1. Fractional Knapsack Problem

```
#include <bits/stdc++.h>
using namespace std;

// Structure for an item which stores weight and
// corresponding value of Item
struct Item {
 int profit, weight;

 // Constructor
 Item(int profit, int weight)
 {
 this->profit = profit;
 this->weight = weight;
 }
};

// Comparison function to sort Item
// according to profit/weight ratio
static bool cmp(struct Item a, struct Item b)
{
  double  r1  =  (double)a.profit  /  (double)a.weight;
double  r2  =  (double)b.profit  /  (double)b.weight;
return r1 > r2;
}

// Main greedy function to solve problem
double fractionalKnapsack(int C, struct Item arr[], int N)
{
 // Sorting Item on basis of ratio
 sort(arr, arr + N, cmp);

 double finalvalue = 0.0;

 // Looping through all items
 for (int i = 0; i < N; i++) {
```

```
 // If adding Item won't overflow,
 // add it completely
 if (arr[i].weight <= C) {
 C -= arr[i].weight;
 finalvalue += arr[i].profit;
 }

 // If we can't add current Item,
 // add fractional part of it
 else {
 double fraction= ((double)C / (double)arr[i].weight);
finalvalue+= arr[i].profit * fraction;
 break;
 }
 }

 // Returning final value
 return finalvalue;
}

// Driver code
int main()
{
 int C = 50;
 Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };
int N = sizeof(arr) / sizeof(arr[0]);

 // Function call
 cout << fractionalKnapsack(C, arr, N);
 return 0;
}
```

2. Coin change Problem

```
#include <stdio.h>
const int numCoins = 5, i=0;
int output[numCoins];
int greedyCoinChange(int c[], int n, int i)
{
 if (n == 0) {
 return 0;
 }
 if(c[i]<=n)
 {
 return 1+greedyCoinChange(c, n-c[i], i);  }
 else
 {
 return greedyCoinChange(c, n, i+1);  }
}

int main()
{
 int c[numCoins] = {50, 25, 20, 10, 5}; //sorted array
int n = 140;
 int total_coin= greedyCoinChange(c, n, i);
printf("Total coins: %d\n", total_coin);
}
```