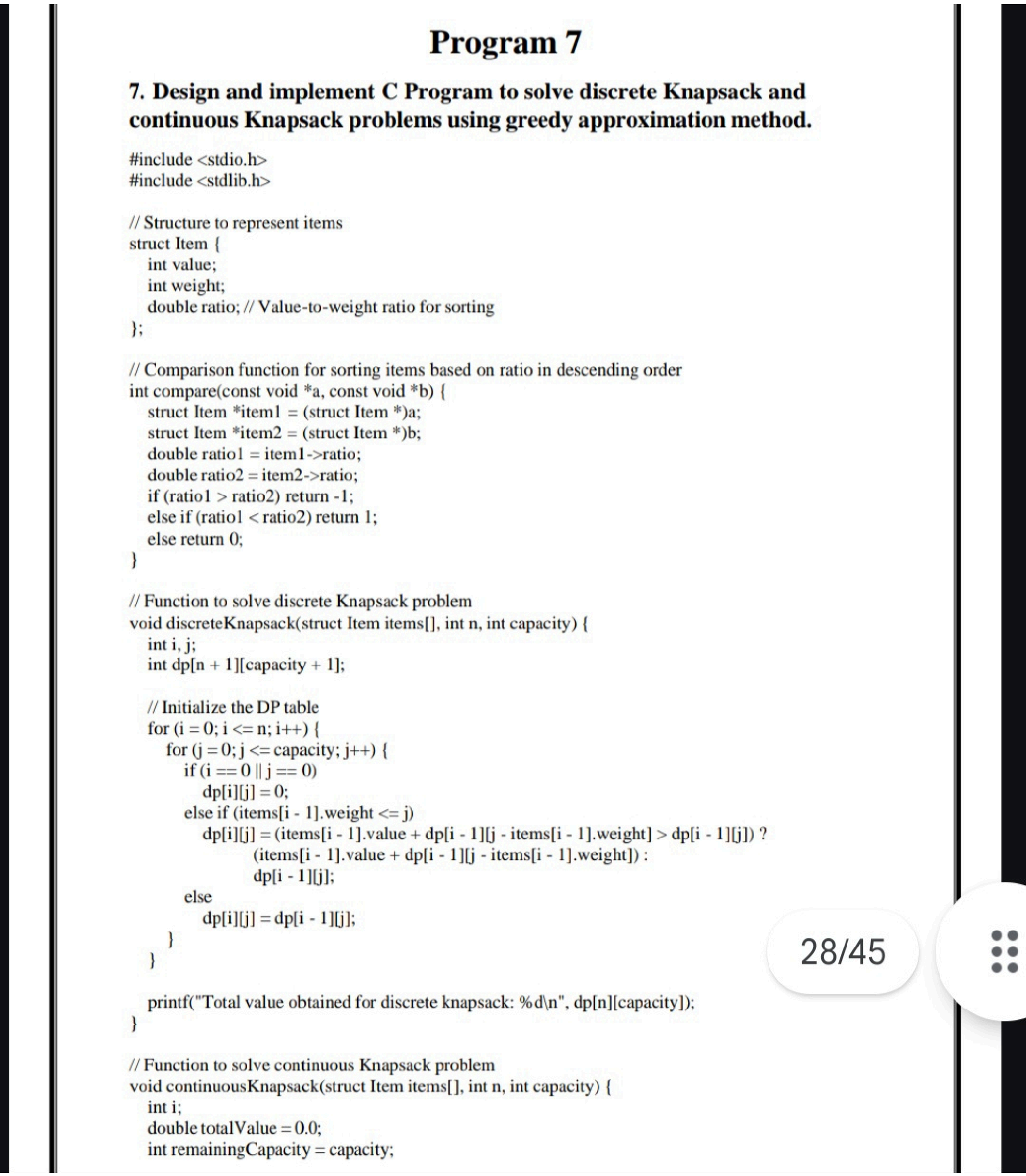


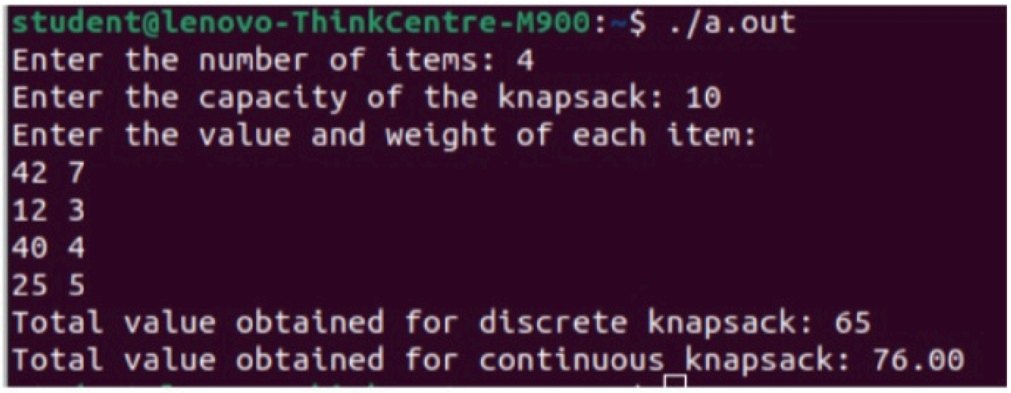
Program 7

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

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

// Structure to represent items
struct Item {
   int value;
   int weight;
   double ratio; // Value-to-weight ratio for sorting
};

// Comparison function for sorting items based on ratio in descending order
int compare(const void *a, const void *b) {
   struct Item *item1 = (struct Item *)a;
   struct Item *item2 = (struct Item *)b;
   double ratio1 = item1->ratio;
   double ratio2 = item2->ratio;
   if (ratio1 > ratio2) return -1;
   else if (ratio1 < ratio2) return 1;
   else return 0;
}

// Function to solve discrete Knapsack problem
void discreteKnapsack(struct Item items[], int n, int capacity) {
   int i, j;
   int dp[n + 1][capacity + 1];

   // Initialize the DP table
   for (i = 0; i <= n; i++) {
      for (j = 0; j <= capacity; j++) {
         if (i == 0 || j == 0)
            dp[i][j] = 0;
         else if (items[i - 1].weight <= j)
            dp[i][j] = (items[i - 1].value + dp[i - 1][j - items[i - 1].weight] > dp[i - 1][j]) ?
                    (items[i - 1].value + dp[i - 1][j - items[i - 1].weight]) :
                    dp[i - 1][j];
         else
            dp[i][j] = dp[i - 1][j];
      }
   }

