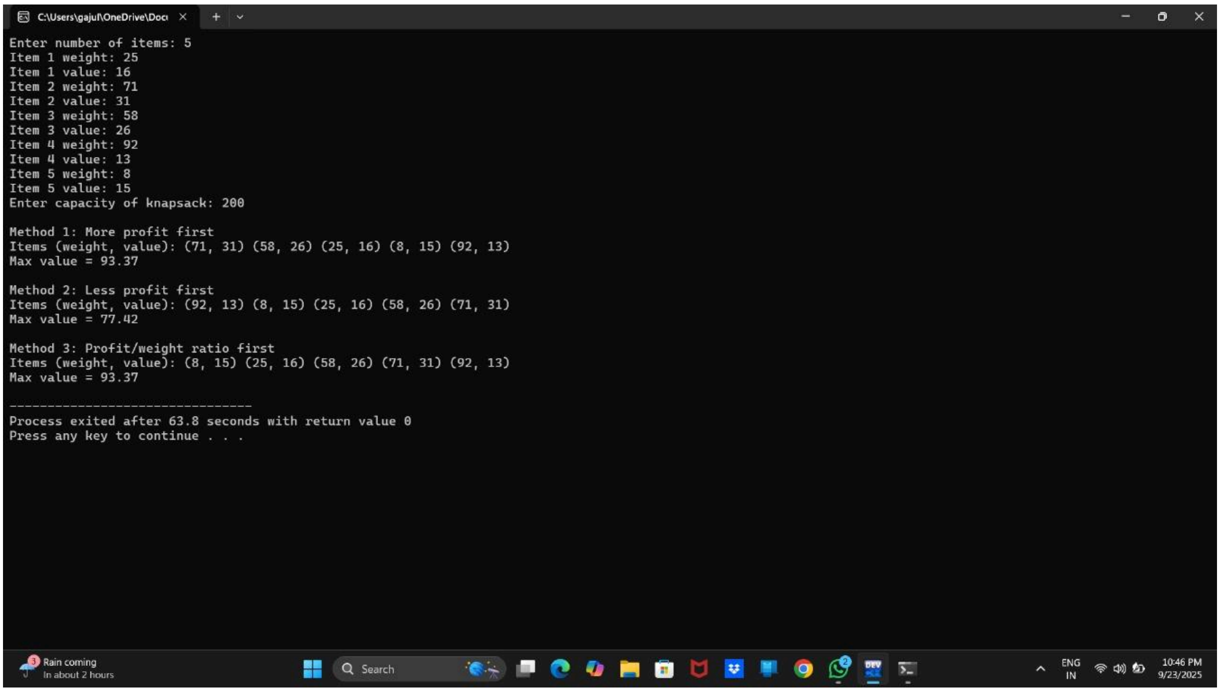
KNAPSACK

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void swapFloat(float *a, float *b) {
    float temp = *a;
    *a = *b;
    *b = temp;
}
void sortByProfitDesc(int weights[], int values[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (values[j] > values[i]) {
                swap(&values[i], &values[j]);
                swap(&weights[i], &weights[j]);
            }
        }
    }
}
void sortByProfitAsc(int weights[], int values[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (values[j] < values[i]) {
                swap(&values[i], &values[j]);
                swap(&weights[i], &weights[j]);
            }
        }
    }
}
void sortByRatioDesc(int weights[], int values[], int n) {
    float ratios[n];
    for (int i = 0; i < n; i++) {
        ratios[i] = (float)values[i] / weights[i];
    }

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (ratios[j] > ratios[i]) {
                swap(&values[i], &values[j]);
                swap(&weights[i], &weights[j]);
                swapFloat(&ratios[i], &ratios[j]);
            }
        }
    }
}
float fractionalKnapsack(int capacity, int weights[], int values[], int n) {
    float totalValue = 0.0;

    for (int i = 0; i < n && capacity > 0; i++) {
        if (weights[i] <= capacity) {
            capacity -= weights[i];
            totalValue += values[i];
        } else {
            totalValue += values[i] * ((float)capacity / weights[i]);
            capacity = 0;
        }
    }

    return totalValue;
}
void printItems(int weights[], int values[], int n) {
    printf("Items (weight, value): ");
    for (int i = 0; i < n; i++) {
        printf("(%d, %d) ", weights[i], values[i]);
    }
    printf("\n");
}

int main() {
    int n, capacity;

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

    int weights[n], values[n];
    int weightsCopy[n], valuesCopy[n];

    for (int i = 0; i < n; i++) {
        printf("Item %d weight: ", i + 1);
        scanf("%d", &weights[i]);
        printf("Item %d value: ", i + 1);
        scanf("%d", &values[i]);
    }

    printf("Enter capacity of knapsack: ");
    scanf("%d", &capacity);
    for (int i = 0; i < n; i++) {
        weightsCopy[i] = weights[i];
        valuesCopy[i] = values[i];
    }
    sortByProfitDesc(weightsCopy, valuesCopy, n);
    printf("\nMethod 1: More profit first\n");
    printItems(weightsCopy, valuesCopy, n);
    printf("Max value = %.2f\n", fractionalKnapsack(capacity, weightsCopy, valuesCopy, n));
    for (int i = 0; i < n; i++) {
        weightsCopy[i] = weights[i];
        valuesCopy[i] = values[i];
    }
    sortByProfitAsc(weightsCopy, valuesCopy, n);
    printf("\nMethod 2: Less profit first\n");
    printItems(weightsCopy, valuesCopy, n);
    printf("Max value = %.2f\n", fractionalKnapsack(capacity, weightsCopy, valuesCopy, n));
    for (int i = 0; i < n; i++) {
        weightsCopy[i] = weights[i];
        valuesCopy[i] = values[i];
    }
    sortByRatioDesc(weightsCopy, valuesCopy, n);
    printf("\nMethod 3: Profit/weight ratio first\n");
    printItems(weightsCopy, valuesCopy, n);
    printf("Max value = %.2f\n", fractionalKnapsack(capacity, weightsCopy, valuesCopy, n));

    return 0;
}
```



```
Enter number of items: 5
Item 1 weight: 25
Item 1 value: 16
Item 2 weight: 71
Item 2 value: 31
Item 3 weight: 58
Item 3 value: 26
Item 4 weight: 92
Item 4 value: 13
Item 5 weight: 8
Item 5 value: 15
Enter capacity of knapsack: 200

Method 1: More profit first
Items (weight, value): (71, 31) (58, 26) (25, 16) (8, 15) (92, 13)
Max value = 93.37

Method 2: Less profit first
Items (weight, value): (92, 13) (8, 15) (25, 16) (58, 26) (71, 31)
Max value = 77.42

Method 3: Profit/weight ratio first
Items (weight, value): (8, 15) (25, 16) (58, 26) (71, 31) (92, 13)
Max value = 93.37

--------------------------------
Process exited after 63.8 seconds with return value 0
Press any key to continue . . .
```