Experiment No.3

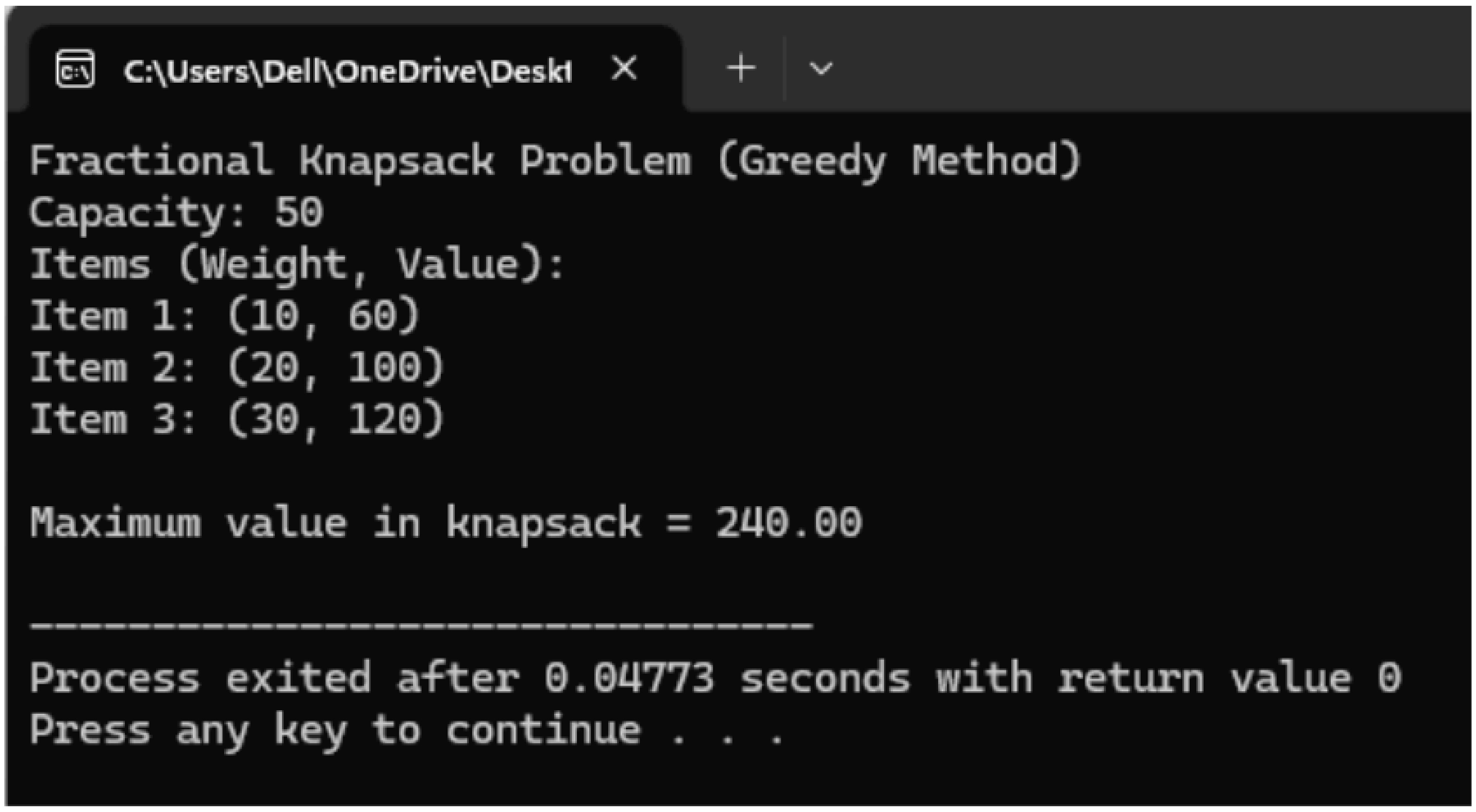
Title : Knapsack Problem Using Greedy Method