   printf("Total value obtained for discrete knapsack: %d\n", dp[n][capacity]);
}

// Function to solve continuous Knapsack problem
void continuousKnapsack(struct Item items[], int n, int capacity) {
   int i;
   double totalValue = 0.0;
   int remainingCapacity = capacity;
```

```
    for (i = 0; i < n; i++) {
        if (remainingCapacity >= items[i].weight) {
            totalValue += items[i].value;
            remainingCapacity -= items[i].weight;
        } else {
            totalValue += (double)remainingCapacity / items[i].weight * items[i].value;
            break;
        }
    }

    printf("Total value obtained for continuous knapsack: %.2lf\n", totalValue);
}

int main() {
    int n, capacity, i;
    printf("Enter the number of items: ");
    scanf("%d", &n);

    struct Item items[n];

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

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

    // Sort items based on value-to-weight ratio
    qsort(items, n, sizeof(struct Item), compare);

    discreteKnapsack(items, n, capacity);
    continuousKnapsack(items, n, capacity);

    return 0;
}
```

OUTPUT:



```
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of items: 4
Enter the capacity of the knapsack: 10
Enter the value and weight of each item:
42 7
12 3
40 4
25 5
Total value obtained for discrete knapsack: 65
Total value obtained for continuous knapsack: 76.00
```

29/45