

```

/**
 * Knapsack problem
 * (solved by filtering pareo-optimal combinations)
 * (to actually get the item-combinations one needs to extend the
 * structure to hold the item-permutations, now it only holds sums
 * of weight and benefit, algorithm still works the same way though)
 */

#include "stdio.h"
#include "stdlib.h"

typedef struct{
    unsigned int weight;
    unsigned int benefit;
} element;
element null_element={0,0};

#define PSIZE 8

// example elements
element pool[PSIZE]={153,232},{54,73},{191,201},{66,50},{239,141},{137,79},{148,48},{249,38};

// element pool[PSIZE]={1,1},{4,2},{1,3},{3,1};

int eq(element *a, element *b) {
    if (a->weight != b->weight) return 0;
    if (a->benefit != b->benefit) return 0;
    return 1;
}

// is a more efficient than or equal to b ?
int eqme(element *a, element *b) {
    if (a->benefit < b->benefit) return 0;
    if (a->weight > b->weight) return 0;
    return 1;
}

void addElement (element *pl, element new) {

    element *e,*end,*insert;

    // find end of list
    for(e=pl;!eq(e,&null_element);e++) {}
    end=e;
    insert=e;

    // create duplicates of all existing elements
    for(e=pl;e!=end;e++,insert++) {
        *insert=*e;
    }

    // now shift the existing combinations
    for(e=pl;e!=end;e++) {
        e->weight += new.weight;
        e->benefit += new.benefit;
    }

    // and create one new element for the "combination" that
    // exists exactly of this single element
    insert->weight = new.weight;
    insert->benefit = new.benefit;
}

void optimize (element *pl) {

    element *e,*f,*c;

    // ZERO the combinations that have a better benefit/weight ratio
    e=pl;
    while (!eq(e,&null_element)) { // to the end of list

        // mark the non-optimal combinations for deletion
        for (f=pl;!eq(f,&null_element);f++)
            if ((e!=f) && eqme(f,e)) { e->benefit=0; break; }
        e++;
    }

    // compact the array (the first null_element marks the end of the list)
    e=f=pl;
    while (!eq(e,&null_element)) { // to the end of list
        if ( e->benefit != 0 ) { // look for valid ones
            if (e!=f) {

```

```

        f->benefit = e->benefit; // copy it down towards the array start
        f->weight = e->weight;
    }
    f++; // step forward (up to this pointer the array is compact)
}
e++;
}
while (!eq(f,&null_element)) {
    f->benefit=0; f->weight=0; // clear this place
    f++;
}
}

int main (void) {

    unsigned int i,c;
    element payload[PSIZE*PSIZE],*e;

    for (i=0;i<PSIZE*PSIZE;i++) payload[i]=null_element;

    for (i=0;i<PSIZE;i++) {
        addElement(payload,pool[i]);

        printf("-----\n");
        e=payload;
        while (!eq(e,&null_element)) { printf("%.3d/%.3d ",e->weight,e->benefit); e++; }
        printf("\n\n");

        optimize(payload);

        while (!eq(e,&null_element)) { printf("%.3d/%.3d ",e->weight,e->benefit); e++; }
        printf("\n");
    }
}

```