LAB ASSIGNMENT – 03

Program

```
#include <stdio.h>
#include <stdlib.h>

// Structure to represent an item
typedef struct {
    int value;
    int weight;
    double ratio; // value-to-weight ratio
    int index;    // original index for tracking
} Item;

// Function to compare items based on value-to-weight ratio (descending order)
int compareItems(const void *a, const void *b) {
    Item *itemA = (Item *)a;
    Item *itemB = (Item *)b;

    if (itemB->ratio > itemA->ratio) return 1;
    if (itemB->ratio < itemA->ratio) return -1;
    return 0;
}

// Function to solve fractional knapsack problem
double fractionalKnapsack(int capacity, Item items[], int n) {
    // Sort items by value-to-weight ratio in descending order
    qsort(items, n, sizeof(Item), compareItems);

    double totalValue = 0.0;
    int currentWeight = 0;

    printf("\nGreedy Selection Process:\n");
    printf("%-10s %-10s %-10s %-15s %-15s\n",
           "Item", "Value", "Weight", "Ratio", "Amount Taken");
    printf("-----------------------------------------------------------\n");
```

```
    for (int i = 0; i < n; i++) {
        if (currentWeight + items[i].weight <= capacity) {
            // Take the whole item
            currentWeight += items[i].weight;
            totalValue += items[i].value;

            printf("%-10d %-10d %-10d %-15.2f %-15s\n",
                   items[i].index, items[i].value, items[i].weight,
                   items[i].ratio, "Full");
        } else {
            // Take fraction of the item
            int remainingCapacity = capacity - currentWeight;
            if (remainingCapacity > 0) {
                double fraction = (double)remainingCapacity / items[i].weight;
                totalValue += items[i].value * fraction;

                printf("%-10d %-10d %-10d %-15.2f %-15.2f\n",
                       items[i].index, items[i].value, items[i].weight,
                       items[i].ratio, fraction);

                currentWeight = capacity;
            }
            break; // Knapsack is full
        }
    }

    return totalValue;
}

// Function to display items
void displayItems(Item items[], int n) {
    printf("\nItems Available:\n");
    printf("%-10s %-10s %-10s %-15s\n", "Item", "Value", "Weight", "Ratio");
    printf("-----------------------------------------\n");
```

```
    for (int i = 0; i < n; i++) {
        printf("%-10d %-10d %-10d %-15.2f\n",
               items[i].index, items[i].value, items[i].weight, items[i].ratio);
    }
}

int main() {
    int n, capacity;
    printf("=== FRACTIONAL KNAPSACK PROBLEM SOLVER ===\n");
    printf("Using Greedy Algorithm (Value-to-Weight Ratio)\n\n");
    // Input number of items
    printf("Enter the number of items: ");
    scanf("%d", &n);
    if (n <= 0) {
        printf("Error: Number of items must be positive.\n");
        return 1;
    }
    // Allocate memory for items
    Item *items = (Item *)malloc(n * sizeof(Item));
    if (items == NULL) {
        printf("Error: Memory allocation failed.\n");
        return 1;
    }
    // Input item details
    printf("Enter value and weight for each item:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d - Value: ", i + 1);
        scanf("%d", &items[i].value);
        printf("Item %d - Weight: ", i + 1);
        scanf("%d", &items[i].weight);
        if (items[i].weight <= 0) {
            printf("Error: Weight must be positive.\n");
            free(items);
            return 1;
        }
```

```
        items[i].ratio = (double)items[i].value / items[i].weight;
        items[i].index = i + 1;
    }

    // Input knapsack capacity
    printf("Enter the knapsack capacity: ");
    scanf("%d", &capacity);
    if (capacity <= 0) {
        printf("Error: Capacity must be positive.\n");
        free(items);
        return 1;
    }

    // Display original items
    displayItems(items, n);

    // Solve fractional knapsack
    printf("\nKnapsack Capacity: %d\n", capacity);
    double maxValue = fractionalKnapsack(capacity, items, n);

    printf("\n=== SOLUTION ===\n");
    printf("Maximum value that can be obtained: %.2f\n", maxValue);

    // Algorithm complexity information
    printf("\n=== ALGORITHM ANALYSIS ===\n");
    printf("Time Complexity: O(n log n) - due to sorting\n");
    printf("Space Complexity: O(1) - excluding input storage\n");
    printf("Method: Greedy Algorithm based on value-to-weight ratio\n");

    // Free allocated memory
    free(items);

    return 0;
}
```

OUTPUT

```
(base) PS C:\Users\Karunya\Documents\Sem 7 - LAs\DAA\execution_daa> gcc .\fractional_knapsack.c
(base) PS C:\Users\Karunya\Documents\Sem 7 - LAs\DAA\execution_daa> ./a.exe
=== FRACTIONAL KNAPSACK PROBLEM SOLVER ===
Using Greedy Algorithm (Value-to-Weight Ratio)

Enter the number of items: 3
Enter value and weight for each item:
Item 1 - Value: 60
Item 1 - Weight: 10
Item 2 - Value: 100
Item 2 - Weight: 20
Item 3 - Value: 120
Item 3 - Weight: 30
Enter the knapsack capacity: 50

Items Available:
Item       Value      Weight     Ratio
-----------------------------------------
1          60         10         6.00
2          100        20         5.00
3          120        30         4.00

Knapsack Capacity: 50

Greedy Selection Process:
Item       Value      Weight     Ratio           Amount Taken
-----------------------------------------------------------
1          60         10         6.00            Full
2          100        20         5.00            Full
3          120        30         4.00            0.67

=== SOLUTION ===
Maximum value that can be obtained: 240.00

=== ALGORITHM ANALYSIS ===
Time Complexity: O(n log n) - due to sorting
Space Complexity: O(1) - excluding input storage
Method: Greedy Algorithm based on value-to-weight ratio
```