KEY

Quiz One: cs5050, 10 points

Name: **Anumber:**

Consider the following simple solution of the Knapsack decision problem
S[i]> 0, 1<= i<= n, initial call knapFit(n,S)

```
Bool knapFit(int i, int s) // i is item index <= n, s is the current capacity of knapsack

    If(s==0) return true;   // perfectly filled so true
    If(i==0 && s>0) return false;  // not filled, no more object so false
    If(s<0) return false;   // over filled so false
    return(knapFit(i-1, s-s[i]) || knapFit(i-1, s) // use or don't use the item
```

a) Add short comments to the code above for each terminal condition to answer the question: "why do we need this case?"

```
Bool knapFit(int i, int s) // i is item index <= n, s is the current capacity of knapsack

    If(s==0) return true;
    If(i==0 && s>0) return false;
    If(s<0) return false;
    return(knapFit(i-1, s-s[i]) || knapFit(i-1,s) // use or don't use the item
```

knapFit(i, s-s[i]) // put the object into the knapsack then try object again

b) Add changes to the code above if the problem statement is extended to include the following: "there is an unbounded number of each object in the **s** array available"

```
Bool knapFit(int i, int s1, int s2) // i is item index <= n, s is capacity of knapsack

    If(s1==0 && s2==0) return true;   // both filled so true
    If(i==0 && (s1>0 || s2>0)) return false;  // not filled, no more object so false
    If(s1<0 || s2<0) return false;   // over filled
    return(knapFill(i-1, s1-s[i], s2) || knapFill(i-1, s1, s2-s[i]) || knapFill(i-1, s1, s2)
```

c) Add changes to the code if the problem statement is changed so there are two knapsacks size S_1 and S_2 and the function returns true if there exists some subset that exactly fill BOTH knapsacks. You can put an item in only one of the knapsacks. Each item can only be used a max of one time.