**Question:**

Program to implement the conceptof hill climbing.

**Description:**

Hill climbing is a local search optimization algorithm used to find the best solution to a problem by making incremental changes to an initial solution. The goal is to find the "peak" or optimal solution in the search space.

**Approach:**

1. Initial Solution:
   * Start with an initial solution, which can be randomly generated or based on some heuristic.
   * This solution represents your starting point in the "landscape" of all possible solutions.
2. Evaluate Current Solution:
   * Calculate the "fitness" or "quality" of the current solution using an objective function.
   * This function quantifies how good the solution is based on your problem's criteria.
3. Generate Neighboring Solutions:
   * Create one or more "neighbor" solutions by making small modifications to the current solution.
   * These modifications are typically minor tweaks, like changing a single variable slightly.
4. Evaluate and Compare:
   * Calculate the fitness of each neighboring solution.
   * Compare the fitness of the neighbors to the current solution.
5. Move to Better Solution:
   * If any neighbor has a better fitness than the current solution, move to the best neighbor.
   * This neighbor becomes the new current solution.
6. Repeat or Terminate:
   * Repeat steps 2-5 until one of these conditions is met: a) No neighbor is better than the current solution (local optimum reached). b) A predefined number of iterations is completed. c) A satisfactory solution is found (reaches a threshold fitness value).
7. Return Best Solution:
   * **Once terminated, return the best solution found during the process.**

**Code Implementation:**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <algorithm>

using namespace std;

*// Function to optimize (objective function)*

int objective\_function(const vector<int>& solution) {

int sum = 0;

for (int value : solution) {

sum += value \* value;

}

return sum;

}

*// Generate a random neighbor*

vector<int> get\_neighbor(const vector<int>& current\_solution) {

vector<int> neighbor = current\_solution;

int index = rand() % neighbor.size();

neighbor[index] += (rand() % 3) - 1; *// -1, 0, or 1*

return neighbor;

}

vector<int> hill\_climbing(int dimension, int max\_iterations) {

srand(time(0));

// Generate initial random solution

vector<int> current\_solution(dimension);

for (int& value : current\_solution) {

value = rand() % 100;

}

// Print initial solution

cout << "Initial solution: ";

for (int value : current\_solution) {

cout << value << " ";

}

cout << endl;

cout << "Initial objective function value: " << objective\_function(current\_solution) << endl;

int current\_score = objective\_function(current\_solution);

for (int i = 0; i < max\_iterations; i++) {

vector<int> neighbor = get\_neighbor(current\_solution);

int neighbor\_score = objective\_function(neighbor);

if (neighbor\_score > current\_score) {

current\_solution = neighbor;

current\_score = neighbor\_score;

}

}

return current\_solution;

}

void run\_hill\_climbing(int dimension, int max\_iterations) {

cout << "Running hill climbing with dimension " << dimension

<< " and " << max\_iterations << " iterations:" << endl;

vector<int> best\_solution = hill\_climbing(dimension, max\_iterations);

cout << "Final solution: ";

for (int value : best\_solution) {

cout << value << " ";

}

cout << endl;

cout << "Final objective function value: " << objective\_function(best\_solution) << endl;

cout << endl;

}

int main() {

// Sample input cases

run\_hill\_climbing(3, 100);

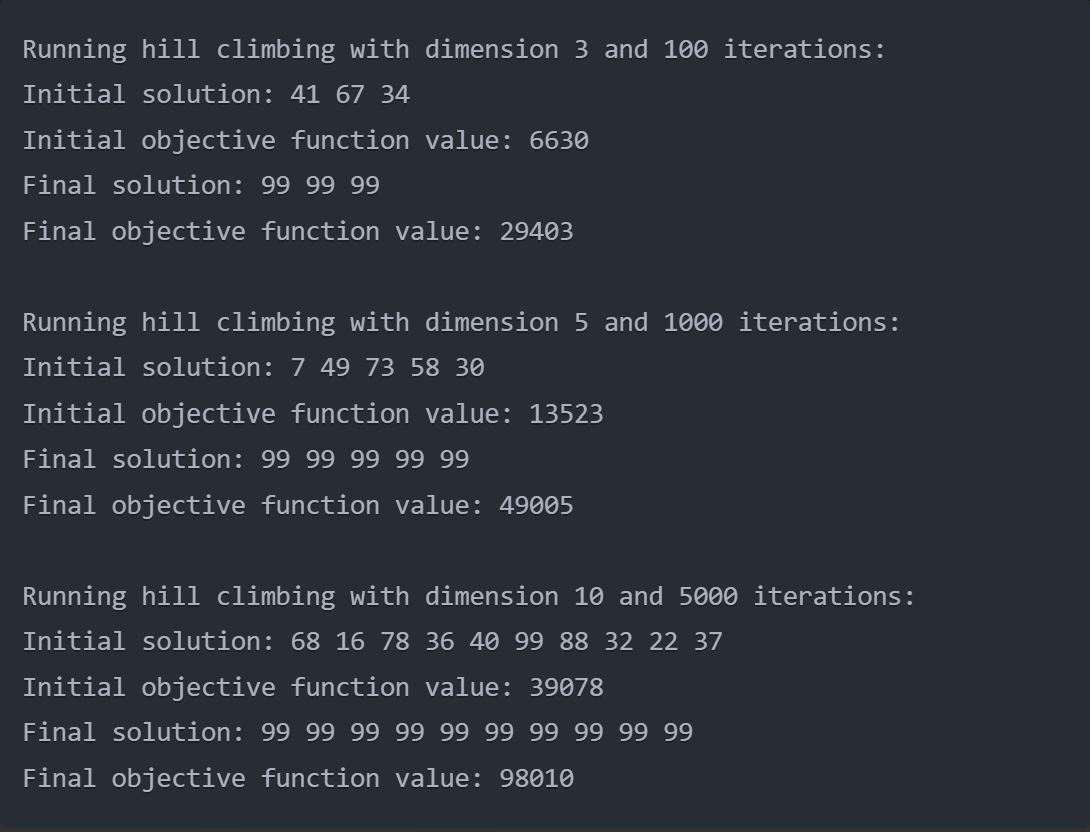
run\_hill\_climbing(5, 1000);

run\_hill\_climbing(10, 5000);

return 0;

}

**Output:**

****