Reg.No.24141028

Program 1 - Fractional Knapsack

```
#include <stdio.h>
struct Item {
    int weight;
    int value;
};
void fractionalKnapsack(struct Item items[], int n, int capacity) {
    float ratio[n], totalValue = 0.0;
    int i, j;
    // Calculate value-to-weight ratio
    for (i = 0; i < n; i++) {
        ratio[i] = (float)items[i].value / items[i].weight;
    }
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (ratio[j] < ratio[j + 1]) {
                // Swap ratios
                float tempRatio = ratio[j];
                ratio[j] = ratio[j + 1];
                ratio[j + 1] = tempRatio;
               // Swap items
                struct Item tempItem = items[j];
                items[j] = items[j + 1];
                items[j + 1] = tempItem;
            }
        }
    }
    // Select items
    for (i = 0; i < n; i++) {
        if (capacity >= items[i].weight) {
            capacity -= items[i].weight;
            totalValue += items[i].value;
        } else {
            totalValue += ratio[i] * capacity;
            break;
        }
    }
      printf("\nMaximum value in knapsack = %.2f\n", totalValue);
}
int main() {
    int i;

    int n = 3;

    int capacity = 50;

    struct Item items[] = {

        {10, 60},

        {20, 100},

        {30, 120}

    };

    printf("Fractional Knapsack Problem (Greedy Method)\n");

    printf("Capacity: %d\n", capacity);

    printf("Items (Weight, Value):\n");

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

        printf("Item %d: (%d, %d)\n", i + 1, items[i].weight,
items[i].value);

    }

    fractionalKnapsack(items, n, capacity);

    return 0;

}
```

Output:



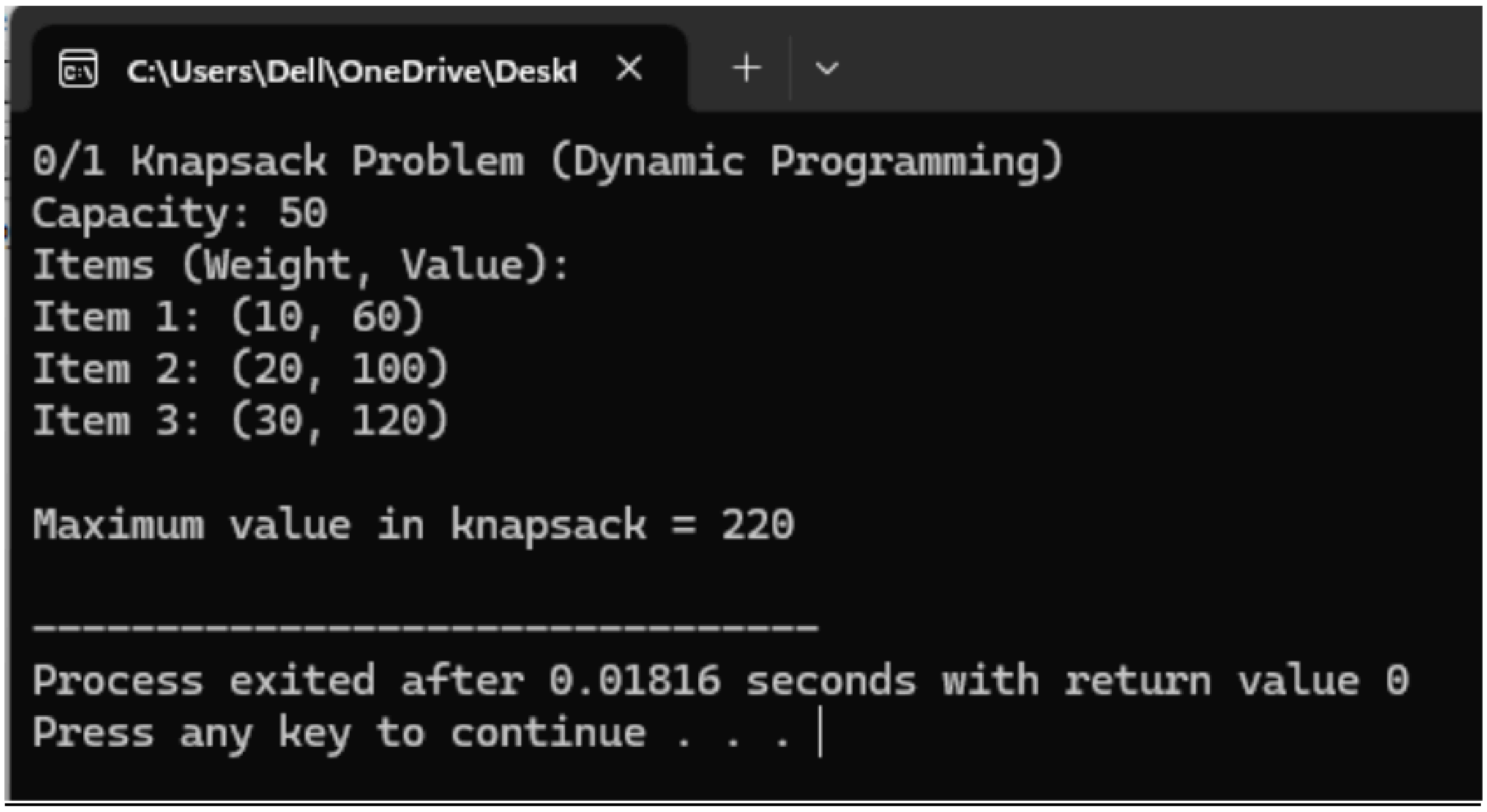
```
Fractional Knapsack Problem (Greedy Method)
Capacity: 50
Items (Weight, Value):
Item 1: (10, 60)
Item 2: (20, 100)
Item 3: (30, 120)

Maximum value in knapsack = 240.00

--------------------------------
Process exited after 0.04773 seconds with return value 0
Press any key to continue . . .
```

