeof(int) * n);
if (!arr) return 0;
for (int i = 0; i < n; i++) {
if (scanf("%d", &arr[i]) != 1) {
free(arr);
return 0;
}
}
for (int i = 0; i < n - 1; i++) {
int swapped = 0;
```

```
for (int j = 0; j < n - 1 - i; j++) {
if (arr[j] > arr[j + 1]) {
int tmp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = tmp;
swapped = 1;
}
}
if (!swapped) break;
}
for (int i = 0; i < n; i++) {
printf("%d", arr[i]);
if (i != n - 1) printf(" ");  }
printf("\n");
free(arr);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 5 8

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5 6 7 8 9 10

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 2 4 4 6 7 9

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2 5 7 10 12 15 18

Compilation Status: Passed

Execution Time:

0.001s

23. Problem Statement:Write a C program to implement Selection Sort on an array of integers. The program should read the size of the array and the elements from the user, sort the array in ascending order using the Selection Sort algorithm, and print the sorted array.

Requirements:

Input should be taken from the user.

Sorting must be done using Selection Sort only.

Display the sorted array as output.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int n;
if (scanf("%d", &n) != 1) return 0;
if (n <= 0) {
printf("\n");
return 0;
}
int *arr = (int*)malloc(sizeof(int) * n);
if (!arr) return 0;
for (int i = 0; i < n; i++) {
if (scanf("%d", &arr[i]) != 1) {
free(arr);
return 0;
}
}
for (int i = 0; i < n - 1; i++) {
int min_idx = i;
for (int j = i + 1; j < n; j++) {
if (arr[j] < arr[min_idx]) min_idx = j;
}
if (min_idx != i) {
int tmp = arr[i];
arr[i] = arr[min_idx];
arr[min_idx] = tmp;
}
}
for (int i = 0; i < n; i++) {
printf("%d", arr[i]);
if (i != n - 1) printf(" ");
}
printf("\n");
free(arr);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

11 12 22 25 64

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5 10 13 14 29 37

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 4 5 6 7

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

40 50 60 70 80 90

Compilation Status: Passed

Execution Time:

0.001s

24. Problem Statement:Write a program in C to sort a given array using the Insertion Sort algorithm.Your program should:

Take the size of the array as input from the user.

Take the array elements as input.

Sort the array in ascending order using Insertion Sort.

Display the array after sorting.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int n;
if (scanf("%d", &n) != 1) return 0;
if (n <= 0) {
printf("\n");
return 0;
}
```

```
int *arr = (int*)malloc(sizeof(int) * n);
if (!arr) return 0;
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
for (int i = 1; i < n; i++) {
int key = arr[i];
int j = i - 1;
while (j >= 0 && arr[j] > key) {
arr[j + 1] = arr[j];j--;
}
arr[j + 1] = key;
}
for (int i = 0; i < n; i++) {
printf("%d", arr[i]);
if (i != n - 1) printf(" ");
}
printf("\n");
free(arr);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5 6 11 12 13

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5 10 15 20 25 30

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3 5 7 9

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 4 5 6 7

Compilation Status: Passed

Execution Time:

0.002s

25. Problem Statement:Title: Network Packet Manipulation

Scenario-Based Problem:In a specific network communication system, the data is transmitted in the form of packets. Each packet contains metadata in the form of an 8-bit header that encodes various properties such as packet type, priority, encryption status, and error flags. The bits in the header have the following significance:

Bit 0: Packet Type (0 for data, 1 for control)Bit 1: Priority (0 for low, 1 for high)Bit 2:

Encryption Status (0 for unencrypted, 1 for encrypted)Bit 3: Error Detected (0 for no error, 1 for error)Bits 4-7: Reserved for future use

You are tasked with writing a program to manipulate this packet header using bitwi operators. The program should perform the following operations based on user inp

Toggle the Packet Type.Set the Priority to High.Clear the Encryption Status (set to unencrypted).Check if an error is detected.Shift the header left by 1 bit and check the new value.

Input:

An integer representing the 8-bit packet header.Operations specified by the user.

Output:

The resulting packet header after performing the specified operations, both in decimal and binary format.

Constraints:

The header of the packet is an 8-bit integer (which values between 0 and 255).Operations must be performed using bitwise operators.The program should handle edge cases where all bits are set to 0 or 1.

Example:

Assume the initial packet header is 11010010 (210 in decimal).

Step 1: Toggle the Packet Type.The initial header is 11010010. Toggling the Packet Type (Bit 0) flips the first bit, changing the header to 11010011 (211 in decimal).Step 2: Set the Priority to High.The current header is 11010011. Setting the Priority to High (Bit 1) ensures the second bit is set to 1, but it is already 1, so the header remains 11010011 (211 in decimal).Step 3: Clear the Encryption Status.The current header is 11010011. Clearing the Encryption Status (Bit 2) sets the third bit to 0, changing the header to 11010001 (209 in decimal).Step 4: Check if an error is detected.The current header is 11010001. Checking if an error is detected (Bit 3) shows that the fourth bit is 1, indicating an error is present.Step 5: Shift the header left by 1 bit.The current header is 11010001. Shifting the header left by 1 bit shifts all bits left, resulting in 10100010 (162 in decimal).

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
void printBinary8(unsigned int x) {
```

```
x &= 0xFF;
for (int i = 7; i >= 0; i--)
printf("%d", (x >> i) & 1);
printf("\n");
}
int main() {
unsigned int header;if (scanf("%u", &header) != 1) return 0;
header ^= (1u << 0);
header |= (1u << 1);
printf("New Header in Decimal: %u\n", header);
printf("New Header in Binary: ");
printBinary8(header);
header &= ~(1u << 2);
printf("New Header after clearing encryption status: %u\n", header);
printf("New Header in Binary: ");
printBinary8(header);
printf("Error Detected: %s\n", ((header >> 3) & 1) ? "Yes" : "No");
unsigned int shifted = header << 1;
printf("Header after Left Shift by 1 Bit: %u\n", shifted);
printf("New Header in Binary: ");
printBinary8(shifted);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

New Header in Decimal: 211
New Header in Binary: 11010011
New Header after clearing encryption status: 211
New Header in Binary: 11010011
Error Detected: No
Header after Left Shift by 1 Bit: 422
New Header in Binary: 10100110

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

New Header in Decimal: 211
New Header in Binary: 11010011
New Header after clearing encryption status: 211
New Header in Binary: 11010011
Error Detected: No
Header after Left Shift by 1 Bit: 422
New Header in Binary: 10100110

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

New Header in Decimal: 86
New Header in Binary: 01010110
New Header after clearing encryption status: 82
New Header in Binary: 01010010
Error Detected: No
Header after Left Shift by 1 Bit: 164
New Header in Binary: 10100100

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

New Header in Decimal: 3
New Header in Binary: 00000011
New Header after clearing encryption status: 3
New Header in Binary: 00000011
Error Detected: No
Header after Left Shift by 1 Bit: 6
New Header in Binary: 00000110

Compilation Status: Passed

Execution Time:

0.001s

26. Problem Statement: Sorting Numbers Using Selection Sort

Write a program to sort an array of integers in ascending order using Selection Sort.

Description

You are given an array of n integers.

The program should implement the Selection Sort algorithm to sort the array in ascending order.

Selection Sort works by repeatedly finding the minimum element from the unsorted part and moving it to the beginning.

Input Format

First line: Integer n (number of elements)

Second line: n space-separated integers

Output Format

Sorted array in ascending order (space-separated).

Sample Input564 25 12 22 11

Sample Output11 12 22 25 64

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
int main() {
int n;
scanf("%d", &n);int arr[n];
for (int i = 0; i < n; i++)
scanf("%d", &arr[i]);
for (int i = 0; i < n - 1; i++) {
int min_idx = i;
for (int j = i + 1; j < n; j++) {
if (arr[j] < arr[min_idx])
min_idx = j;
}
if (min_idx != i) {
int temp = arr[i];
arr[i] = arr[min_idx];
arr[min_idx] = temp;
}
}
for (int i = 0; i < n; i++) {
printf("%d", arr[i]);
if (i != n - 1) printf(" ");
}
printf("\n");
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 4 5

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5 6 7 8 9 10

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 4 7

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 1 2 2 3 4 4

Compilation Status: Passed

Execution Time:

0.001s

27. Problem Statement: Sort Words by Length Using Selection Sor

Write a program to sort a list of words based on their length using Selection Sort.If two words have the same length, maintain lexicographic order.

