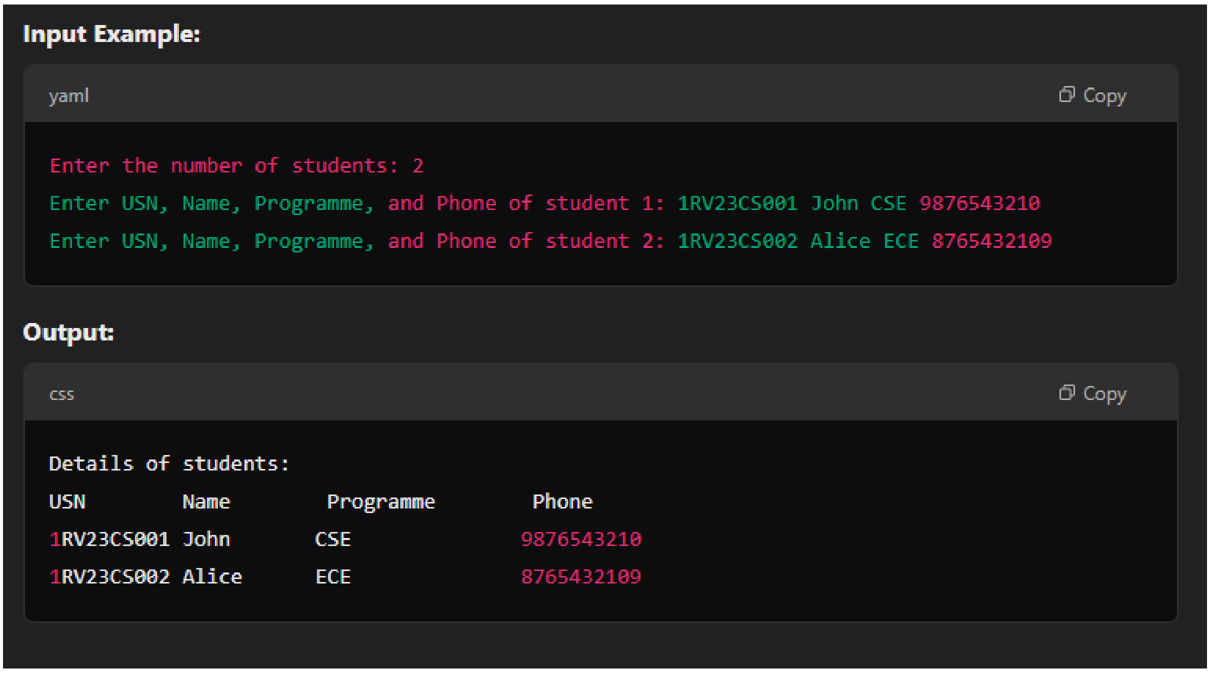
1.ACreate a Java class called Student with the following details as variables within it.
(i) USN
(ii) Name
(iii) Programme
(iv) Phone
Write a CPP program to create nStudent objects and print the USN, Name, Programme, and Phoneof these objects with suitable headings.

PROGRAM:

```
#include <iostream>
#include <string>
using namespace std;
class Student {
 public:
  string USN;
  string Name;
  string Programme;
  string Phone;
};
int main() {
  int n;
  cout << "Enter the number of students: ";
  cin >> n;
  Student *s = new Student[n];
  for (int i = 0; i < n; i++) {
    cout << "Enter USN, Name, Programme, and Phone of student " << i + 1 << ": ";
    cin >> s[i].USN >> s[i].Name >> s[i].Programme >> s[i].Phone;
  }
  cout << "\nDetails of students: \n";
  cout << "USN\t\tName\t\tProgramme\t\tPhone\n";
  for (int i = 0; i < n; i++) {
    cout << s[i].USN << "\t\t" << s[i].Name << "\t\t" << s[i].Programme << "\t\t" << s[i].Phone << endl;
  }
  return 0;
}
```



Input Example:

```
Enter the number of students: 2
Enter USN, Name, Programme, and Phone of student 1: 1RV23CS001 John CSE 9876543210
Enter USN, Name, Programme, and Phone of student 2: 1RV23CS002 Alice ECE 8765432109
```

Output:

```
Details of students:
USN        Name        Programme        Phone
1RV23CS001 John       CSE              9876543210
1RV23CS002 Alice      ECE              8765432109
```

1.B Write a CPP program to implement the Stack using arrays. Write Push(), Pop(), and Display() methods to demonstrate its working

PROGRAM:

```
#include <iostream>
#include <climits>

using namespace std;

const int MAX = 1000;

class Stack {
  private:
    int top;
    int a[MAX];
  public:
    Stack() { top = -1; }

    bool push(int x) {
      if (top >= (MAX-1)) {
        cout << "Stack Overflow";
        return false;
      } else {
        a[++top] = x;
        return true;
      }
    }

    int pop() {
      if (top < 0) {
        cout << "Stack Underflow";
        return INT_MIN;
      } else {
        int x = a[top--];
```

```
                return x;
            }
        }

        int peek() {
            if (top < 0) {
                cout << "Stack is Empty";
                return INT_MIN;
            } else {
                int x = a[top];
                return x;
            }
        }

        bool isEmpty() {
            return (top < 0);
        }

        void display() {
            if (top < 0) {
                cout << "Stack is Empty";
                return;
            } else {
                cout << "Stack elements are: ";
                for (int i = top; i >= 0; i--)
                    cout << a[i] << " ";
                cout << endl;
            }
        }
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.display();
    cout << "Top element is: " << s.peek() << endl;
    s.pop();
    s.display();
    return 0;
}
```

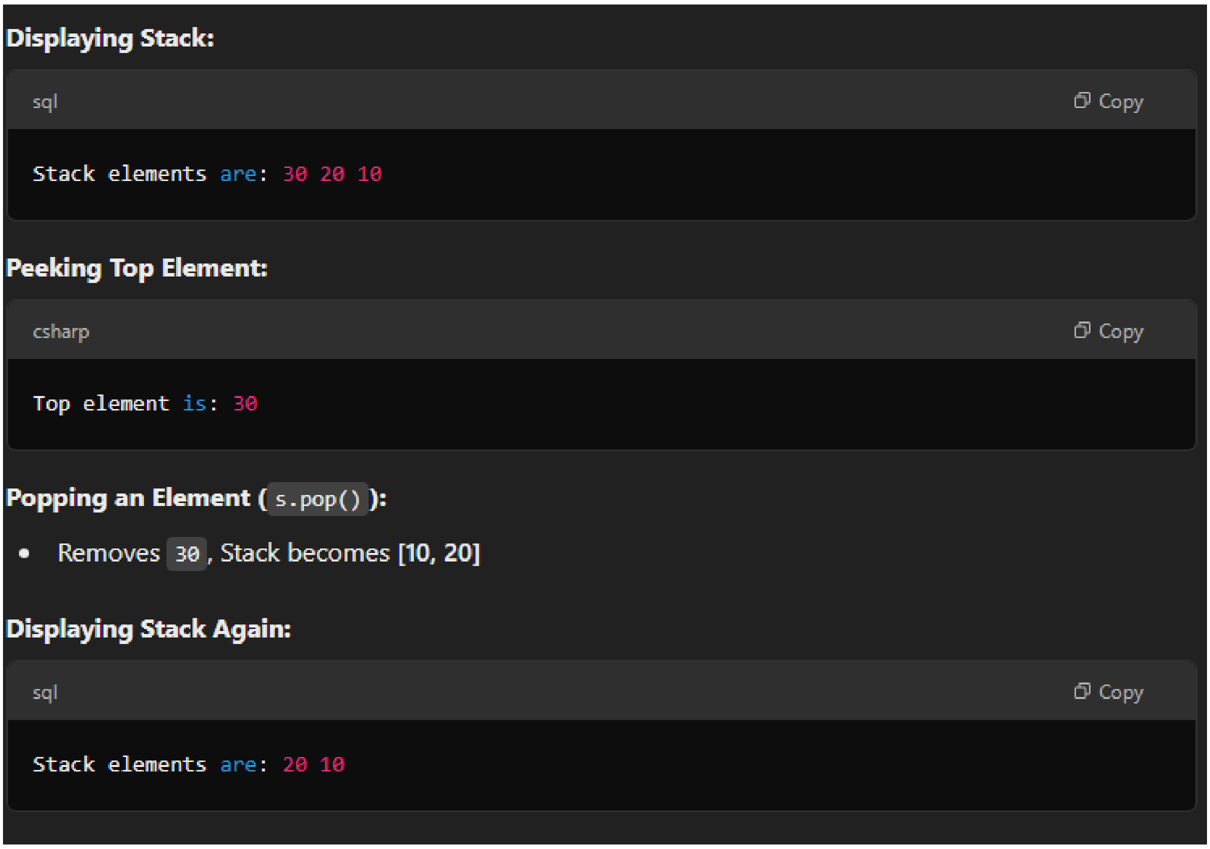
The given **Stack** program implements basic stack operations using an **array** and supports:

- `push(x)` → Adds `x` to the stack
- `pop()` → Removes the top element
- `peek()` → Returns the top element without removing it
- `isEmpty()` → Checks if the stack is empty
- `display()` → Prints all elements from top to bottom

Step-by-step Execution

Pushing Elements:

- `s.push(10);` → Stack: **[10]**
- `s.push(20);` → Stack: **[10, 20]**
- `s.push(30);` → Stack: **[10, 20, 30]**



Displaying Stack:

```
Stack elements are: 30 20 10
```

Peeking Top Element:

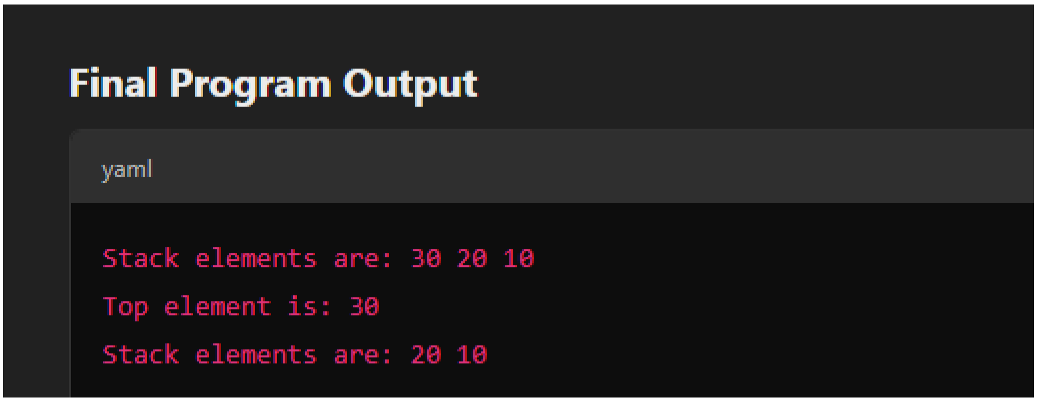
```
Top element is: 30
```

Popping an Element (`s.pop()`):

- Removes `30`, Stack becomes [10, 20]

Displaying Stack Again:

```
Stack elements are: 20 10
```



Final Program Output

```
Stack elements are: 30 20 10
Top element is: 30
Stack elements are: 20 10
```

2. A Design a superclass called Staff with details as StaffId, Name, Phone, Salary. Extend this class by writing three subclasses namely Teaching (domain, publications), Technical (skills), and Contract (period). Write a CPP program to read and display at least 3 staff objects of all three categories.

