/\* KNAPSACK GREEDY \*/

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Item {

int value;

int weight;

double ratio; // Value-to-weight ratio

};

// Function to calculate value-to-weight ratios for items

void calculateRatio(vector<Item>& items) {

for (auto& item : items) {

item.ratio = (double)item.value / item.weight;

}

}

// Greedy approach for fractional knapsack problem

double fractionalKnapsack(int W, vector<Item>& items) {

// Calculate value-to-weight ratios for items

calculateRatio(items);

// Sort items based on value-to-weight ratio (in non-increasing order)

sort(items.begin(), items.end(), [](const Item& a, const Item& b) {

return a.ratio > b.ratio;

});

double totalValue = 0.0;

int currentWeight = 0;

// Iterate through sorted items

for (const auto& item : items) {

// If adding the whole item doesn't exceed the capacity

if (currentWeight + item.weight <= W) {

totalValue += item.value;

currentWeight += item.weight;

} else { // Otherwise, add a fraction of the item

int remainingWeight = W - currentWeight;

totalValue += item.ratio \* remainingWeight;

break; // Knapsack full, break the loop

}

}

return totalValue;

}

int main() {

int capacity;

cout << "Enter the capacity of the knapsack: ";

cin >> capacity;

int n;

cout << "Enter the number of items: ";

cin >> n;

vector<Item> items(n);

cout << "Enter the value and weight of each item:" << endl;

for (int i = 0; i < n; ++i) {

cout << "Item " << i + 1 << ": ";

cin >> items[i].value >> items[i].weight;

}

double maxValue = fractionalKnapsack(capacity, items);

cout << "Maximum value that can be obtained: " << maxValue << endl;

return 0;

}