EXPERIMENT 7

Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

```
#include <stdio.h>

// Define a structure to represent an item
typedef struct {
    int weight;
    int value;
    double ratio;
} Item;

// Function to sort items based on their value-to-weight ratio using bubble sort
void bubbleSort(Item items[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; j++) {
            if (items[j].ratio < items[j + 1].ratio) {
                Item temp = items[j];
                items[j] = items[j + 1];
                items[j + 1] = temp;
            }
        }
    }
}

// Function to solve the continuous Knapsack problem using the greedy method
void continuousKnapsack(Item items[], int n, int capacity) {
    bubbleSort(items, n);
    int currentWeight = 0;
    double finalValue = 0.0;
```

```
    printf("Continuous Knapsack selected items:\n");
    for (int i = 0; i < n; i++) {
        if (currentWeight + items[i].weight <= capacity) {
            currentWeight += items[i].weight;
            finalValue += items[i].value;
            printf("Item %d: fully selected\n", i + 1);
        } else {
            int remainingCapacity = capacity - currentWeight;
finalValue += items[i].value * ((double) remainingCapacity / items[i].weight);
printf("Item %d: %.2f%% selected\n", i + 1, (double) remainingCapacity / items[i].weight *100);
            break;
        }
    }
    printf("Total value: %.2f\n", finalValue);
}

// Function to solve the discrete Knapsack problem using the greedy method
void discreteKnapsack(Item items[], int n, int capacity) {
    bubbleSort(items, n);

    int currentWeight = 0;
    int finalValue = 0;
    printf("Discrete Knapsack selected items:\n");
    for (int i = 0; i < n; i++) {
        if (currentWeight + items[i].weight <= capacity) {
            currentWeight += items[i].weight;
            finalValue += items[i].value;
            printf("Item %d: selected\n", i + 1);
        }
    }
    printf("Total value: %d\n", finalValue);
```

```
}

// Main function to demonstrate both Knapsack solutions
int main() {
    int n, capacity;

    printf("Enter the number of items: ");
    scanf("%d", &n);

    Item items[n];

    for (int i = 0; i < n; i++) {
        printf("Enter weight and value for item %d: ", i + 1);
        scanf("%d %d", &items[i].weight, &items[i].value);
        items[i].ratio = (double) items[i].value / items[i].weight;
    }

    printf("Enter the knapsack capacity: ");
    scanf("%d", &capacity);

    // Solve the continuous Knapsack problem
    continuousKnapsack(items, n, capacity);

    // Solve the discrete Knapsack problem
    discreteKnapsack(items, n, capacity);

    return 0;
}
```