**#include <iostream>**

**#include <algorithm>  // Required for the sort() function**

**using namespace std;**

**// Structure to represent an item with a value and weight**

**struct Item {**

**int value, weight;**

**// Constructor to initialize an item with a specific value and weight**

**Item(int value, int weight) {**

**this->value = value;**

**this->weight = weight;**

**}**

**};**

**// Comparison function to sort items based on their value-to-weight ratio**

**bool cmp(struct Item a, struct Item b) {**

**double r1 = (double)a.value / (double)a.weight; // Ratio of value to weight for item a**

**double r2 = (double)b.value / (double)b.weight; // Ratio of value to weight for item b**

**return r1 > r2; // Sort in descending order of value-to-weight ratio**

**}**

**// Function to calculate the maximum value in the knapsack using the fractional knapsack approach**

**double fractionalKnapsack(int W, struct Item arr[], int N) {**

**// Sort the items by their value-to-weight ratio in descending order**

**sort(arr, arr + N, cmp);**

**double finalValue = 0.0; // Variable to store the final maximum value**

**// Loop through all items**

**for (int i = 0; i < N; i++) {**

**// If adding the current item doesn't exceed the capacity**

**if (arr[i].weight <= W) {**

**W -= arr[i].weight;          // Subtract item weight from the total capacity**

**finalValue += arr[i].value; // Add item value to the final value**

**}**

**// If adding the current item exceeds the capacity**

**else {**

**// Add the fraction of the item that can fit in the remaining capacity**

**finalValue += arr[i].value \* ((double)W / (double)arr[i].weight);**

**break; // Knapsack is full, exit the loop**

**}**

**}**

**return finalValue; // Return the final maximum value obtained**

**}**

**int main() {**

**int W = 20; // Total weight capacity of the knapsack**

**// Array of items with value and weight pairs**

**Item arr[] = { {10, 3}, {20, 5}, {21, 5}, {30, 8}, {16, 4} };**

**int N = sizeof(arr) / sizeof(arr[0]); // Number of items**

**// Output the maximum value we can obtain from the knapsack**

**cout << "Maximum value we can obtain = " << fractionalKnapsack(W, arr, N);**

**return 0;**

**}**