Program 2 - 0/1 Knapsack

```
#include <stdio.h>
int max(int a, int b)
{
    return (a > b) ? a : b;
}
// Function to solve 0/1 Knapsack Problem
int knapsack(int weights[], int values[], int n, int capacity)
{
    int i, w;
    int result[n + 1][capacity + 1];
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0)
                result[i][w] = 0;
            else if (weights[i - 1] <= w)
                result[i][w] = max(values[i - 1] + result[i - 1][w - weights[i - 1]], result[i - 1][w]);
            else
                result[i][w] = result[i - 1][w];
         }
    }
       return result[n][capacity];
}
int main() {
    int i;
    int values[] = {60, 100, 120};
    int weights[] = {10, 20, 30};
    int capacity = 50;
    int n = sizeof(values) / sizeof(values[0]);
    printf("0/1 Knapsack Problem (Dynamic Programming)\n");
    printf("Capacity: %d\n", capacity);
    printf("Items (Weight, Value):\n");
    for ( i = 0; i < n; i++) {
        printf("Item %d: (%d, %d)\n", i + 1, weights[i], values[i]);
    }
    int maxValue = knapsack(weights, values, n, capacity);
    printf("\nMaximum value in knapsack = %d\n", maxValue);
    return 0;
```

```
}
```

Output:



