```
Aim:- implement bankers algorithm using cpp code
```

```
 include <vector>

Using namespace std;

// Function to check if the requested resources can be granted
Bool isSafe(vector<vector<int>>& allocation, vector<vector<int>>& max, vector<int>&
available, vector<int>& need, int process) {
  // Check if the requested resources are less than or equal to available resources
  For (int i = 0; i < allocation[process].size(); ++i) {
    If (need[process][i] > available[i]) {
      Return false;
    }
  }

  // Simulate the allocation
  For (int i = 0; i < allocation[process].size(); ++i) {
    Available[i] -= need[process][i];
    Allocation[process][i] += need[process][i];
    Need[process][i] = 0;
  }

  // Check if the system is still in a safe state after allocation
  Vector<bool> finish(allocation.size(), false);
  Int count = 0;
```

```
Vector<int> safeSequence;


While (count < allocation.size()) {
    Bool found = false;
    For (int i = 0; i < allocation.size(); ++i) {
        If (!finish[i]) {
            Bool safe = true;
            For (int j = 0; j < allocation[i].size(); ++j) {
                If (need[i][j] > available[j]) {
                    Safe = false;
                    Break;
                }
            }

            If (safe) {
                For (int j = 0; j < allocation[i].size(); ++j) {
                    Available[j] += allocation[i][j];
                }
                Finish[i] = true;
                safeSequence.push_back(i);
                ++count;
                Found = true;
            }
        }
    }
```

```
        If (!found) {
            Return false; // Deadlock detected
        }
    }

    Cout << “Safe Sequence: “;
    For (int i = 0; i < safeSequence.size(); ++i) {
        Cout << safeSequence[i] << “ “;
    }
    Cout << endl;

    Return true; // System is in a safe state
}

Int main() {
    Int numProcesses, numResources;

    Cout << “Enter number of processes: “;
    Cin >> numProcesses;
    Cout << “Enter number of resources: “;
    Cin >> numResources;

    Vector<vector<int>> allocation(numProcesses, vector<int>(numResources));
    Vector<vector<int>> max(numProcesses, vector<int>(numResources));
    Vector<int> available(numResources);
```

```
// Input allocation matrix
Cout << “Enter allocation matrix:” << endl;
For (int i = 0; i < numProcesses; ++i) {
  For (int j = 0; j < numResources; ++j) {
    Cin >> allocation[i][j];
  }
}

// Input max matrix
Cout << “Enter max matrix:” << endl;
For (int i = 0; i < numProcesses; ++i) {
  For (int j = 0; j < numResources; ++j) {
    Cin >> max[i][j];
  }
}

// Input available resources
Cout << “Enter available resources:” << endl;
For (int i = 0; i < numResources; ++i) {
  Cin >> available[i];
}

// Calculate need matrix
Vector<vector<int>> need(numProcesses, vector<int>(numResources));
For (int i = 0; i < numProcesses; ++i) {
  For (int j = 0; j < numResources; ++j) {
```

```
            Need[i][j] = max[i][j] – allocation[i][j];
        }
    }

    // Input process requesting resources
    Int process;
    Cout << “Enter process requesting resources: “;
    Cin >> process;

    // Check if the requested resources can be granted
    If (isSafe(allocation, max, available, need, process)) {
        Cout << “Resources granted.” << endl;
    } else {
        Cout << “Resources cannot be granted due to deadlock.” << endl;
    }

    Return 0;
}
```