Description

Input contains n words.

Sort by word length in ascending order.

If lengths match, compare alphabetically.

Input Format

First line: Integer n

Next n lines: One word per line

Output Format

Sorted list of words.

Sample Input5appleisbananacatdog

Sample Outputiscatdogapplebanana

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
void swap(char a[], char b[]) {
char temp[101];
strcpy(temp, a);
strcpy(a, b);
strcpy(b, temp);
}
int main() {
int n;
scanf("%d", &n);
char words[n][101];
for (int i = 0; i < n; i++)
scanf("%s", words[i]);
for (int i = 0; i < n - 1; i++) {
int min_idx = i;
for (int j = i + 1; j < n; j++) {
```

```
int len_j = strlen(words[j]);
int len_min = strlen(words[min_idx]);
if (len_j < len_min || (len_j == len_min && strcmp(words[j], words[min_idx]) < 0)) {
min_idx = j;
}
}
if (min_idx != i)swap(words[i], words[min_idx]);
}
for (int i = 0; i < n; i++)
printf("%s\n", words[i]);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
ant
bat
cat
dog
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
c
i
in
love
coding
```

language

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

hi
ant
zoo
ball
apple
hello

Compilation Status: Passed

Execution Time:

0.001s

28. Problem Statement: Sort Integers in Ascending Order Using Insertion SortWrite a program to sort an array of integers in ascending order using insertion sort.

Description:The program should take an integer array as input and apply insertion sort to arrange elements in non-decreasing order.

Input Format:

First line: Integer n (array size)

Next line: n integers

Output Format:Sorted array in ascending order.

Sample Input:

59 4 6 2 1

Sample Output:

1 2 4 6 9

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
int main() {
int n;
scanf("%d", &n);
int arr[n];
for (int i = 0; i < n; i++)
scanf("%d", &arr[i]);
for (int i = 1; i < n; i++) {
int key = arr[i];
int j = i - 1;
while (j >= 0 && arr[j] > key) {
arr[j + 1] = arr[j];
j--;
}
arr[j + 1] = key;
}
for (int i = 0; i < n; i++) {
printf("%d", arr[i]);
if (i != n - 1) printf(" ");
}
printf("\n");
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 4 5

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5 6 7 8 9 10

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 2 3 4 4 5

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

42

Compilation Status: Passed

Execution Time:

0.001s

29. Problem Statement: Sort Words Alphabetically Using Insertion Sort

Write a program that sorts words in lexicographical (dictionary) order using the insertion sort algorithm.

Description:

The program should take n words as input and arrange them in alphabetical order.

1. Use insertion sort logic for sorting (do not use built-in sort functions).

2. Lexicographical order means words are ordered like in a dictionary (e.g., "apple" < "banana" < "cherry").

3. Sorting should be case-sensitive (assume all words are lowercase for simplicity).

Input Format:

1. First line: Integer n (number of words)

2. Next n lines: One word per line

Output Format:

Words sorted alphabetically, each printed on a new line

Sample Input:4bananaapplecherrydate

Sample Output:applebananacherrydate

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

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

```
int main(void) {
int n;
if (scanf("%d", &n) != 1) return 0;
char **arr = malloc(n * sizeof(char*));
if (!arr) return 0;
for (int i = 0; i < n; ++i) {char buf[1001];
if (scanf("%1000s", buf) != 1) {
for (int k = 0; k < i; ++k) free(arr[k]);
free(arr);
return 0;
}
arr[i] = strdup(buf);
if (!arr[i]) {
for (int k = 0; k < i; ++k) free(arr[k]);
free(arr);
return 0;
}
}
for (int i = 1; i < n; ++i) {
char *key = arr[i];
int j = i - 1;
while (j >= 0 && strcmp(arr[j], key) > 0) {
arr[j + 1] = arr[j];--j;
}
arr[j + 1] = key;
}
for (int i = 0; i < n; ++i) {
printf("%s\n", arr[i]);
free(arr[i]);
}
free(arr);
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
ant
bat
cat
dog
elephant
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

wolf
xray
yak
zebra

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

bag
ball
banana
band
bat

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

a
apple
apply
sun
super
superb

Compilation Status: Passed

Execution Time:

0.001s

30. Problem Statement:Write a C program to implement Quick Sort to sort an array of integers in ascending order.

Your program should take the number of elements n as input, followed by the array elements.

Use the last element as the pivot.

Print the array after it is sorted.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *a, int *b) {
int t = *a; *a = *b; *b = t;
}
int partition(int arr[], int low, int high) {
int pivot = arr[high];
int i = low - 1;
for (int j = low; j <= high - 1; ++j) {
if (arr[j] <= pivot) {
++i;
```

```
swap(&arr[i], &arr[j]);
}
}swap(&arr[i + 1], &arr[high]);
return i + 1;
}
void quickSort(int arr[], int low, int high) {
if (low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}
int main() {
int n;
if (scanf("%d", &n) != 1) return 0;
if (n <= 0) return 0;
int *arr = (int*)malloc(n * sizeof(int));
for (int i = 0; i < n; ++i) scanf("%d", &arr[i]);
quickSort(arr, 0, n - 1);
for (int i = 0; i < n; ++i) {
if (i) printf(" ");
printf("%d", arr[i]);
}
printf("\n");
free(arr);
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

11 12 22 25 64

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 5 7 9

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

10 30 40 50 70 80 90

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 4

Compilation Status: Passed

Execution Time:

0.001s

31. Problem Statement:Write a C program to implement the Merge Sort algorithm.The program should:

Accept an integer n (size of the array).

Accept n integers as input elements of the array.

Sort the array in ascending order using the Merge Sort technique (divide and conquer).

Print the sorted array as output.

Instructions:

Implement recursive merge sort.

Use a separate function for merging two halves.

Do not use library functions like qsort().

Clearly show input and output in your program.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r) {
int n1 = m - l + 1;
int n2 = r - m;
int *L = (int*)malloc(n1 * sizeof(int));
int *R = (int*)malloc(n2 * sizeof(int));
for (int i = 0; i < n1; ++i) L[i] = arr[l + i];
for (int j = 0; j < n2; ++j) R[j] = arr[m + 1 + j];
int i = 0, j = 0, k = l;
while (i < n1 && j < n2) {
if (L[i] <= R[j]) arr[k++] = L[i++];
else arr[k++] = R[j++];
}
while (i < n1) arr[k++] = L[i++];
while (j < n2) arr[k++] = R[j++];
free(L);
free(R);
}
void mergeSort(int arr[], int l, int r) {
if (l >= r) return;
```

```
int m = l + (r - l) / 2;
mergeSort(arr, l, m);mergeSort(arr, m + 1, r);
merge(arr, l, m, r);
}
int main() {
int n;
if (scanf("%d", &n) != 1) return 0;
if (n <= 0) return 0;
int *arr = (int*)malloc(n * sizeof(int));
for (int i = 0; i < n; ++i) scanf("%d", &arr[i]);
mergeSort(arr, 0, n - 1);
for (int i = 0; i < n; ++i) {
if (i) printf(" ");
printf("%d", arr[i]);
}
printf("\n");
free(arr);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

12 22 25 34 64

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 5 5 6 9

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

6 8 10 12

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

-2 0 5 8 12 18 20

Compilation Status: Passed

Execution Time:

0.001s

32. Problem StatementTitle: Read-Only Data Access in Sensor Systems

In a weather monitoring system, multiple sensors are deployed to collect critical information like temperature, humidity, and wind speed. These sensor readings are stored in memory and should be made read-only to ensure data integrity once the readings are taken. You are tasked with creating a program that collects sensor data, stores it in a way that the values cannot be altered, and displays the collected data.