PROGRAM:

```
#include <iostream>
#include <string>

using namespace std;

class Staff {
  protected:
    int StaffId;
    string Name;
    string Phone;
    float Salary;
  public:
    Staff() {}

    Staff(int id, string name, string phone, float salary) {
      StaffId = id;
      Name = name;
      Phone = phone;
      Salary = salary;
    }

    void display() {
      cout << "Staff ID: " << StaffId << endl;
      cout << "Name: " << Name << endl;
      cout << "Phone: " << Phone << endl;
      cout << "Salary: " << Salary << endl;
    }
};

class Teaching : public Staff {
  private:
    string domain;
    int publications;
  public:
    Teaching() {}

    Teaching(int id, string name, string phone, float salary, string d, int pub)
      : Staff(id, name, phone, salary) {
        domain = d;
        publications = pub;
      }

    void display() {
      Staff::display();
      cout << "Domain: " << domain << endl;
      cout << "Publications: " << publications << endl;
    }
};

class Technical : public Staff {
  private:
        string skills;
    public:
        Technical() {}

        Technical(int id, string name, string phone, float salary, string s)
            : Staff(id, name, phone, salary) {
                skills = s;
            }

        void display() {
            Staff::display();
            cout << "Skills: " << skills << endl;
        }
};

class Contract : public Staff {
    private:
        int period;
    public:
        Contract() {}

        Contract(int id, string name, string phone, float salary, int p)
            : Staff(id, name, phone, salary) {
                period = p;
            }

        void display() {
            Staff::display();
            cout << "Period: " << period << endl;
        }
};

int main() {
    Teaching t1(1, "John Doe", "555-555-5555", 60000, "Computer Science", 20);
    Technical t2(2, "Jane Doe", "555-555-5556", 55000, "C++, Java");
    Contract t3(3, "Jim Doe", "555-555-5557", 50000, 6);

    cout << "Teaching Staff 1" << endl;
    t1.display();
    cout << endl;

    cout << "Technical Staff 2" << endl;
    t2.display();
    cout << endl;

    cout << "Contract Staff 3" << endl;
    t3.display();
    cout << endl;

    return 0;
}
```

Program Execution and Output Explanation

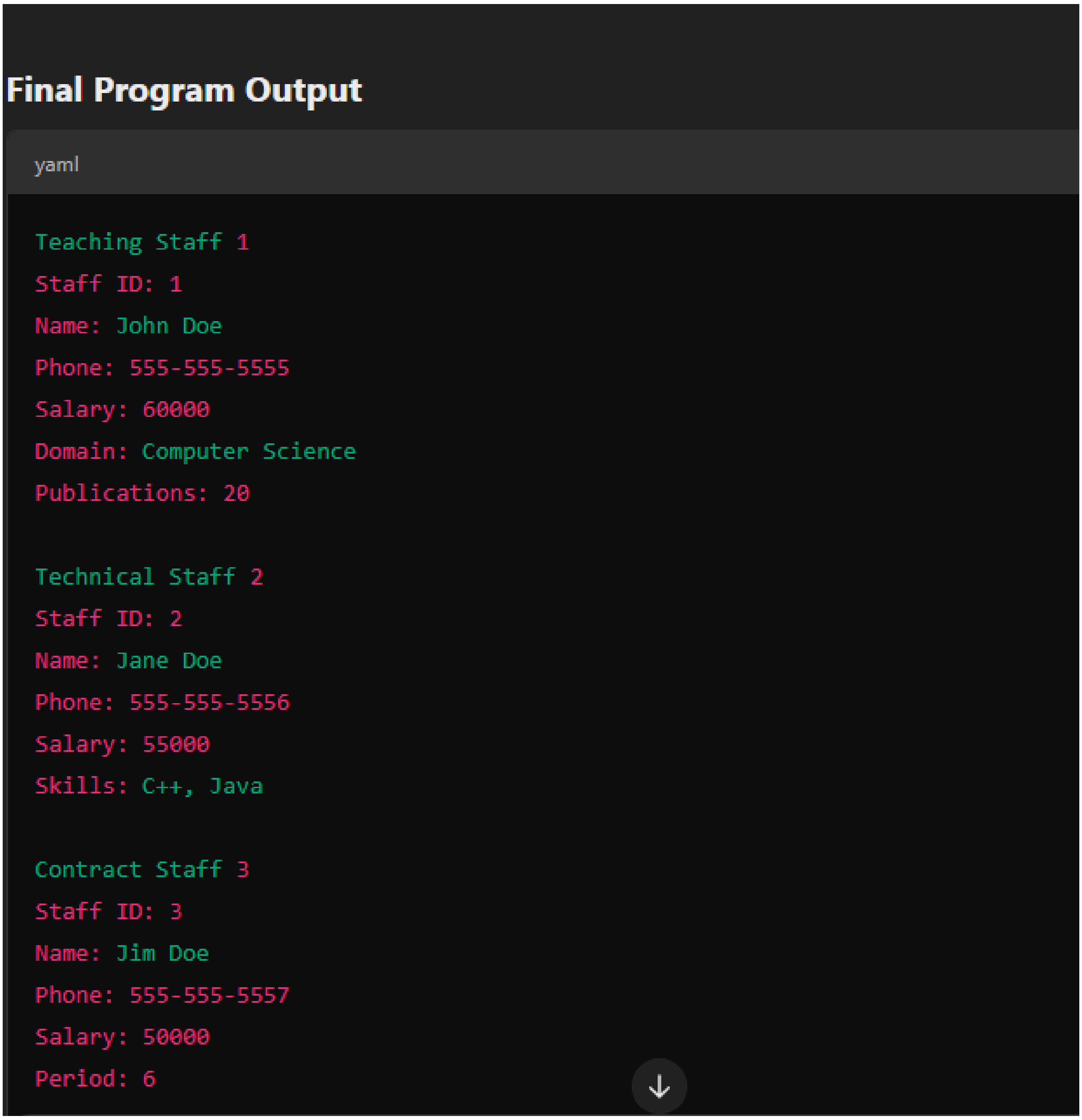
This C++ program implements an **inheritance hierarchy** for different types of **staff members**:

1. **`Staff`** (Base Class) → Common attributes: `StaffId`, `Name`, `Phone`, `Salary`
2. **`Teaching`** (Derived Class) → Additional attributes: `domain`, `publications`
3. **`Technical`** (Derived Class) → Additional attribute: `skills`
4. **`Contract`** (Derived Class) → Additional attribute: `period`

Each class has a **display method** to print details.

Step-by-Step Execution

1. **Object Creation**
   - `Teaching t1(1, "John Doe", "555-555-5555", 60000, "Computer Science", 20);`
   - `Technical t2(2, "Jane Doe", "555-555-5556", 55000, "C++, Java");`
   - `Contract t3(3, "Jim Doe", "555-555-5557", 50000, 6);`
2. **Displaying Data**



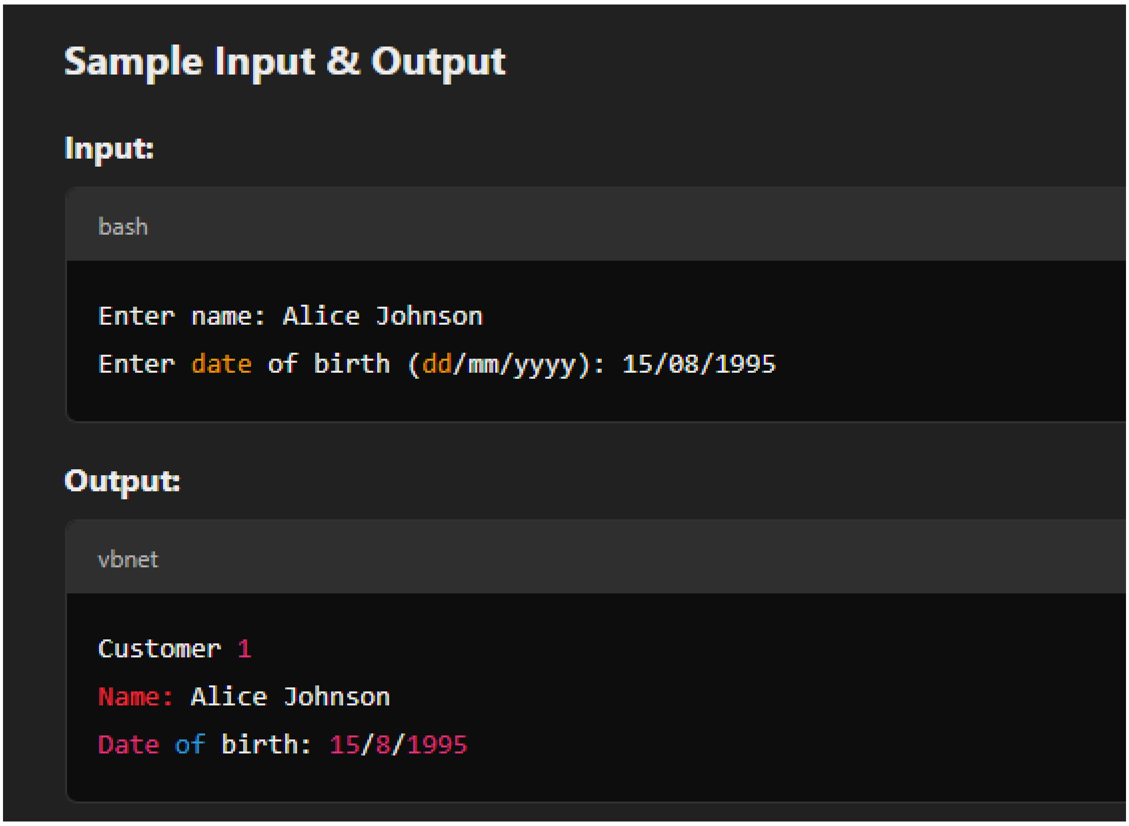
Final Program Output

```
Teaching Staff 1
Staff ID: 1
Name: John Doe
Phone: 555-555-5555
Salary: 60000
Domain: Computer Science
Publications: 20

Technical Staff 2
Staff ID: 2
Name: Jane Doe
Phone: 555-555-5556
Salary: 55000
Skills: C++, Java

Contract Staff 3
Staff ID: 3
Name: Jim Doe
Phone: 555-555-5557
Salary: 50000
Period: 6
```