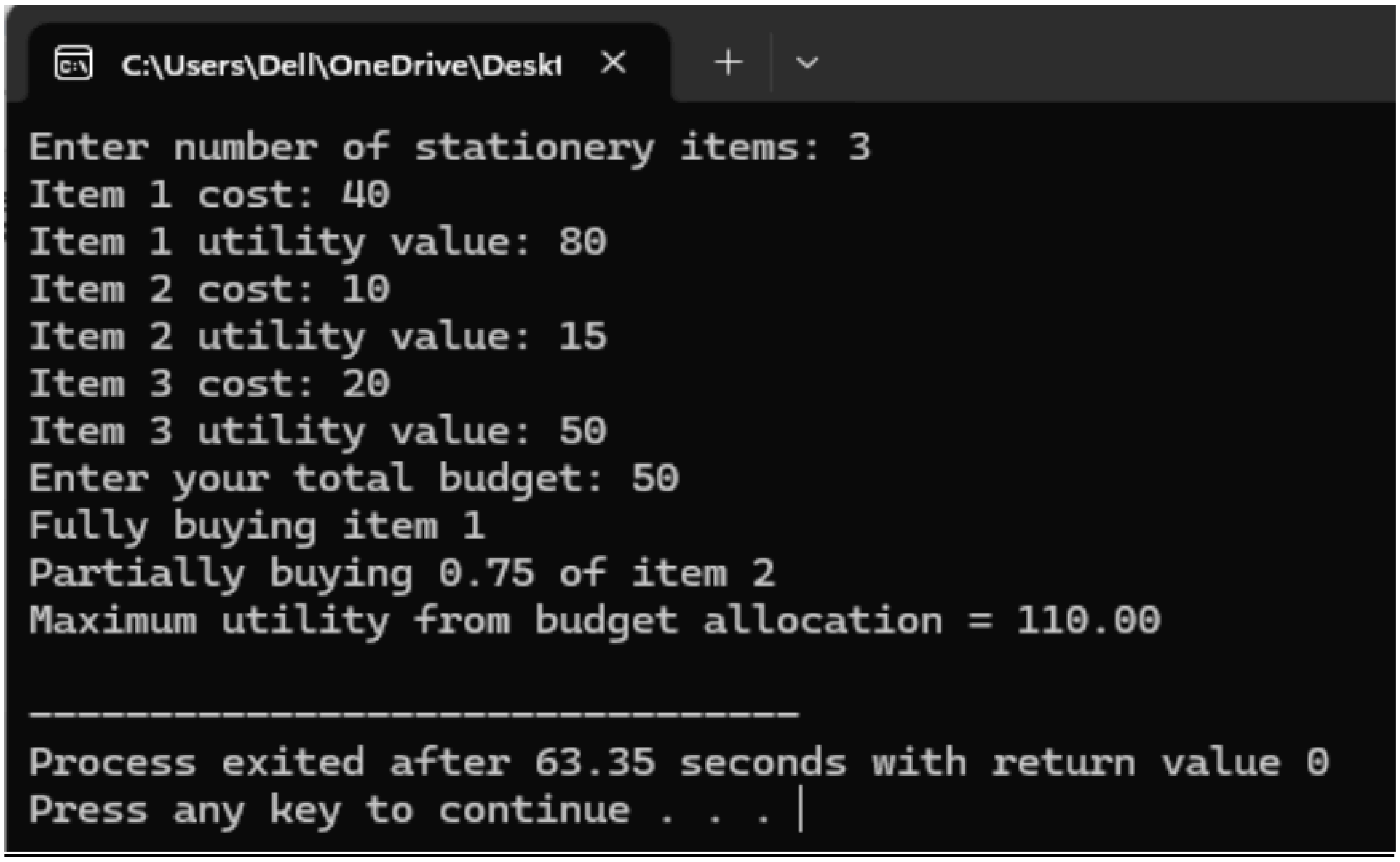
```
0/1 Knapsack Problem (Dynamic Programming)
Capacity: 50
Items (Weight, Value):
Item 1: (10, 60)
Item 2: (20, 100)
Item 3: (30, 120)

Maximum value in knapsack = 220

--------------------------------
Process exited after 0.01816 seconds with return value 0
Press any key to continue . . .
```

Application Based on Knapsack :

```
#include <stdio.h>
struct Item
{
    int cost;
    int utility;
    float ratio;
};
void sort(struct Item items[], int n) {
    int i, j;
    struct Item temp;
    for (i = 0; i < n -1; i++) {
        for (j = 0; j < n - i -1; j++) {
            if (items[j].ratio < items[j+1].ratio) {
                temp = items[j];
                items[j] = items[j+1];
                items[j+1] = temp;
            }
        }
    }
}
float optimizeBudget(struct Item items[], int n, int budget) {
    float totalUtility = 0.0;
    int i;
 for (i = 0; i < n; i++) {
        if (items[i].cost <= budget) {
            budget -= items[i].cost;
            totalUtility += items[i].utility;
            printf("Fully buying item %d\n", i+1);
        } else {
            float fraction = (float)budget / items[i].cost;
            totalUtility += items[i].utility * fraction;
            printf("Partially buying %.2f of item %d\n", fraction, i+1);
            break;
        }
    }
    return totalUtility;
```

```
}
int main()
  {
    int n, budget;
    printf("Enter number of stationery items: ");
    scanf("%d", &n);
    struct Item items[n];
    for (int i = 0; i < n; i++) {
        printf("Item %d cost: ", i+1);
        scanf("%d", &items[i].cost);
        printf("Item %d utility value: ", i+1);
        scanf("%d", &items[i].utility);
        items[i].ratio = (float)items[i].utility / items[i].cost;
    }
    printf("Enter your total budget: ");
    scanf("%d", &budget);
    sort(items, n);
    float maxUtility = optimizeBudget(items, n, budget);
    printf("Maximum utility from budget allocation = %.2f\n",
maxUtility);
    return 0;
}
```

Output:



```
Enter number of stationery items: 3
Item 1 cost: 40
Item 1 utility value: 80
Item 2 cost: 10
Item 2 utility value: 15
Item 3 cost: 20
Item 3 utility value: 50
Enter your total budget: 50
Fully buying item 1
Partially buying 0.75 of item 2
Maximum utility from budget allocation = 110.00

--------------------------------
Process exited after 63.35 seconds with return value 0
Press any key to continue . . .
```