Your task is to write a C program that uses the const qualifier to store sensor readings in an immutable manner, making sure that no function can alter the senso data once it has been set.

Input:

The first input is an integer n (1 <= n <= 10), representing the number of sensors.Then, you need to take n floating-point numbers, which represent the sensor readings for different environmental parameters.

Output:

The program should output the sensor readings after all inputs have been received, ensuring they remain unaltered.

Constraints:

You must use the const type qualifier to ensure that the sensor readings cannot be modified once initialized.You cannot use any external files for input or output; all input should be handled via standard input methods.Global variables are not allowed.The input values can have up to 2 decimal places, and the output must show values up to 2 decimal places.

Example Input/OutputExample 1:

Input:423.5 78.6 19.2 45.3Output:Sensor Readings:23.5078.6019.2045.30

Example 2:

Input:250.5 60.1Output:Sensor Readings:50.5060.10

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

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

void displayReadings(const float *readings, int n) {
printf("Sensor Readings:\n");
for (int i = 0; i < n; i++) {
printf("%.2f\n", readings[i]);
}
}

int main() {
```

```
int n;
scanf("%d", &n);

if (n < 1 || n > 10) {
printf("Invalid number of sensors.\n");
return 1;
}

float *temp = (float *)malloc(n * sizeof(float));
if (temp == NULL) {
printf("Memory allocation failed.\n");
return 1;
}

for (int i = 0; i < n; i++) {
scanf("%f", &temp[i]);
}

const float *readings = temp;
displayReadings(readings, n);

free(temp);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Sensor Readings:
24.10
18.50
45.90
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sensor Readings:
100.00
55.30

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sensor Readings:
15.60

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Sensor Readings:
32.00
28.90
44.60
90.10
100.50

Compilation Status: Passed

Execution Time:

0.001s

33. Problem Statement: Multifunction AggregatorYou are building a program that processes multiple operations on a variable number of integers passed as arguments. The operations include finding the sum, the product, and the maximum of a list of integers. The number of integers is variable and can differ each time the function is called. You are required to implement a solution using variable argument lists (varargs) in C.

Input:The first input is a string that defines the operation. It will be of three values:"sum": Calculate the sum of the integers."product": Calculate the product of the integers."max": Find the maximum value among the integers.The second input is an integer n (number of integers to be processed).The third input is a list of n integers.

Output:Print a single integer representing the result of the specified operation.

Problem Statement:Given an operation ("sum", "product", or "max") and a variable number of integers, your task is to implement a function that performs the specified operation using C's variable argument capabilities (stdarg.h).

Example:Input:sum51 2 3 4 5Output:15

Explanation:The operation is "sum", and the integers are 1, 2, 3, 4, 5. The sum of these integers is 15.

Constraints:1 <= n <= 100-10^6 <= value of each integer <= 10^6The input operation is guaranteed to be one of "sum", "product", or "max".

Solution Explanation:You need to create a function that takes the operation type and a variable number of integers. Depending on the operation type, the function should either sum, multiply, or find the maximum of the provided integers. This is accomplished using C's stdarg.h library, which allows us to handle functions with a variable number of arguments.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