2.B Write a CPP class called Customer to store their name and date_of_birth. The date_of_birth format should be dd/mm/yyyy. Write methods to read customer data as and display as using StringTokenizer class considering the delimiter character as “ /” .

PROGRAM:

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

class Customer {
  private:
    string name;
    int day, month, year;
  public:
        Customer() {}

        Customer(string n, int d, int m, int y) {
            name = n;
            day = d;
            month = m;
            year = y;
        }

        void readData() {
            cout << "Enter name: ";
            getline(cin, name);
            cout << "Enter date of birth (dd/mm/yyyy): ";
            string dob;
            getline(cin, dob);

            stringstream ss(dob);
            char delimiter;
            ss >> day >> delimiter >> month >> delimiter >> year;
        }

        void displayData() {
            cout << "Name: " << name << endl;
            cout << "Date of birth: " << day << "/" << month << "/" << year << endl;
        }
};

int main() {
    Customer c1;
    c1.readData();
    cout << endl;

    cout << "Customer 1" << endl;
    c1.displayData();

    return 0;
}
```



Sample Input & Output

Input:

bash

```
Enter name: Alice Johnson
Enter date of birth (dd/mm/yyyy): 15/08/1995
```

Output:

vbnet

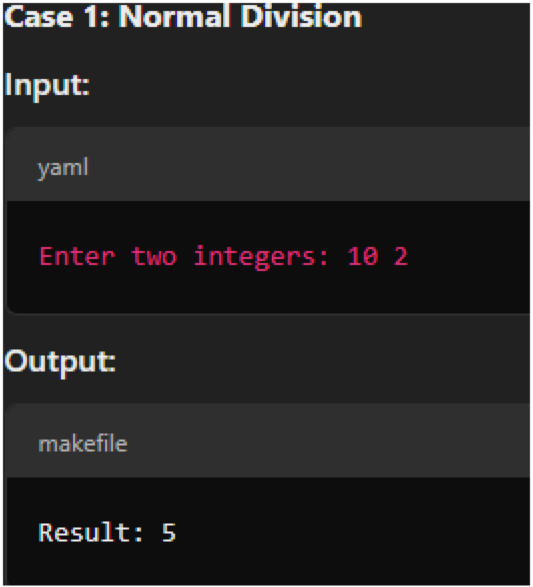
```
Customer 1
Name: Alice Johnson
Date of birth: 15/8/1995
```

3.A Write a CPP program to read two integers a andb. Compute a/b and print, when b is not zero. Raise an exception when b is equal to zero.

PROGRAM:

```
#include <iostream>
#include <exception>

using namespace std;

class DivideByZeroException : public exception {
  public:
    const char* what() const throw() {
      return "Division by zero exception";
    }
};

int main() {
  int a, b;
  cout << "Enter two integers: ";
  cin >> a >> b;

  try {
    if (b == 0) {
      throw DivideByZeroException();
    } else {
      cout << "Result: " << (double)a/b << endl;
    }
  } catch (DivideByZeroException& e) {
    cout << e.what() << endl;
  }
```

```
    return 0;
}
```



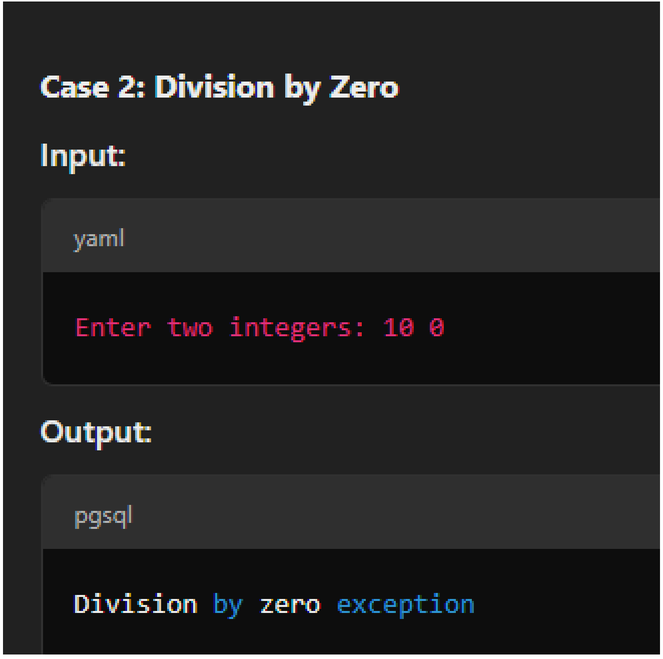
Case 1: Normal Division

Input:

```
Enter two integers: 10 2
```

Output:

```
Result: 5
```



Case 2: Division by Zero

Input:

```
Enter two integers: 10 0
```

Output:

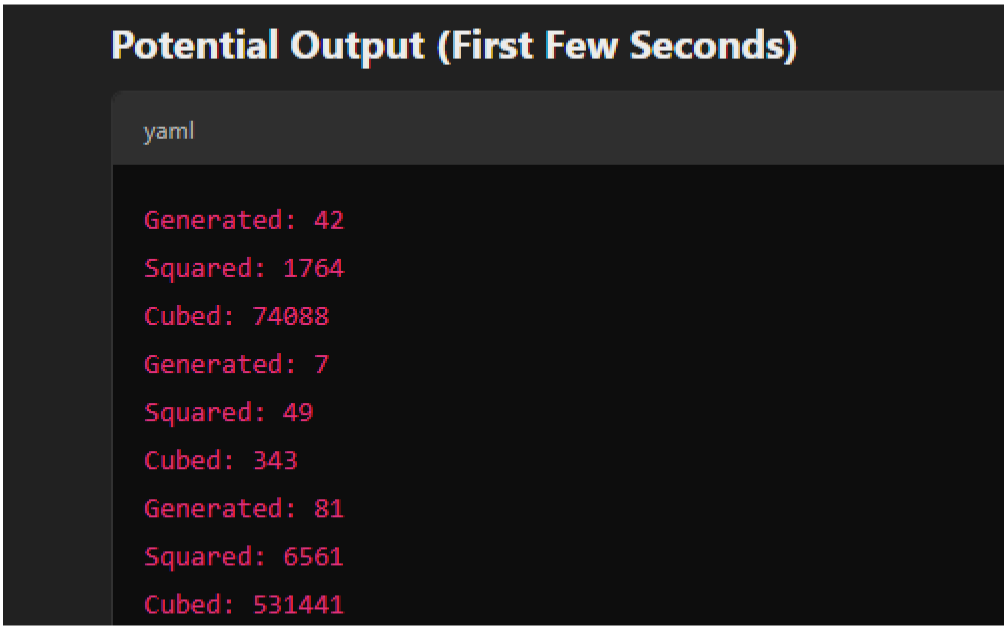
```
Division by zero exception
```

3.B Write a CPP program that implements a multi-thread application that has three threads. First thread generates a random integer for every 1 second; second thread computes the square of the number and prints; third thread will print the value of cube of the number.

PROGRAM:

```
#include <iostream>
#include <thread>
#include <chrono>
#include <cstdlib>
#include <ctime>

using namespace std;

int number;
```

```
void generateNumber() {
    while (true) {
        number = rand() % 100;
        cout << "Generated: " << number << endl;
        this_thread::sleep_for(chrono::seconds(1));
    }
}

void squareNumber() {
    while (true) {
        int square = number * number;
        cout << "Squared: " << square << endl;
        this_thread::sleep_for(chrono::seconds(1));
    }
}

void cubeNumber() {
    while (true) {
        int cube = number * number * number;
        cout << "Cubed: " << cube << endl;
        this_thread::sleep_for(chrono::seconds(1));
    }
}

int main() {
    srand(time(0));

    thread t1(generateNumber);
    thread t2(squareNumber);
    thread t3(cubeNumber);

    t1.join();
    t2.join();
    t3.join();

    return 0;
}
```



Potential Output (First Few Seconds)

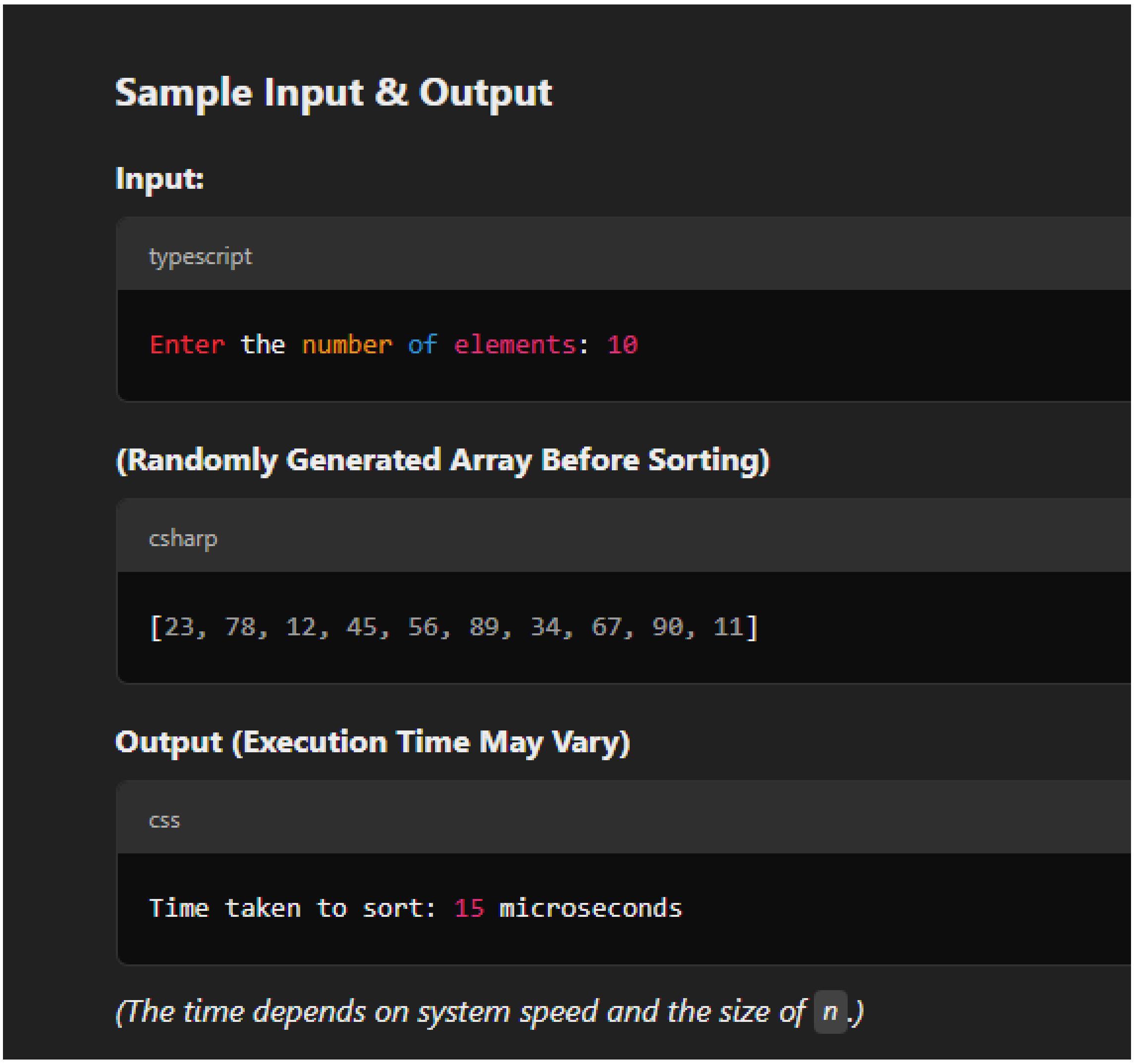
```
Generated: 42
Squared: 1764
Cubed: 74088
Generated: 7
Squared: 49
Cubed: 343
Generated: 81
Squared: 6561
Cubed: 531441
```

4. Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using CPP how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.

PROGRAM:

```
#include <iostream>
#include <chrono>
#include <cstdlib>
#include <ctime>

using namespace std;

void quickSort(int arr[], int left, int right) {
   int i = left, j = right;
   int pivot = arr[(left + right) / 2];

   while (i <= j) {
      while (arr[i] < pivot) {
         i++;
      }
      while (arr[j] > pivot) {
         j--;
      }
      if (i <= j) {
         swap(arr[i], arr[j]);
         i++;
         j--;
      }
   }

   if (left < j) {
      quickSort(arr, left, j);
   }
```

```
    if (i < right) {
        quickSort(arr, i, right);
    }
}

int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];
    srand(time(0));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100;
    }
    auto start = chrono::high_resolution_clock::now();

    quickSort(arr, 0, n - 1);
    auto end = chrono::high_resolution_clock::now();
    auto elapsed = chrono::duration_cast<chrono::microseconds>(end - start);
    cout << "Time taken to sort: " << elapsed.count() << " microseconds" << endl;

    return 0;
}
```



Sample Input & Output

Input:

typescript

```
Enter the number of elements: 10
```

(Randomly Generated Array Before Sorting)

csharp

```
[23, 78, 12, 45, 56, 89, 34, 67, 90, 11]
```

Output (Execution Time May Vary)

css

```
Time taken to sort: 15 microseconds
```

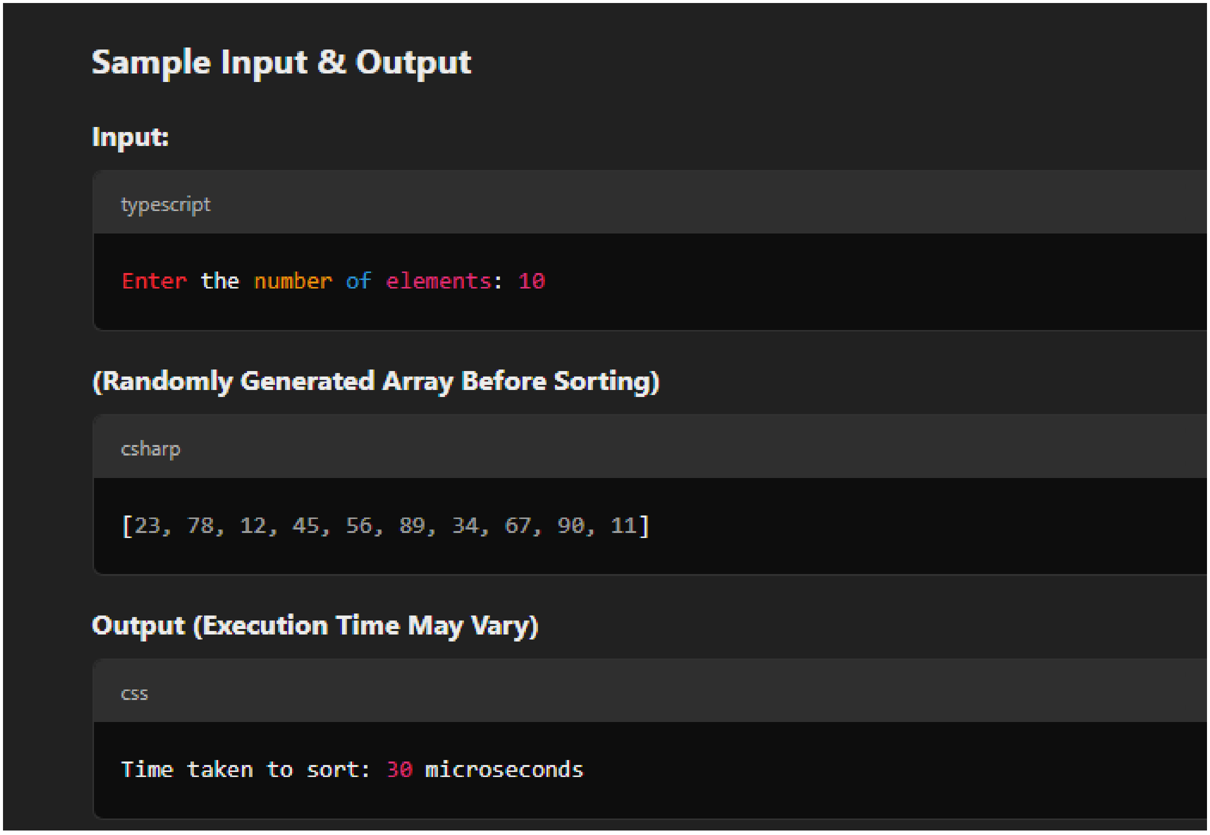
(The time depends on system speed and the size of `n`.)

5. Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using CPP how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.

PROGRAM:

```
#include <iostream>
#include <chrono>
#include <cstdlib>
#include <ctime>
using namespace std;
void merge(int arr[], int l, int m, int r) {
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;
   int L[n1], R[n2];
   for (i = 0; i < n1; i++) {
      L[i] = arr[l + i];
   }
   for (j = 0; j < n2; j++) {
      R[j] = arr[m + 1 + j];
```

```
    }
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];
    srand(time(0));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100;
    }

    auto start = chrono::high_resolution_clock::now();
    mergeSort(arr, 0, n - 1);

    auto end = chrono::high_resolution_clock::now();
    auto elapsed = chrono::duration_cast<chrono::microseconds>(end - start);
    cout << "Time taken to sort: " << elapsed.count() << " microseconds" << endl;

    return 0;
}
```



Sample Input & Output

Input:

```
Enter the number of elements: 10
```

(Randomly Generated Array Before Sorting)