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

```
#include <string.h>
#include <limits.h>
int aggregate(const char *operation, int count, ...) {
va_list args;
va_start(args, count);
int result;if (strcmp(operation, "sum") == 0) {
result = 0;
for (int i = 0; i < count; i++) {
result += va_arg(args, int);
}
}
else if (strcmp(operation, "product") == 0) {
result = 1;
for (int i = 0; i < count; i++) {
result *= va_arg(args, int);
}
}
else if (strcmp(operation, "max") == 0) {
result = INT_MIN;
for (int i = 0; i < count; i++) {
int val = va_arg(args, int);
if (val > result) result = val;
}
}
else {
result = 0;
}
va_end(args);
return result;
}
int main() {
char operation[10];
int n;
scanf("%s", operation);
scanf("%d", &n);
int values[n];
for (int i = 0; i < n; i++) {
scanf("%d", &values[i]);
}
int result = aggregate(operation, n, values[0], values[1], values[2], values[3], values[4],
values[5], values[6], values[7], values[8], values[9]);
printf("%d\n", result);
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

15

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

24

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

40

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

-5

Compilation Status: Passed

Execution Time:

0.001s

34. Problem Statement: Sort Integers in Descending Order Using Quick SortWrite a program to sort an array of integers in descending order using the Quick Sort algorithm.

Description

Quick Sort is a divide-and-conquer algorithm. The program should take an integer array as input and rearrange the elements in non-increasing order. Use Quick Sort to partition the array around a pivot and recursively sort the sub-arrays.

Input Format

First line: An integer n (size of the array)

Second line: n integers separated by spaces

Output Format

Sorted array in descending order

Sample Input52 9 1 7 3

Sample Output9 7 3 2 1

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
void swap(int *a, int *b) {
int temp = *a;
```

```
*a = *b;
*b = temp;
}
int partition(int arr[], int low, int high) {
int pivot = arr[high];
int i = low - 1;
for (int j = low; j < high; j++) {
if (arr[j] > pivot) {
i++;swap(&arr[i], &arr[j]);
}
}
swap(&arr[i + 1], &arr[high]);
return i + 1;
}
void quickSort(int arr[], int low, int high) {
if (low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}
int main() {
int n;
scanf("%d", &n);
int arr[n];
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
quickSort(arr, 0, n - 1);
for (int i = 0; i < n; i++) {
printf("%d", arr[i]);
if (i < n - 1) printf(" ");
}
printf("\n");
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

25 20 15 10 5 5

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3 0 -2 -5

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

20 15 15 11 9 3 3

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

50 50 23 7 0 -1 -5 -10

Compilation Status: Passed

Execution Time:

0.001s

TestCase5:

Input:

< hidden >

Expected Output:

< hidden >

Output:

9 7 4 2 1

Compilation Status: Passed

Execution Time:

0.001s

35. Problem Statement: Sort Employees by Salary Using Quick Sort

Write a program to sort employee records based on their salary in descending order using Quick Sort. If two employees have the same salary, sort them by their name in ascending lexicographic order.

Description

Each employee record consists of a name (string) and salary (integer). The program should apply Quick Sort to sort by salary first. In case of ties, names should be compared alphabetically.

Input Format

First line: An integer n (number of employees)

Next n lines: Each line contains a string (employee name) and an integer (salary)

Output Format

Sorted list of employees with their salaries

Sample Input3Amit 50000Riya 60000Karan 50000

Sample OutputRiya 60000Amit 50000Karan 50000

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
typedef struct {
char name[101];
int salary;
} Employee;
void swap(Employee *a, Employee *b) {
Employee temp = *a;
*a = *b;
*b = temp;
}
int compare(Employee a, Employee b) {
if (a.salary > b.salary) return 1;
else if (a.salary < b.salary) return 0;
else {
if (strcmp(a.name, b.name) < 0) return 1;
else return 0;
}
}
int partition(Employee arr[], int low, int high) {
Employee pivot = arr[high];
int i = low - 1;
for (int j = low; j < high; j++) {
if (compare(arr[j], pivot)) {
i++;
swap(&arr[i], &arr[j]);
}
}
swap(&arr[i + 1], &arr[high]);
return i + 1;
}
void quickSort(Employee arr[], int low, int high) {
if (low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}
int main() {
int n;
scanf("%d", &n);
Employee arr[n];
for (int i = 0; i < n; i++) {
scanf("%s %d", arr[i].name, &arr[i].salary);
}
quickSort(arr, 0, n - 1);
for (int i = 0; i < n; i++) {
```

```
printf("%s %d\n", arr[i].name, arr[i].salary);
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Alice 75000
Bob 75000
Eve 50000
John 40000

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Arjun 70000
Meena 70000
Kiran 60000
Ravi 55000
Zara 45000

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Jerry 30000
Tom 30000

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Priya 60000
Vikram 60000
Aman 40000
Zoya 40000
Isha 30000

Compilation Status: Passed

Execution Time:

0.001s

36. Problem Statement: Sorting Student Records Using Merge Sort

Write a program that sorts student records based on their marks using Merge Sort.

Description:Each student record contains a name and marks. The program should sort the students in descending order of marks. If marks are the same, sort by name in ascending order.

Input Format:

First line: Integer n (number of students)

Next n lines: name marks

Output Format:Sorted list of students with their marks.

Sample Input:

4Amit 78Raj 90Sneha 78Kiran 85

Sample Output:

Raj 90Kiran 85Amit 78Sneha 78

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
typedef struct {
char name[101];
int marks;
} Student;
void merge(Student arr[], int left, int mid, int right) {
int n1 = mid - left + 1;
int n2 = right - mid;
Student L[n1], R[n2];
for (int i = 0; i < n1; i++)
L[i] = arr[left + i];
for (int j = 0; j < n2; j++)
R[j] = arr[mid + 1 + j];
int i = 0, j = 0, k = left;
while (i < n1 && j < n2) {
if (L[i].marks > R[j].marks ||
(L[i].marks == R[j].marks && strcmp(L[i].name, R[j].name) < 0)) {
arr[k++] = L[i++];
}
else {
arr[k++] = R[j++];
}
}
while (i < n1) arr[k++] = L[i++];
while (j < n2) arr[k++] = R[j++];
}
void mergeSort(Student arr[], int left, int right) {
if (left < right) {
int mid = left + (right - left)/2;
mergeSort(arr, left, mid);
```

```
mergeSort(arr, mid+1, right);
merge(arr, left, mid, right);
}
}
int main() {
int n;
scanf("%d", &n);Student arr[n];
for (int i = 0; i < n; i++) {
scanf("%s %d", arr[i].name, &arr[i].marks);
}
mergeSort(arr, 0, n - 1);
for (int i = 0; i < n; i++) {
printf("%s %d\n", arr[i].name, arr[i].marks);
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Bob 70
Charlie 60
Alice 55

Compilation Status: Passed

Execution Time:

0.002s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Bob 70
Charlie 60

Alice 55

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Amit 80
Kiran 80
Ravi 80
Zoya 80

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Anil 92
Rohan 92
Kavya 88
Meena 88
Sneha 70

Compilation Status: Passed

Execution Time:

0.001s

37. Problem Statement Sorting Fractions Using Merge Sort

Write a program to sort an array of fractions in ascending order using Merge Sort.Each fraction is given as numerator/denominator. The fractions must be sorte based on their decimal values.

Description

1. You are given n fractions, where each fraction is represented by two integers: numerator and denominator.

2. The task is to sort these fractions in ascending order according to their decimal values (i.e., numerator ÷ denominator).

3. Use the Merge Sort algorithm for sorting.

The program should handle:

1. Proper fractions (numerator < denominator)

2. Improper fractions (numerator ≥ denominator)

3. Negative fractions

Input Format

First line: An integer n representing the number of fractions

Next n lines: Each line contains two integers separated by space numerator denominator

Output Format

Print the sorted fractions in ascending order, one per line, in the format:

numerator/denominator

Sample Input 141 23 41 35 6

Sample Output 11/31/23/45/6

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
typedef struct {
int numerator;
int denominator;
} Fraction;
```

```
void merge(Fraction arr[], int left, int mid, int right) {
int n1 = mid - left + 1;
int n2 = right - mid;
Fraction L[n1], R[n2];
for (int i = 0; i < n1; i++) L[i] = arr[left + i];
for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j]; int i = 0, j = 0, k = left;
while (i < n1 && j < n2) {
if ((long long)L[i].numerator * R[j].denominator <= (long long)R[j].numerator *
L[i].denominator) {
arr[k++] = L[i++];
} else {
arr[k++] = R[j++];
}
}
while (i < n1) arr[k++] = L[i++];
while (j < n2) arr[k++] = R[j++];
}
void mergeSort(Fraction arr[], int left, int right) {
if (left < right) {
int mid = left + (right - left)/2;
mergeSort(arr, left, mid);
mergeSort(arr, mid + 1, right);
merge(arr, left, mid, right);
}
}
int main() {
int n;
scanf("%d", &n);
Fraction arr[n];
for (int i = 0; i < n; i++) {
scanf("%d %d", &arr[i].numerator, &arr[i].denominator);
}
mergeSort(arr, 0, n - 1);
for (int i = 0; i < n; i++) {
printf("%d/%d\n", arr[i].numerator, arr[i].denominator);
}
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1/2

2/4
3/6

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

-3/4
-1/2
1/3
2/5

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3/2
5/2
7/2

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

-3/4
-1/2
2/3

Compilation Status: Passed

Execution Time:

0.001s

38. Problem Statement:Dynamic Memory Management in Linked List Using Pointers and Functions

You're needed to produce a dynamic linked list of integers using pointers and functions in C. The task involves several operations similar as fitting new bumps, deleting bumps grounded on their values, and reversing the linked list. You must produce individual functions for insertion, omission, and reversal, all while using pointer manipulation. This simulates real- world scripts of managing dynamic data structures like linked lists in memory- constrained systems or databases.

Problem DescriptionThe problem involves enforcing a independently linked list with the following operations

fit fit a knot at the end of the linked list.cancel cancel all bumps with a given value from the linked list.Rear Rear the entire linked list.The program should take the number of operations and also execute each operation on the linked list. After performing all the operations, the final state of the linked list should be published.

Input FormatThe first line contains an integer Q, which represents the number of operations to be performed.Each of the coming Q lines contains an operationThe operation" INSERT X" means to fit an integer X at the end of the linked list.The operation" cancel X" means to cancel all bumps that contain the integer X.The operation" REVERSE" means to reverse the linked list.

Output FormatAfter performing all operations, publish the final state of the linked list. publish EMPTY if the linked list is empty.

Constraints1 ≤ Q ≤ 1000(Number of operations).* 109 ≤ X ≤ 109(Value of integer bumps).

illustrationInput5INSERT 1INSERT 2INSERT 3cancel 2REVERSE

Output3 1

Explanationoriginally, the linked list is empty. After performing the operations

Insert 1, Insert 2, and fit 3 results in the list 1-> 2-> 3.Deleting 2 results in the list 1-> 3.Reversing the list results in 3-> 1.The final affair is 3 1.

ApproachTo achieve this, you will

apply a linked list using a struct for the bumps and stoutly manage memory using pointers.produce separate functions for insertion, omission, and reversal.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node {
int data;
struct Node *next;
} Node;
void insert(Node **head, int value) {
Node *newNode = (Node *)malloc(sizeof(Node));
newNode->data = value;
newNode->next = NULL;
if (*head == NULL) {
*head = newNode;
}
else {
Node *temp = *head;
while (temp->next != NULL)
temp = temp->next;
temp->next = newNode;
}
}
void deleteValue(Node **head, int value) {
Node *temp = *head, *prev = NULL;
while (temp != NULL) {
if (temp->data == value) {
Node *toDelete = temp;
if (prev == NULL) {
*head = temp->next;
}
else {
prev->next = temp->next;
}
temp = temp->next;
free(toDelete);
```

```
}
else {
prev = temp;
temp = temp->next;
}
}
}
void reverse(Node **head) {Node *prev = NULL, *current = *head, *next = NULL;
while (current != NULL) {
next = current->next;
current->next = prev;
prev = current;
current = next;
}
*head = prev;
}
void printList(Node *head) {
if (head == NULL) {
printf("EMPTY\n");
return;
}
Node *temp = head;
while (temp != NULL) {
printf("%d", temp->data);
if (temp->next != NULL) printf(" ");
temp = temp->next;
}
printf("\n");
}
int main() {
int Q;
scanf("%d", &Q);
getchar();
Node *head = NULL;
char command[100];
for (int i = 0; i < Q; i++) {
fgets(command, sizeof(command), stdin);
if (strncmp(command, "INSERT", 6) == 0) {
int value;
sscanf(command + 7, "%d", &value);
insert(&head, value);
}
else if (strncmp(command, "DELETE", 6) == 0) {
int value;
sscanf(command + 7, "%d", &value);
deleteValue(&head, value);
}
else if (strncmp(command, "REVERSE", 7) == 0) {
reverse(&head);
}
}
printList(head);
Node *temp;
while (head != NULL) {
```

```
temp = head;
head = head->next;
free(temp);
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3 1

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

10

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

100

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

80 70

Compilation Status: Passed

Execution Time:

0.001s

39. Problem Statement: Distributed Data Management Using Linked ListsTitle: Merge and Sort K Linked Lists in a Distributed System

Problem Description:In a distributed system, data is partitioned into multiple servers. Each server holds a linked list of integers. The system periodically merges the data from these servers into a single, sorted list for centralized processing. You are tasked with simulating this process by merging K sorted linked lists into a single sorted linked list.

Your task is to write a C program that takes K linked lists and merges them into one sorted list. The merging process must maintain the original order of elements in each individual list. The challenge lies in handling large amounts of data efficiently, ensuring the merged list is sorted in ascending order.

Input:The first line contains an integer K, the number of linked lists (1 ≤ K ≤ 100).For each of the next K lines:The first integer N indicates the number of elements in the list (1 ≤ N ≤ 10^4).The next N integers represent the elements of the linked list in ascending order.

Output:Print the elements of the merged and sorted linked list.

Constraints:Each linked list is already sorted in ascending order.The total number of

elements across all lists will not exceed 10^5.Efficient merging is required to handle large datasets within time limits.

Example Input and Output:Input:34 1 4 7 83 2 5 95 0 3 6 10 11Output:0 1 2 3 4 5 6 7 9 10 11

Detailed Explanation:Test Case 1:

Linked List 1: 1 -> 4 -> 7 -> 8Linked List 2: 2 -> 5 -> 9Linked List 3: 0 -> 3 -> 6 -> 10 -> 11You need to merge these three sorted linked lists into one sorted list. The final merged list should look like:0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
int data;
struct Node *next;
} Node;
Node* createNode(int value) {
Node* newNode = (Node*)malloc(sizeof(Node));
newNode->data = value;
newNode->next = NULL;
return newNode;
}
void append(Node **head, int value) {
Node* newNode = createNode(value);
if (*head == NULL) {
*head = newNode;
return;
}
Node* temp = *head;
while (temp->next != NULL) temp = temp->next;
temp->next = newNode;
}
Node* mergeTwoLists(Node* l1, Node* l2) {
Node dummy;
Node* tail = &dummy;
dummy.next = NULL;
while (l1 && l2) {
if (l1->data <= l2->data) {
tail->next = l1;
l1 = l1->next;
```

```
}
else {
tail->next = l2;
l2 = l2->next;
}
tail = tail->next;
}
if (l1) tail->next = l1;
if (l2) tail->next = l2;
return dummy.next;
}
Node* mergeKLists(Node** lists, int K) {if (K == 0) return NULL;
int interval = 1;
while (interval < K) {
for (int i = 0; i + interval < K; i += interval * 2) {
lists[i] = mergeTwoLists(lists[i], lists[i + interval]);
}
interval *= 2;
}
return lists[0];
}
void printList(Node* head) {
Node* temp = head;
while (temp != NULL) {
printf("%d", temp->data);
if (temp->next != NULL) printf(" ");
temp = temp->next;
}
printf("\n");
}
int main() {
int K;
scanf("%d", &K);
Node* lists[K];
for (int i = 0; i < K; i++) {
lists[i] = NULL;
int N;
scanf("%d", &N);
for (int j = 0; j < N; j++) {
int val;
scanf("%d", &val);
append(&lists[i], val);
}
}
Node* mergedList = mergeKLists(lists, K);
printList(mergedList);
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0 1 2 3 4 5 6 7 8 9 10 11

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 3 5 8 9 12 13 15 20

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0 1 2 3 4 5 6 7 10 11 12

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 2 3 4 5

Compilation Status: Passed

Execution Time:

0.001s

40. Problem StatementTitle: Secure Configuration Settings for an Embedded Device

In an embedded system for controlling a ballast water management system, you need to ensure that certain critical configuration settings are never altered after they are set. These settings are used to control various parameters like water levels, salinity thresholds, and ballast pump speed.

You need to write a C program that sets these configurations in a secure and non-mutable way using the const qualifier. Once the values are set, they should never be modified by any function in your program.

Input:

The first input is an integer n (1 <= n <= 5), which represents the number of configurations to set.Then, you need to take n floating-point numbers which represent the configuration values for the device's ballast control parameters.

Output:

After the input is received, display the configuration values.Your task is to ensure these values cannot be altered after being set, even if later functions in the program attempt to do so.

Constraints:

You must use the const type qualifier to ensure immutability of configuration values.You cannot use any external files for input or output, and all input should be taken through standard input methods.No global variables are allowed.

ExplanationThe program takes in the number of configurations and their values, stores them in a way that prevents modification using const. The output is the same set of configuration values as a confirmation that they have been stored correctly.

Example Input/OutputExample 1:

Input:31.2 3.4 5.6Output:Configuration Values:1.23.45.6

Example 2:

Input:19.81Output:Configuration Values:9.81

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
int n;
scanf("%d", &n); // Number of configurations

const float config[n]; // Array of const floats
float temp;

// Read inputs into temporary variable first
for(int i = 0; i < n; i++) {
scanf("%f", &temp);
((float*)config)[i] = temp; // cast to allow initialization
}

// Output the configuration values
printf("Configuration Values:\n");
for(int i = 0; i < n; i++) {
printf("%.5f\n", config[i]);
}

return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Configuration Values:
1.20000
3.40000
5.60000

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Configuration Values:
10.50000
20.10000

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Configuration Values:
9.81000

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Configuration Values:
100.00000
50.50000
25.25000
12.12000

Compilation Status: Passed

Execution Time:

0.001s

41. Problem Statement: "Discount Based on Membership Level"A retail store offers different discounts based on the customer's membership level. You are tasked to write a program that calculates the discount based on the membership level using conditional compilation and macros.

The store offers the following discounts:

Bronze Membership: 5% discountSilver Membership: 10% discountGold Membership: 15% discountPlatinum Membership: 20% discountThe program should use conditional compilation to apply the correct discount for the membership level chosen. Use macros to define the discount percentages and apply them based on the membership level.

Input:The total bill amount (a floating-point number).The membership level of the customer (an integer representing the level: 1 for Bronze, 2 for Silver, 3 for Gold, and 4 for Platinum).

Output:The discount percentage and the final price after applying the discount.

Constraints:The total bill amount will be a positive float number, between 0.0 and 10,000.0.Membership level will be an integer between 1 and 4.The solution must be compatible with online compilers.

Example:Input:2000.03Output:Gold Membership: 15% DiscountFinal Price: 1700.00

Explanation:The total bill amount is 2000.0, and the customer has a Gold membership (level 3).The program calculates a 15% discount (Gold Membership), reducing the price by 300.0, resulting in a final price of 1700.0.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>

#define BRONZE_DISCOUNT   5
#define SILVER_DISCOUNT   10
#define GOLD_DISCOUNT     15
#define PLATINUM_DISCOUNT 20

int main() {
float billAmount;
int membershipLevel;

scanf("%f", &billAmount);
scanf("%d", &membershipLevel);

int discount = 0;
char membershipName[30] = "";

#if 1
if (membershipLevel == 1) {
discount = BRONZE_DISCOUNT;
sprintf(membershipName, "Bronze Membership");
} else if (membershipLevel == 2) {
discount = SILVER_DISCOUNT;
sprintf(membershipName, "Silver Membership");
} else if (membershipLevel == 3) {
discount = GOLD_DISCOUNT;
sprintf(membershipName, "Gold Membership");
} else if (membershipLevel == 4) {
discount = PLATINUM_DISCOUNT;
sprintf(membershipName, "Platinum Membership");
} else {
printf("Invalid Membership Level\n");
return 0;
}
#endif

float finalPrice = billAmount * (1 - discount / 100.0);

printf("%s: %d%% Discount\n", membershipName, discount);
printf("Final Price: %.2f\n", finalPrice);

return 0;
```