```
[23, 78, 12, 45, 56, 89, 34, 67, 90, 11]
```

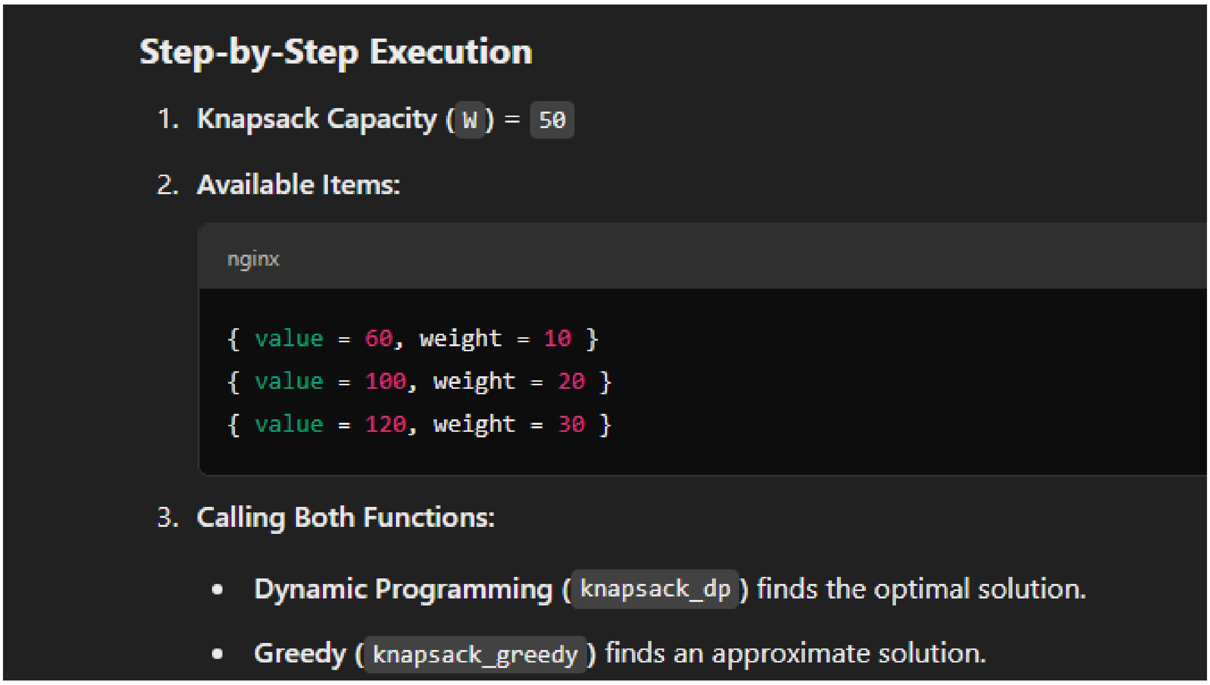
Output (Execution Time May Vary)

```
Time taken to sort: 30 microseconds
```

6.Implement in CPP, the 0/1 Knapsack problem using (a) Dynamic Programming method (b) Greedy method.

PROGRAM:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Define the structure for an item in the knapsack
struct Item {
    int value;
    int weight;
};

// Dynamic Programming solution
int knapsack_dp(int W, vector<Item> items) {
    int n = items.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1));
    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= W; w++) {
            if (items[i - 1].weight > w) {
                dp[i][w] = dp[i - 1][w];
            } else {
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - items[i - 1].weight] + items[i - 1].value);
            }
        }
    }
    return dp[n][W];
}

// Greedy solution
int knapsack_greedy(int W, vector<Item> items) {
    sort(items.begin(), items.end(), [](Item a, Item b) { return a.value > b.value; });
    int n = items.size();
    int weight = 0;
    int value = 0;
    for (int i = 0; i < n; i++) {
        if (weight + items[i].weight <= W) {
            weight += items[i].weight;
            value += items[i].value;
        } else {
            break;
        }
    }
    return value;
}

int main() {
    int W = 50;
    vector<Item> items = {{60, 10}, {100, 20}, {120, 30}};

    cout << "Using Dynamic Programming: " << knapsack_dp(W, items) << endl;
    cout << "Using Greedy: " << knapsack_greedy(W, items) << endl;
    return 0;
}
```

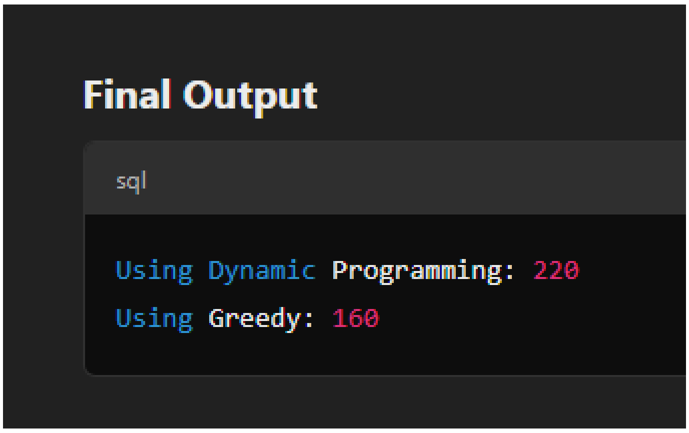


Step-by-Step Execution

1. **Knapsack Capacity (`W`) =** `50`
2. **Available Items:**

```nginx
{ value = 60, weight = 10 }
{ value = 100, weight = 20 }
{ value = 120, weight = 30 }
```

3. **Calling Both Functions:**
   - **Dynamic Programming (`knapsack_dp`)** finds the optimal solution.
   - **Greedy (`knapsack_greedy`)** finds an approximate solution.



Final Output

```sql
Using Dynamic Programming: 220
Using Greedy: 160
```

7. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in CPP.

PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 100;  // Maximum number of vertices in the graph

int graph[MAX][MAX];  // Adjacency matrix representation of the graph
int dist[MAX];        // Array to store the shortest distance from the source vertex
bool sptSet[MAX];     // Boolean array to keep track of vertices included in shortest path tree

// A function to find the vertex with minimum distance value, from the set of vertices not yet
included in shortest path tree
int minDistance(int V)
{
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)
```

```
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A function that implements Dijkstra's algorithm
void dijkstra(int src, int V)
{
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V-1; count++)
    {
        int u = minDistance(V);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u]+graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];

    cout << "Vertex \t Distance from Source" << endl;
    for (int i = 0; i < V; i++)
        cout << i << "\t\t " << dist[i] << endl;
}
int main()
{
    int V, src;
    cout << "Enter the number of vertices: ";
    cin >> V;
    cout << "Enter the adjacency matrix representation of the graph: " << endl;
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            cin >> graph[i][j];
    cout << "Enter the source vertex: ";
    cin >> src;
    dijkstra(src, V);

    return 0;
}
```

Bug in Code & Fix

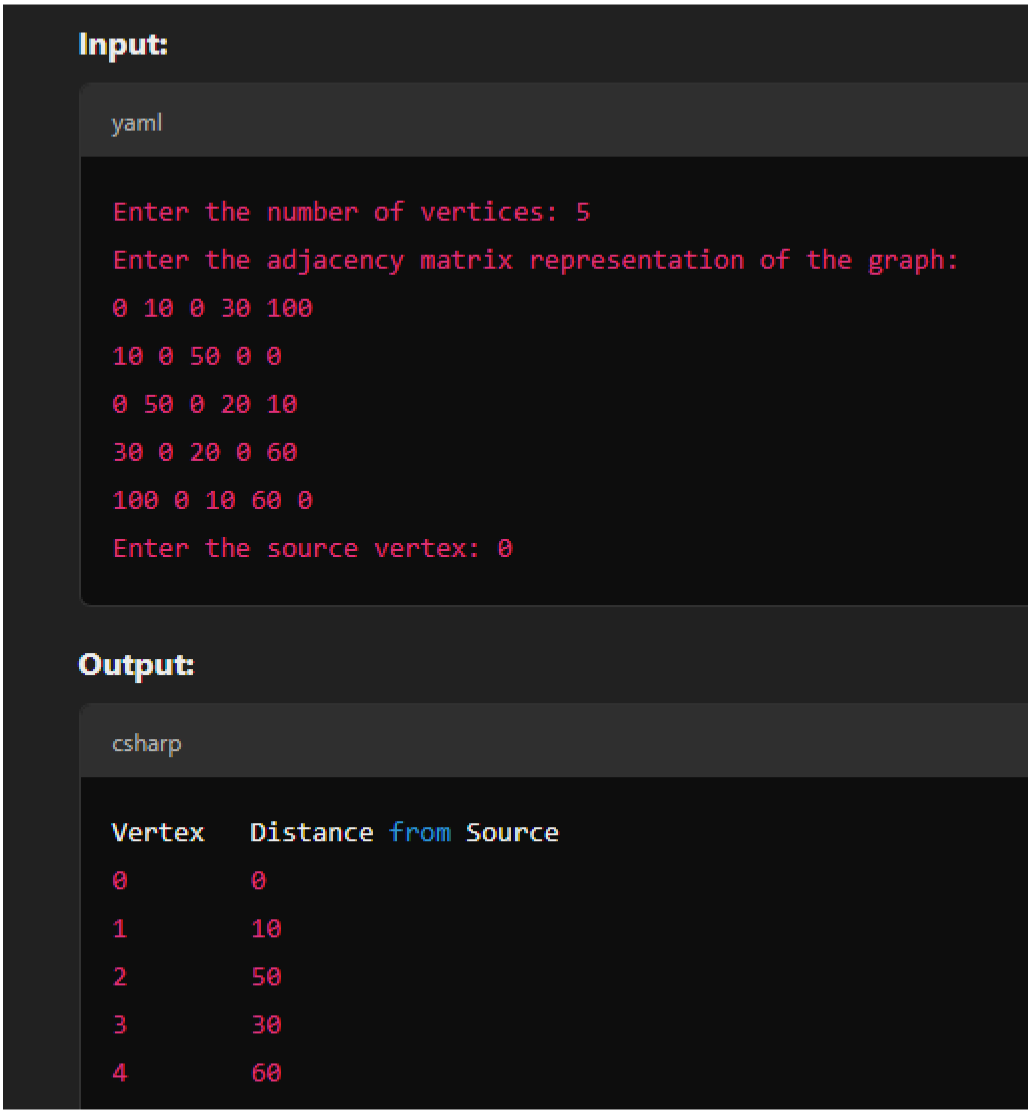
Issue:

The closing brace `}` for the `for` loop in `dijkstra()` is misplaced. The final output printing is inside the loop instead of after it.

Fix:

Move the output section **outside** the loop:

```
void dijkstra(int src, int V)
{
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

    for (int count = 0; count < V-1; count++)
    {
        int u = minDistance(V);
        sptSet[u] = true;

        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u]+graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // Moved outside the loop
    cout << "Vertex \t Distance from Source" << endl;
    for (int i = 0; i < V; i++)
        cout << i << "\t\t " << dist[i] << endl;
}
```



Input:

yaml