}

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Bronze Membership: 5% Discount
Final Price: 950.00

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Silver Membership: 10% Discount
Final Price: 2250.00

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Gold Membership: 15% Discount
Final Price: 3400.00

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Platinum Membership: 20% Discount
Final Price: 4000.00

Compilation Status: Passed

Execution Time:

0.001s

42. Problem Statement:

Reverse a Linked ListWrite a program to reverse a singly linked list.

Description

The program should take n integers as input to create a singly linked list.

Reverse the linked list and print the elements from the new head to tail.

Do not use extra arrays for storing elements; reverse the list in place.

Input Format

First line: Integer n (number of elements in the linked list)

Next line: n integers representing the linked list elements

Output Format

Elements of the reversed linked list in a single line separated by spaces

Sample Input510 20 30 40 50

Sample Output50 40 30 20 10

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {int data;
struct Node* next;
} Node;
Node* createNode(int value) {
Node* newNode = (Node*)malloc(sizeof(Node));
newNode->data = value;
newNode->next = NULL;
return newNode;
}
void append(Node** head, int value) {
Node* newNode = createNode(value);
if (*head == NULL) {
*head = newNode;
return;
}
Node* temp = *head;
while (temp->next != NULL)
temp = temp->next;
temp->next = newNode;
}
void reverseList(Node** head) {
Node* prev = NULL;
Node* current = *head;
Node* next = NULL;
while (current != NULL) {
next = current->next;
current->next = prev;
prev = current;
current = next;
}
*head = prev;
}
void printList(Node* head) {
Node* temp = head;
while (temp != NULL) {
printf("%d", temp->data);
if (temp->next != NULL)
printf(" ");
temp = temp->next;
}
```

```
printf("\n");
}
int main() {
int n;
scanf("%d", &n);
Node* head = NULL;
for (int i = 0; i < n; i++) {
int val;
scanf("%d", &val);
append(&head, val);
}
reverseList(&head);
printList(head);
Node* temp;while (head != NULL) {
temp = head;
head = head->next;
free(temp);
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

6 5 4 3 2 1

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

-4 -3 -2 -1

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

15 5

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

8 0 7 -3 5 -10

Compilation Status: Passed

Execution Time:

0.001s

43. Problem Statement:Detect a Cycle in a Linked List

Check if a singly linked list contains a cycle.

Description

The program should take n integers as input to create a singly linked list.

Detect if the list contains a loop (cycle).

If a cycle exists, print "Cycle Found". Otherwise, print "No Cycle".

Use Floyd’s cycle detection algorithm (Tortoise and Hare) or equivalent.

Input Format

First line: Integer n (number of elements in the linked list)

Next line: n integers representing the linked list elements

(For testing, the cycle may be created manually in code, or a position index can be specified to link the last node to a previous node.)

Output Format

"Cycle Found" if a cycle exists

"No Cycle" if there is no cycle

Sample Input51 2 3 4 5-1

Sample OutputNo Cycle

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
int data;
struct Node* next;
} Node;
Node* createNode(int value) {
Node* newNode = (Node*)malloc(sizeof(Node));
newNode->data = value;
newNode->next = NULL;
return newNode;
}
void append(Node** head, int value) {
Node* newNode = createNode(value);
if (*head == NULL) {
*head = newNode;
return;
}
Node* temp = *head;
```

```
while (temp->next != NULL)
temp = temp->next;
temp->next = newNode;
}
void createCycle(Node* head, int pos) {
if (pos == -1) return;
Node* temp = head;
Node* cycleNode = NULL;
int index = 0;
while (temp->next != NULL) {
if (index == pos)
cycleNode = temp;
temp = temp->next;
index++;
}if (cycleNode != NULL)
temp->next = cycleNode;
}
int hasCycle(Node* head) {
Node* slow = head;
Node* fast = head;
while (fast != NULL && fast->next != NULL) {
slow = slow->next;
fast = fast->next->next;
if (slow == fast)
return 1;
}
return 0;
}
int main() {
int n;
scanf("%d", &n);
Node* head = NULL;
for (int i = 0; i < n; i++) {
int val;
scanf("%d", &val);
append(&head, val);
}
int pos;
scanf("%d", &pos);
createCycle(head, pos);
if (hasCycle(head))
printf("Cycle Found\n");
else
printf("No Cycle\n");
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Cycle Found

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Cycle Found

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Cycle Found

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Cycle Found

Compilation Status: Passed

Execution Time:

0.001s

44. Problem Statement: Complex Playlist Manager Using Doubly Linked ListTitle: Design a Playlist with Dynamic Operations on Songs Using a Doubly Linked List

Problem Description:You are tasked with implementing a playlist manager for a music app. Each song in the playlist is represented by a doubly linked list node that stores the song's name and duration in seconds. Users can perform the following operations on the playlist:

Add a song at the end.Delete a song from the playlist by its name.Move to the next song.Move to the previous song.Print the entire playlist (from the current song to the end of the list).Your task is to design a program that implements these functionalities using a doubly linked list.

Input:The first line contains an integer Q, the number of operations to be performed (1 ≤ Q ≤ 1000).Each of the next Q lines contains an operation in one of the following formats:ADD : Adds the song at the end of the playlist.DELETE : Deletes the song with the given name.NEXT: Moves to the next song in the playlist (if available).PREV: Moves to the previous song in the playlist (if available).PRINT: Prints the playlist starting from the current song.

Output:For every PRINT operation, print the names of the songs from the current position to the end of the playlist.If a DELETE operation attempts to remove a song that does not exist, print "Song not found".If a NEXT or PREV operation cannot move (e.g., you are at the end or the start of the playlist), print "No next song" or "No previous song", respectively.

Constraints:The song name will be a string of up to 100 characters.The duration will be a positive integer ≤ 10^4.The total number of songs in the playlist will not exceed 10^3.The total number of operations will not exceed 10^3.

Example Input and Output:Input:7ADD SongA 180ADD SongB 200ADD SongC 220NEXTPRINTPREVPRINT

Output:SongB SongCSongA SongB SongC

Detailed Explanation:Test Case 1:

Initially, we add three songs to the playlist:

SongA (duration: 180s)SongB (duration: 200s)SongC (duration: 220s)Then we mov to the next song, which brings the current song to SongB.

We print the playlist from the current song (SongB), and the output is:SongB SongC

After that, we move to the previous song, bringing the current song back to SongA.

Printing the playlist from SongA results in:SongA SongB SongC

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Song {
char name[101];
int duration;
struct Song* prev;
struct Song* next;
} Song;
Song* createSong(char* name, int duration) {
Song* newSong = (Song*)malloc(sizeof(Song));
strcpy(newSong->name, name);
newSong->duration = duration;
newSong->prev = newSong->next = NULL;
return newSong;
}
void addSong(Song** head, Song** tail, char* name, int duration) {
Song* newSong = createSong(name, duration);
if (*head == NULL) {
*head = *tail = newSong;
}
else {
(*tail)->next = newSong;
newSong->prev = *tail;
*tail = newSong;
}
}
void deleteSong(Song** head, Song** tail, Song** current, char* name) {
Song* temp = *head;
while (temp != NULL) {
if (strcmp(temp->name, name) == 0) {
```

```
if (temp->prev) temp->prev->next = temp->next;
else *head = temp->next; // deleting head
if (temp->next) temp->next->prev = temp->prev;
else *tail = temp->prev; // deleting tail
if (*current == temp) {
*current = temp->next ? temp->next : temp->prev;
}
free(temp);
return;
}
temp = temp->next;
}
printf("Song not found\n");
}void nextSong(Song** current) {
if (*current && (*current)->next)
*current = (*current)->next;
else
printf("No next song\n");
}
void prevSong(Song** current) {
if (*current && (*current)->prev)
*current = (*current)->prev;
else
printf("No previous song\n");
}
void printPlaylist(Song* current) {
Song* temp = current;
while (temp) {
printf("%s", temp->name);
if (temp->next) printf(" ");
temp = temp->next;
}
printf("\n");
}
int main() {
int Q;
scanf("%d", &Q);
getchar();
Song* head = NULL;
Song* tail = NULL;
Song* current = NULL;
char operation[10];
for (int i = 0; i < Q; i++) {
scanf("%s", operation);
if (strcmp(operation, "ADD") == 0) {
char name[101];
int duration;
scanf("%s %d", name, &duration);
addSong(&head, &tail, name, duration);
if (!current) current = head; // first song added
}
else if (strcmp(operation, "DELETE") == 0) {
char name[101];
scanf("%s", name);
```

```
deleteSong(&head, &tail, &current, name);
}
else if (strcmp(operation, "NEXT") == 0) {
nextSong(&current);
}
else if (strcmp(operation, "PREV") == 0) {
prevSong(&current);
}
else if (strcmp(operation, "PRINT") == 0) {
if (current) printPlaylist(current);
}
}
Song* temp;
while (head) {
temp = head;
head = head->next;
free(temp);  }
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
SongB SongC
SongA SongB SongC
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

No next song
Music2

Compilation Status: Passed

Execution Time:

0.001s

45. Problem Statement: File System Simulation Using Advanced Data Structures

Title: File System Simulation with Hierarchical Structure and Memory Management

Problem Description:You are tasked with developing a simplified file system simulation for a large enterprise environment. The file system needs to support multiple operations like creating directories, creating files within directories, updating files, deleting files, and querying directories. Each directory can contain multiple files and subdirectories. Files and directories should be organized hierarchically, and the memory used by files and directories needs to be managed efficiently.

In this problem, you'll use dynamic data structures like trees and linked lists to simulate the file system. The operations include adding, deleting, updating files, and directories, as well as querying the system to return directory contents.

Supported Operations:Create a directory: Create a new directory under an existing directory.Create a file: Add a new file with given content in a specific directory.Update a file: Modify the content of an existing file in a directory.Delete a file: Remove a file from a directory.Query directory: List all files and subdirectories in a given directory, sorted lexicographically.Get directory size: Calculate the total size of all files in a given directory (including subdirectories).Your task is to implement this file system simulation in C using appropriate data structures (e.g., tree for directories and linked list for files).

Input:The first line contains an integer n, the number of operations (1 ≤ n ≤ 10^5).The next n lines describe operations in one of the following formats:1 parent_dir new_dir: Create a new directory new_dir under parent_dir.2 dir_name file_name file_size: Create a new file with file_name and size file_size in dir_name.3 dir_name file_name new_size: Update the content of the file file_name in dir_name with new size new_size.4 dir_name file_name: Delete the file file_name from dir_name.5 dir_name: List all files and subdirectories in dir_name in lexicographical order.6 dir_name: Print the total size of all files (including those in subdirectories) in dir_name.

Output:For operation 5, output the list of files and directories in lexicographical order.For operation 6, output the total size of all files in the specified directory (including all subdirectories).

Constraints:File and directory names will only consist of lowercase English letters and will not exceed 100 characters.File size will be between 1 and 10^6.No two files or directories in the same directory will have the same name.If a directory or file to be operated on does not exist, print an appropriate error message.

Example Input and Output:Input 1:81 root dir11 root dir22 dir1 file1 1002 dir1 file2 2005 dir13 dir1 file1 3005 dir16 rootOutput1:file1file2file1file2Total size: 500

Explanation:Initially, two directories are created under the root directory: dir1 and dir2.Two files are added to dir1, and then the contents of dir1 are displayed.The content of file1 in dir1 is updated, and the directory contents are shown again.Final the total size of all files in root is calculated.

Input 2:71 root dir11 root dir22 dir1 file1 5004 dir1 file15 dir16 rootOutput 2:file1Total size: 0

Explanation:The directory dir1 is created under the root and a file is added to it.The file is then deleted, and the directory contents are shown.The total size of all files in the root directory is calculated and shown as 0 since the file was deleted.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct File {
char name[101];
int size;
struct File* next;
} File;
typedef struct Directory {
char name[101];
struct File* files;
struct Directory* subdirs;
struct Directory* next;
} Directory;
File* createFile(const char* name, int size) {
File* newFile = (File*)malloc(sizeof(File));
strcpy(newFile->name, name);
newFile->size = size;
newFile->next = NULL;
return newFile;
}
Directory* createDirectory(const char* name) {
Directory* newDir = (Directory*)malloc(sizeof(Directory));
strcpy(newDir->name, name);
newDir->files = NULL;
newDir->subdirs = NULL;
newDir->next = NULL;
return newDir;
}
void addSubdirectory(Directory* parent, const char* newDirName) {
```

```
Directory* newDir = createDirectory(newDirName);
if (!parent->subdirs || strcmp(newDirName, parent->subdirs->name) < 0) {
newDir->next = parent->subdirs;
parent->subdirs = newDir;
return;
}
Directory* current = parent->subdirs;
while (current->next && strcmp(current->next->name, newDirName) < 0) {
current = current->next;
}
newDir->next = current->next;
current->next = newDir;
}
Directory* findDirectory(Directory* dir, const char* name) {
if (strcmp(dir->name, name) == 0) {return dir;
}
Directory* current = dir->subdirs;
while (current) {
Directory* found = findDirectory(current, name);
if (found) return found;
current = current->next;
}
return NULL;
}
void addFile(Directory* dir, const char* fileName, int size) {
File* newFile = createFile(fileName, size);
if (!dir->files || strcmp(fileName, dir->files->name) < 0) {
newFile->next = dir->files;
dir->files = newFile;
return;
}
File* current = dir->files;
while (current->next && strcmp(current->next->name, fileName) < 0) {
current = current->next;
}
newFile->next = current->next;
current->next = newFile;
}
void updateFile(Directory* dir, const char* fileName, int newSize) {
File* current = dir->files;
while (current) {
if (strcmp(current->name, fileName) == 0) {
current->size = newSize;
return;
}
current = current->next;
}
printf("Error: File not found\n");
}
void deleteFile(Directory* dir, const char* fileName) {
File* current = dir->files;
File* prev = NULL;
while (current) {
if (strcmp(current->name, fileName) == 0) {
```

```
if (prev) {
prev->next = current->next;
}
else {
dir->files = current->next;
}
free(current);
return;
}
prev = current;
current = current->next;
}
printf("Error: File not found\n");
}
void listDirectory(Directory* dir) {int printed = 0;
File* currentFile = dir->files;
while (currentFile) {
printf("%s\n", currentFile->name);
printed = 1;
currentFile = currentFile->next;
}
Directory* currentDir = dir->subdirs;
while (currentDir) {
printf("%s/\n", currentDir->name);
printed = 1;
currentDir = currentDir->next;
}
if (!printed) {
printf("Nil\n");
}
}
int calculateTotalSize(Directory* dir) {
int totalSize = 0;
File* currentFile = dir->files;
while (currentFile) {
totalSize += currentFile->size;
currentFile = currentFile->next;
}
Directory* currentSubdir = dir->subdirs;
while (currentSubdir) {
totalSize += calculateTotalSize(currentSubdir);
currentSubdir = currentSubdir->next;
}
return totalSize;
}
int main() {
int n;
if (scanf("%d", &n) != 1) return 0;
Directory* root = createDirectory("root");
for (int i = 0; i < n; i++) {
int operation;
if (scanf("%d", &operation) != 1) break;
if (operation == 1) {
char parentDir[101], newDir[101];
```