```
Enter the number of vertices: 5
Enter the adjacency matrix representation of the graph:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0
Enter the source vertex: 0
```

Output:

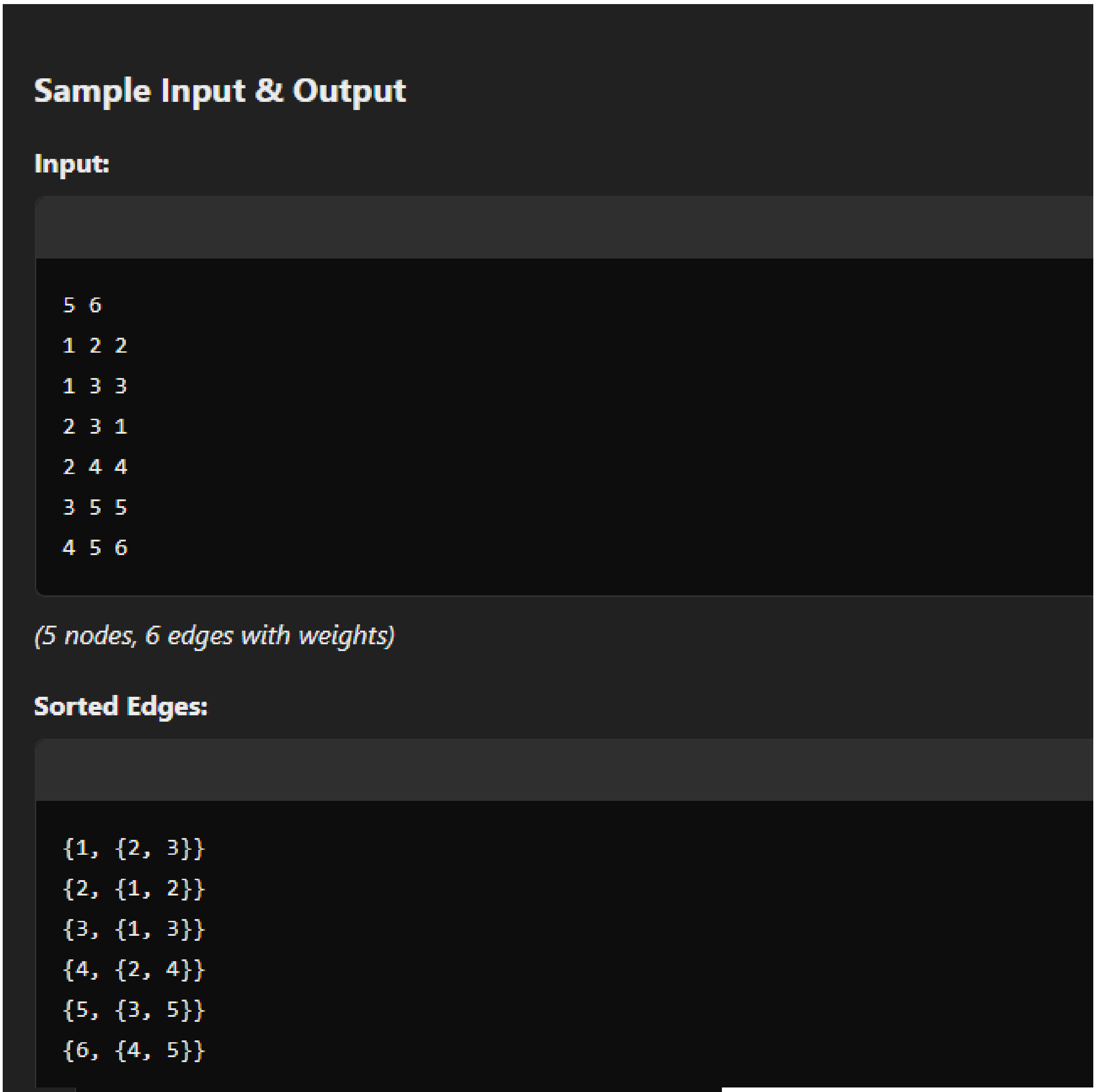
csharp

```
Vertex   Distance from Source
0        0
1        10
2        50
3        30
4        60
```

8. Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal'salgorithm. Use Union-Find algorithms in your program

PROGRAM:

```
#include<bits/stdc++.h>
using namespace std;

const int MAX = 1e4 + 5;
int id[MAX], nodes, edges;
pair <long long, pair<int, int> > p[MAX];

void initialize()
{
   for(int i = 0;i < MAX;++i)
      id[i] = i;
}

int root(int x)
{
   while(id[x] != x)
   {
      id[x] = id[id[x]];
      x = id[x];
   }
   return x;
}

void union1(int x, int y)
{
   int p = root(x);
   int q = root(y);
   id[p] = id[q];
}

long long kruskal(pair<long long, pair<int, int> > p[])
{
   int x, y;
   long long cost, minimumCost = 0;
   for(int i = 0;i < edges;++i)
   {
      x = p[i].second.first;
      y = p[i].second.second;
      cost = p[i].first;
      if(root(x) != root(y))
      {
         minimumCost += cost;
         union1(x, y);
      }
   }
   return minimumCost;
}

int main()
{
   int x, y;
   long long weight, cost, minimumCost;
   initialize();
   cin >> nodes >> edges;
   for(int i = 0;i < edges;++i)
   {
      cin >> x >> y >> weight;
      p[i] = make_pair(weight, make_pair(x, y));
   }
   sort(p, p + edges);
   minimumCost = kruskal(p);
   cout << minimumCost << endl;
   return 0;
}
```



Sample Input & Output

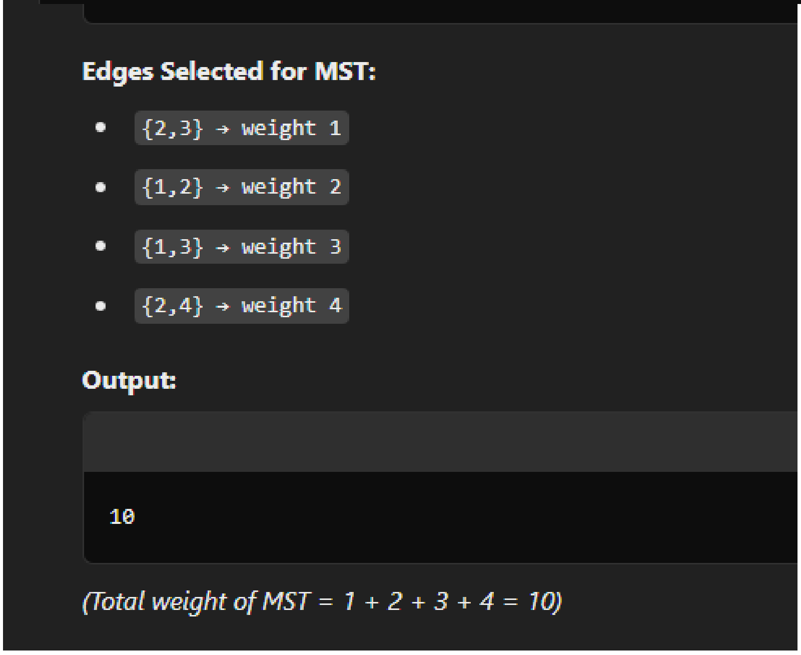
Input:

```
5 6
1 2 2
1 3 3
2 3 1
2 4 4
3 5 5
4 5 6
```

(5 nodes, 6 edges with weights)

Sorted Edges:

```
{1, {2, 3}}
{2, {1, 2}}
{3, {1, 3}}
{4, {2, 4}}
{5, {3, 5}}
{6, {4, 5}}
```



Edges Selected for MST:

- `{2,3} → weight 1`
- `{1,2} → weight 2`
- `{1,3} → weight 3`
- `{2,4} → weight 4`

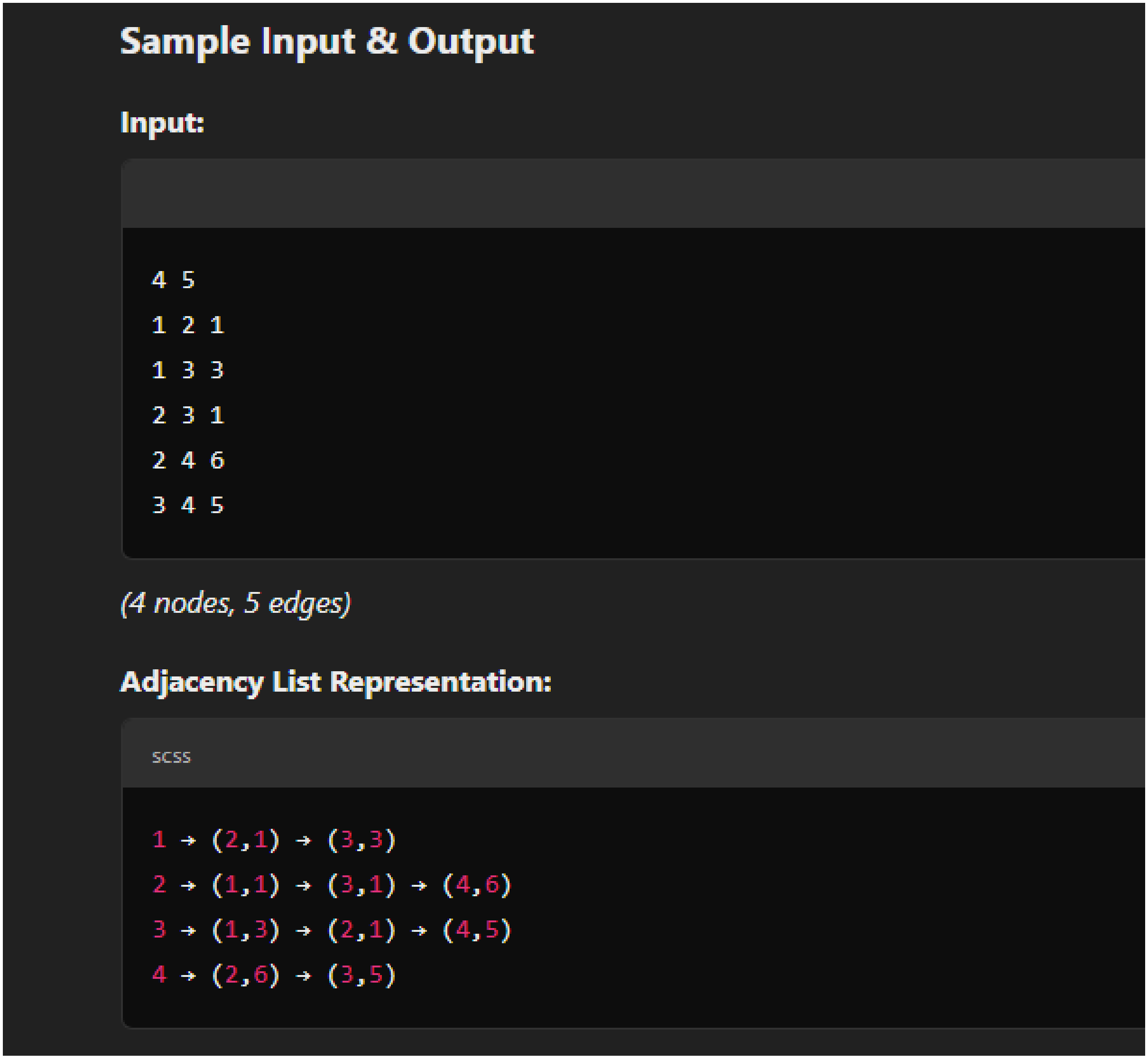
Output:

```
10
```

(Total weight of MST = 1 + 2 + 3 + 4 = 10)

9.Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.
PROGRAM:

```
#include <iostream>
#include <cstring>
#include <vector>
#include <queue>
using namespace std;

const int N = 110, M = 10010, INF = 0x3f3f3f3f;
int n, m, h[N], e[M], ne[M], w[M], idx, dist[N];
bool st[N];

void add(int a, int b, int c)
{
   e[idx] = b, ne[idx] = h[a], w[idx] = c, h[a] = idx ++ ;
}

int prim()
{
   memset(dist, 0x3f, sizeof dist);
   memset(st, false, sizeof st);
   priority_queue<pair<int, int> > heap;
   heap.push({0, 1});
   dist[1] = 0;
   int res = 0;
   while (heap.size())
   {
      int ver = heap.top().second, distance = -heap.top().first;
      heap.pop();
      if (st[ver]) continue;
      st[ver] = true, res += distance;
      for (int i = h[ver]; ~i; i = ne[i])
      {
         int j = e[i];
         if (!st[j] && dist[j] > w[i])
         {
            dist[j] = w[i];
            heap.push({-dist[j], j});
         }
      }
   }
   return res;
}

int main()
{
   memset(h, -1, sizeof h);
   cin >> n >> m;
   while (m -- )
   {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c), add(b, a, c);
    }
    cout << prim() << endl;
    return 0;
}
```



Sample Input & Output

Input:

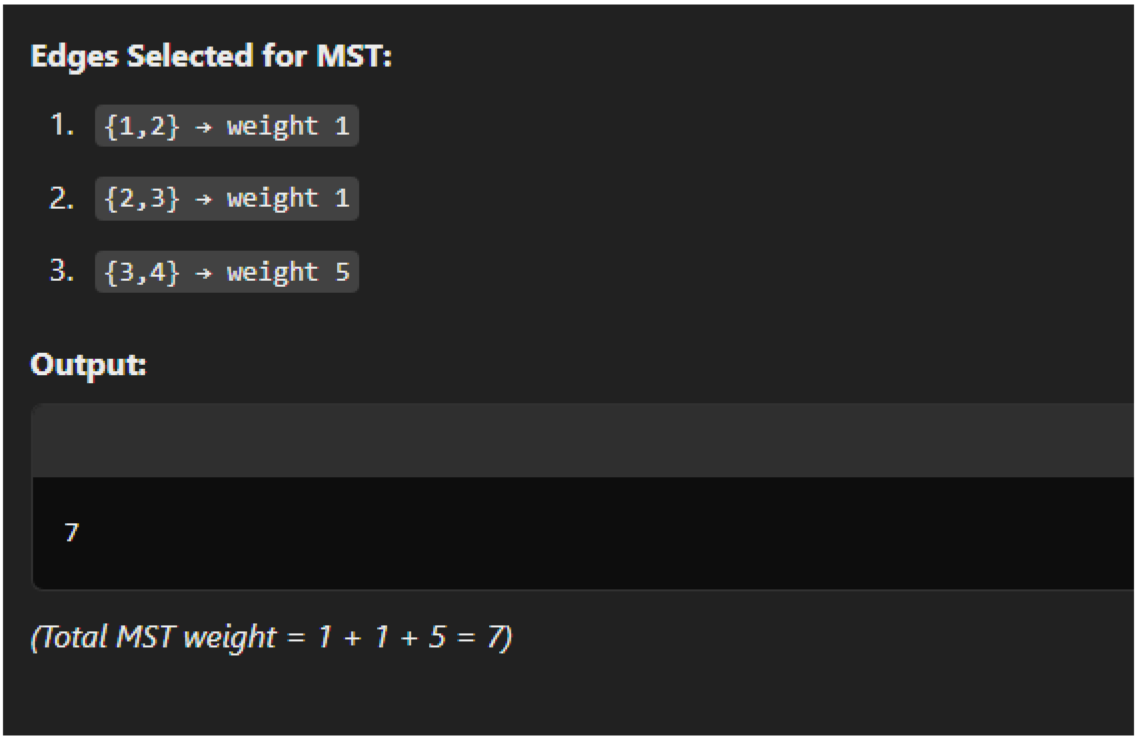
```
4 5
1 2 1
1 3 3
2 3 1
2 4 6
3 4 5
```

(4 nodes, 5 edges)

Adjacency List Representation:

scss

```
1 → (2,1) → (3,3)
2 → (1,1) → (3,1) → (4,6)
3 → (1,3) → (2,1) → (4,5)
4 → (2,6) → (3,5)
```



Edges Selected for MST:

1. `{1,2} → weight 1`
2. `{2,3} → weight 1`
3. `{3,4} → weight 5`

Output:

```
7
```

(Total MST weight = 1 + 1 + 5 = 7)

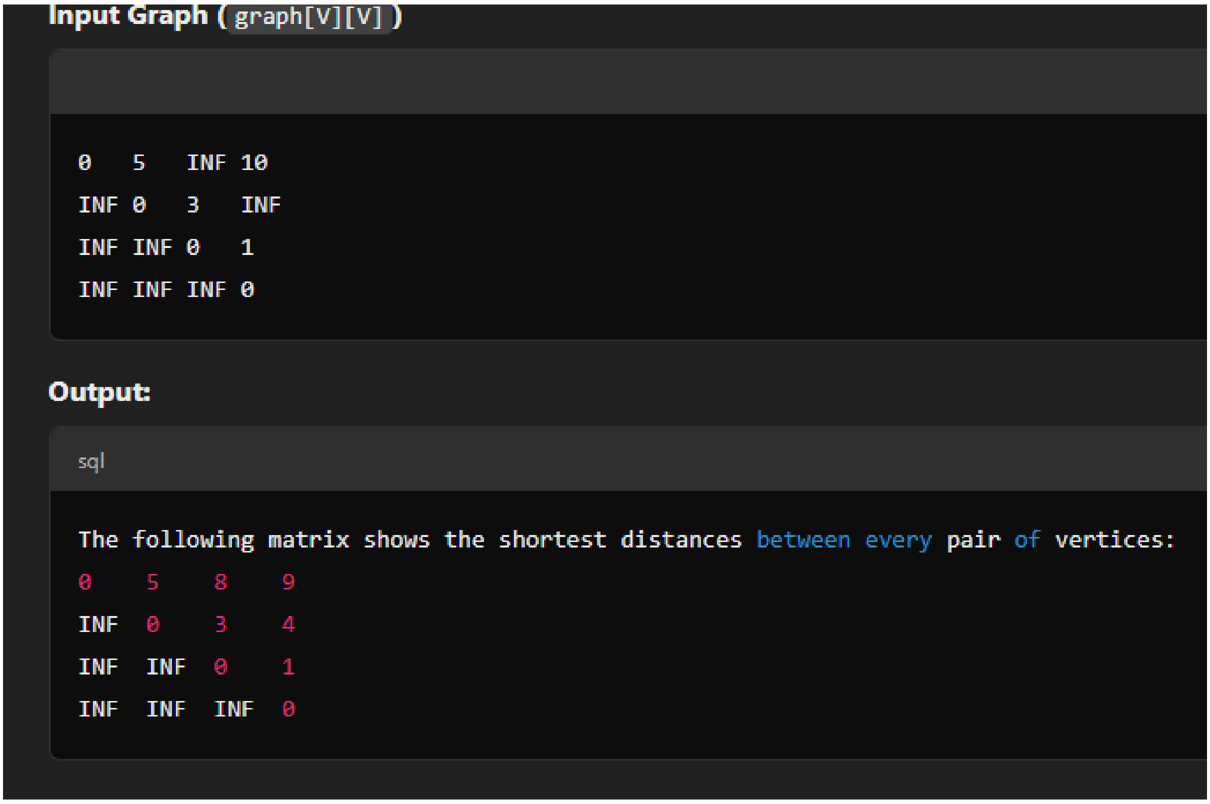
10.A.Write CPP programs to Implement All-Pairs Shortest Paths problem using Floyd's algorithm.
PROGRAM:

```
#include <iostream>
#include <cmath>
using namespace std;

#define V 4 // Number of vertices in the graph
#define INF 99999 // Representing infinite distance

void floyd(int graph[][V]) {
   int dist[V][V];

   // Initializing the distance matrix with the graph
   for (int i = 0; i < V; i++)
      for (int j = 0; j < V; j++)
         dist[i][j] = graph[i][j];

   // Using Floyd's algorithm to find all-pairs shortest paths
   for (int k = 0; k < V; k++) {
      for (int i = 0; i < V; i++) {
         for (int j = 0; j < V; j++) {
            // If the distance through vertex k is shorter than the current distance
            if (dist[i][k] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];
         }
      }
   }

   cout << "The following matrix shows the shortest distances between every pair of vertices:" <<
endl;
   for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                cout << "INF" << "    ";
            else
                cout << dist[i][j] << "    ";
        }
        cout << endl;
    }
}

int main() {
    int graph[V][V] = { {0,   5,  INF, 10},
                        {INF, 0,   3, INF},
                        {INF, INF, 0,   1},
                        {INF, INF, INF, 0}
                      };
    floyd(graph);
    return 0;
}
```

10.B.Write CPP programs to Implement Travelling Sales Person problem using Dynamic programming.

PROGRAM:

```
#include<bits/stdc++.h>
using namespace std;

const int N=105;

int n;
int dp[N][N];
int dist[N][N];
int path[N][N];

int TSP(int u, int mask)
{
    if (mask == (1 << n) - 1)
        return dist[u][0];
    if (dp[u][mask] != -1)
        return dp[u][mask];

    int ans = INT_MAX;
    for (int v = 0; v < n; v++)
    {
        if (v == u)
            continue;
        if (mask & (1 << v))
            continue;

        int cost = dist[u][v] + TSP(v, mask | (1 << v));
        if (cost < ans)
        {
            ans = cost;
            path[u][mask] = v;
        }
    }
    return dp[u][mask] = ans;
}

void printPath(int u, int mask)
{
    if (mask == (1 << n) - 1)
        return;

    int v = path[u][mask];
    cout<<v<<" ";
    printPath(v, mask | (1 << v));
}

int main()
{
    cin>>n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin>>dist[i][j];

    memset(dp, -1, sizeof dp);

    int ans = TSP(0, 1);
    cout<<"Minimum cost: "<<ans<<endl;
    cout<<"Path: "<<0<<" ";
    printPath(0, 1);
    cout<<endl;

    return 0;
}
```



Input Graph (`graph[V][V]` **)**

```
0   5   INF 10
INF 0   3   INF
INF INF 0   1
INF INF INF 0
```

Output:

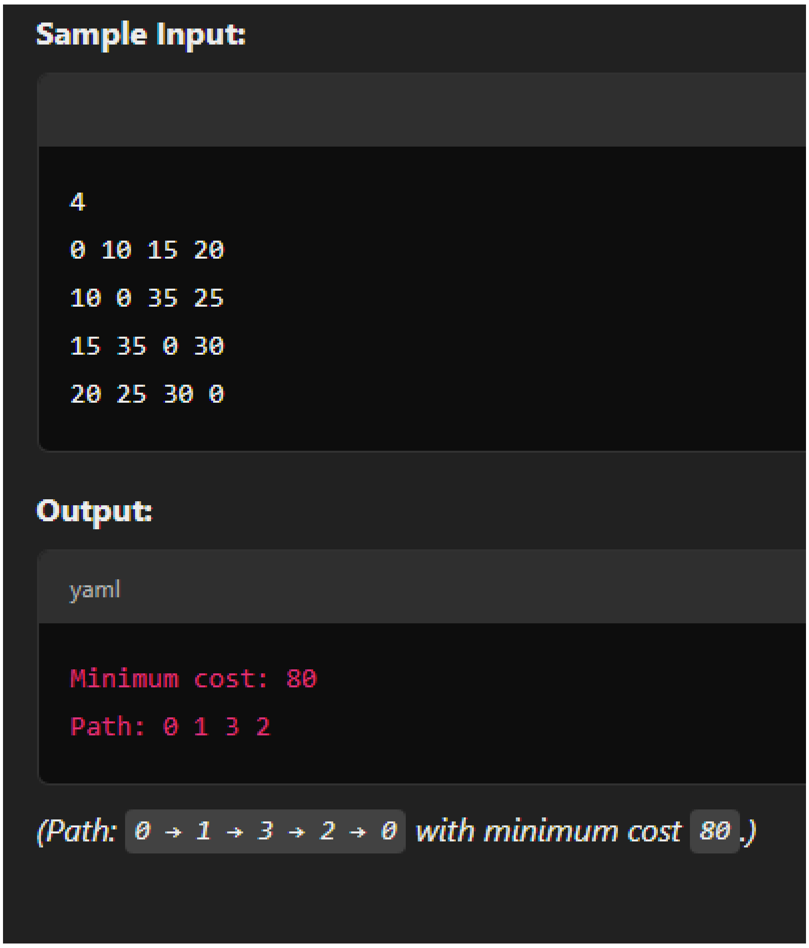
```
The following matrix shows the shortest distances between every pair of vertices:
0     5     8     9
INF   0     3     4
INF   INF   0     1
INF   INF   INF   0
```

Program 2: Traveling Salesperson Problem (TSP) using Dynamic Programming

This program finds the **minimum cost path** for the **Traveling Salesperson Problem (TSP)** using **Bitmasking + Dynamic Programming**.

Step-by-Step Execution

1. **User Inputs the Distance Matrix (`dist[n][n]`)**
2. **Recursive TSP Function (`TSP(u, mask)`)**
    - If all cities are visited, return distance to start city.
    - If already computed (`dp[u][mask]`), return stored value.
    - Try visiting unvisited cities and update the minimum cost.
    - Store the best next city in `path[][]`.
3. **Print the Minimum Cost & Path**
    - Use `printPath(u, mask)` to display the tour.



Sample Input:

```
4
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
```

Output:

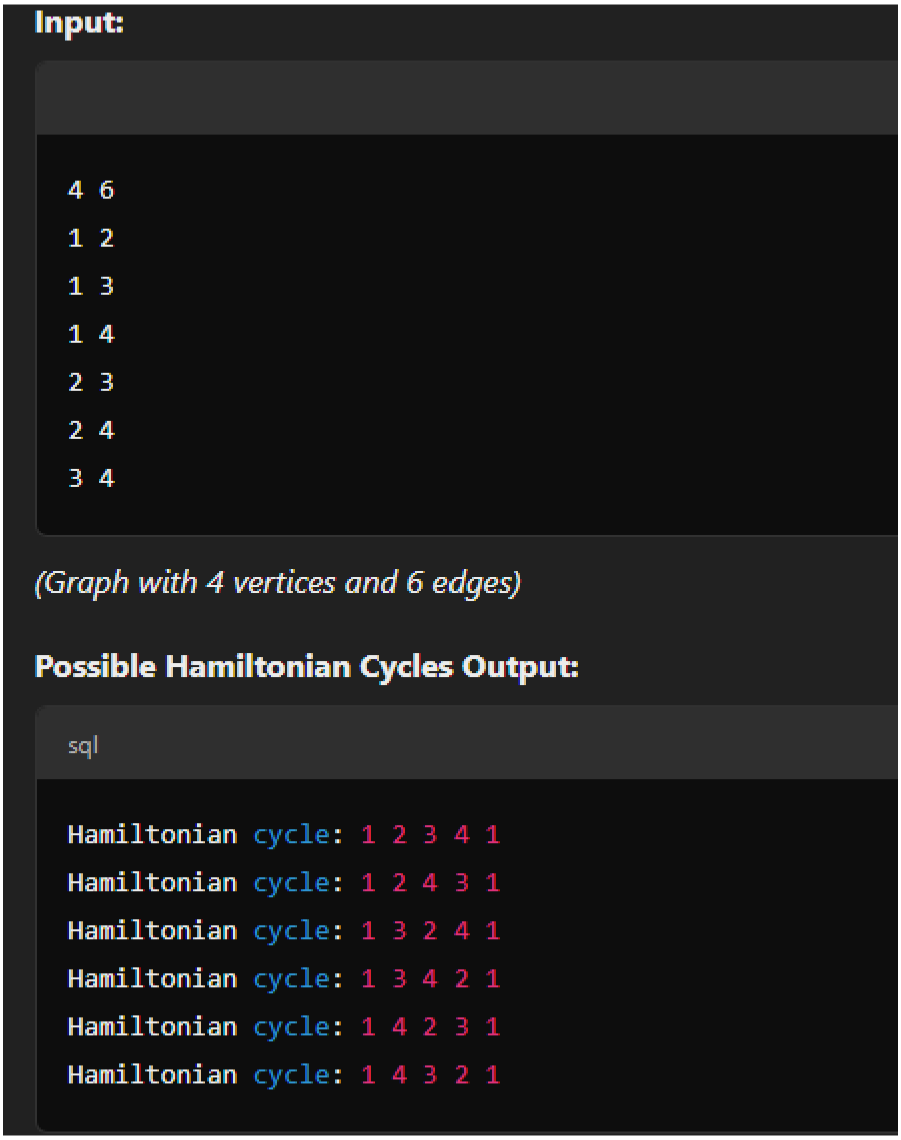
```
Minimum cost: 80
Path: 0 1 3 2
```

(Path: `0 → 1 → 3 → 2 → 0` with minimum cost `80`.)

11. Design and implement in CPP to find a subset of a given set S = {Sl, S2,.....,Sn} of n positive integers whose SUM is equal to a given positive integer d. For example, if S ={1, 2, 5, 6, 8} and d= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution

PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1005;
int n, e, u, v, g[N][N], path[N];
bool vis[N];

bool isValid(int pos) {
   if (g[path[pos - 1]][path[pos]] == 0) {
      return false;
   }
   for (int i = 0; i < pos; i++) {
      if (path[i] == path[pos]) {
         return false;
      }
   }
   return true;
}

void printPath(int pos) {
    cout << "Hamiltonian cycle: ";
    for (int i = 0; i < pos; i++) {
        cout << path[i] << " ";
    }
    cout << path[0] << endl;
}

void findCycle(int pos) {
    if (pos == n) {
        if (g[path[pos - 1]][path[0]]) {
            printPath(pos);
        }
        return;
    }
    for (int i = 1; i <= n; i++) {
        path[pos] = i;
        if (isValid(pos)) {
            findCycle(pos + 1);
        }
    }
}

int main() {
    cin >> n >> e;
    for (int i = 0; i < e; i++) {
        cin >> u >> v;
        g[u][v] = g[v][u] = 1;
    }
    path[0] = 1;
    findCycle(1);
    return 0;
}
```



Input:

```
4 6
1 2
1 3
1 4
2 3
2 4
3 4
```

(Graph with 4 vertices and 6 edges)

Possible Hamiltonian Cycles Output:

```sql
Hamiltonian cycle: 1 2 3 4 1
Hamiltonian cycle: 1 2 4 3 1
Hamiltonian cycle: 1 3 2 4 1
Hamiltonian cycle: 1 3 4 2 1
Hamiltonian cycle: 1 4 2 3 1
Hamiltonian cycle: 1 4 3 2 1
```