```
if (scanf("%s %s", parentDir, newDir) != 2) break;
Directory* parent = findDirectory(root, parentDir);
if (parent) addSubdirectory(parent, newDir);
}
else if (operation == 2) {
char dirName[101], fileName[101];
int fileSize;
if (scanf("%s %s %d", dirName, fileName, &fileSize) != 3) break;
Directory* dir = findDirectory(root, dirName);
if (dir) addFile(dir, fileName, fileSize);
}
else if (operation == 3) {
char dirName[101], fileName[101];
int newSize;if (scanf("%s %s %d", dirName, fileName, &newSize) != 3) break;
Directory* dir = findDirectory(root, dirName);
if (dir) updateFile(dir, fileName, newSize);
}
else if (operation == 4) {
char dirName[101], fileName[101];
if (scanf("%s %s", dirName, fileName) != 2) break;
Directory* dir = findDirectory(root, dirName);
if (dir) deleteFile(dir, fileName);
}
else if (operation == 5) {
char dirName[101];
if (scanf("%s", dirName) != 1) break;
Directory* dir = findDirectory(root, dirName);
if (dir) listDirectory(dir);
}
else if (operation == 6) {
char dirName[101];
if (scanf("%s", dirName) != 1) break;
Directory* dir = findDirectory(root, dirName);
if (dir) {
printf("Total size: %d\n", calculateTotalSize(dir));
}
}
}
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

file1
file2
file1
file2
Total size: 500

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Nil
Total size: 0

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

fileX
fileY
Total size: 400

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

fileA
Total size: 600

Compilation Status: Passed

Execution Time:

0.001s

46. Problem Statement: "Discount Based on Membership Level"A retail store offers different discounts based on the customer's membership level. You are tasked to write a program that calculates the discount based on the membership level using conditional compilation and macros.

The store offers the following discounts:

Bronze Membership: 5% discountSilver Membership: 10% discountGold Membership: 15% discountPlatinum Membership: 20% discountThe program should use conditional compilation to apply the correct discount for the membership level chosen. Use macros to define the discount percentages and apply them based on the membership level.

Input:The total bill amount (a floating-point number).The membership level of the customer (an integer representing the level: 1 for Bronze, 2 for Silver, 3 for Gold, and 4 for Platinum).

Output:The discount percentage and the final price after applying the discount.

Constraints:The total bill amount will be a positive float number, between 0.0 and 10,000.0.Membership level will be an integer between 1 and 4.The solution must be compatible with online compilers.

Example:Input:2000.03Output:Gold Membership: 15% DiscountFinal Price: 1700.00

Explanation:The total bill amount is 2000.0, and the customer has a Gold membership (level 3).The program calculates a 15% discount (Gold Membership), reducing the price by 300.0, resulting in a final price of 1700.0.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>

#define BRONZE    5
#define SILVER    10
#define GOLD      15
#define PLATINUM  20

int main() {
double bill;
int level;

scanf("%lf", &bill);
scanf("%d", &level);

int discount = 0;
const char *membershipName = "";

#if 1
if (level == 1) {
discount = BRONZE;
membershipName = "Bronze";
} else if (level == 2) {
discount = SILVER;
membershipName = "Silver";
} else if (level == 3) {
discount = GOLD;
membershipName = "Gold";
} else if (level == 4) {
discount = PLATINUM;
membershipName = "Platinum";
} else {
printf("Invalid membership level\n");
return 0;
}
#endif

double finalPrice = bill * (100 - discount) / 100.0;

printf("%s Membership: %d%% Discount\n", membershipName, discount);
printf("Final Price: %.2lf\n", finalPrice);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Bronze Membership: 5% Discount
Final Price: 950.00

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Silver Membership: 10% Discount
Final Price: 2250.00

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Gold Membership: 15% Discount

Final Price: 3400.00

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Platinum Membership: 20% Discount
Final Price: 4000.00

Compilation Status: Passed

Execution Time:

0.001s

47. Problem Statement: "Employee Salary Classification Based on Experience"In a company, employees are categorized into three different salary brackets based on their years of experience. You are required to classify employees into these brackets using conditional compilation directives.

The company decides the following categories based on experience:

Less than 5 years: Junior (Salary = 30000)Between 5 and 10 years: Mid-level (Salary = 50000)More than 10 years: Senior (Salary = 70000)However, these categories can change depending on certain conditions, such as promotions or company policies. Therefore, you will use macros and conditional compilation to classify employees into the correct salary bracket.

Input:An integer value representing the years of experience of an employee.

Output:The corresponding salary bracket based on experience.

Constraints:The experience (years) will be a non-negative integer between 0 and 50.You are not allowed to use file handling.Your solution must be compatible with online compilers.

Example:Input:7Output:Mid-levelSalary: 50000

Explanation:The employee has 7 years of experience. Therefore, they fall into the Mid-level salary bracket, which corresponds to a salary of 50000.

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#define JUNIOR_SALARY 30000
#define MID_LEVEL_SALARY 50000
#define SENIOR_SALARY 70000
#define JUNIOR_CATEGORY "Junior"
#define MID_LEVEL_CATEGORY "Mid-level"
#define SENIOR_CATEGORY "Senior"
#ifdef HIGH_PAY_POLICY#define JUNIOR_SALARY 35000
#endif
int main() {
int experience;
if (scanf("%d", &experience) != 1) {
fprintf(stderr, "Invalid input.\n");
return 1;
}
const char* category;
int salary;
if (experience < 5) {
category = JUNIOR_CATEGORY;
salary = JUNIOR_SALARY;
}
else if (experience >= 5 && experience <= 10) {
category = MID_LEVEL_CATEGORY;
salary = MID_LEVEL_SALARY;
}
else {
category = SENIOR_CATEGORY;
salary = SENIOR_SALARY;
}
printf("%s\n", category);
printf("Salary: %d\n", salary);
return 0;
}
```


Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Junior
Salary: 30000

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Mid-level
Salary: 50000

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Senior
Salary: 70000

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Junior
Salary: 30000

Compilation Status: Passed

Execution Time:

0.002s

48. Problem Statement:Insert and Display Doubly Linked List

Write a program to create a doubly linked list and display it in both forward and backward order.

Description

Insert n elements into a doubly linked list.

Display elements from head to tail and then tail to head.

Each node contains a single integer value.

Input Format

First line: Integer n (number of elements)

Next line: n integers representing the list elements

Output Format

First line: Forward traversal: "Forward: <elements>"

Second line: Backward traversal: "Backward: <elements>"

Sample Input510 20 30 40 50

Sample OutputForward: 10 20 30 40 50Backward: 50 40 30 20 10

Completion Status: Completed

Concepts Included:

GU 28 3rd Sem DSA Pre-Midterm

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node* prev;
struct Node* next;
};
struct Node* createNode(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->prev = NULL;
newNode->next = NULL;
return newNode;
}
int main() {
int n;scanf("%d", &n);
if (n <= 0)
return 0;
struct Node* head = NULL;
struct Node* tail = NULL;
for (int i = 0; i < n; i++) {
int val;
scanf("%d", &val);
struct Node* newNode = createNode(val);
if (head == NULL) {
head = newNode;
tail = newNode;
}
else {
tail->next = newNode;
newNode->prev = tail;
tail = newNode;
}
}
printf("Forward: ");
struct Node* temp = head;
while (temp != NULL) {
printf("%d", temp->data);
if (temp->next != NULL)
printf(" ");
temp = temp->next;
}
printf("\n");
printf("Backward: ");
temp = tail;
while (temp != NULL) {
printf("%d", temp->data);
```

```
if (temp->prev != NULL)
printf(" ");
temp = temp->prev;
}
printf("\n");
temp = head;
while (temp != NULL) {
struct Node* nextNode = temp->next;
free(temp);
temp = nextNode;
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Forward: 1 2 3
Backward: 3 2 1

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Forward: 5 10 15 20
Backward: 20 15 10 5

Compilation Status: Passed

Execution Time:

0.001s

TestCase3:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Forward: 7 14 21 28 35 42
Backward: 42 35 28 21 14 7

Compilation Status: Passed

Execution Time:

0.001s

TestCase4:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Forward: 10 20 20 30 30 40
Backward: 40 30 30 20 20 10

Compilation Status: Passed

Execution Time:

0.001s