Programmieren C: Backtracking, Strukturen, `qsort`, File-I/O: "Rucksack packen"

Schreib ein Programm, das den *optimalen Inhalt eines Rucksacks* berechnet:

- Es ist eine *fixe Anzahl von Gegenständen* gegeben.
- Jeder Gegenstand hat ein *Gewicht* und einen *Wert*, beides vom Typ **`double`**, und sinnvollerweise einen *Namen*, z.B. "Fotoapparat" oder "Regenmantel".
- Weiters ist das *Maximal-Füllgewicht* des Rucksacks (ein **`double`**) gegeben.

  Dieses Maximalgewicht wird im Normalfall *kleiner* als die Summe der Gewichte aller Gegenstände sein, d.h. es können *nicht alle Gegenstände eingeladen* werden.
- Jeder Gegenstand kann entweder *ganz oder gar nicht* in den Rucksack gepackt werden, aber *nicht teilweise*.
- Eine Beladung ist dann *zulässig*, wenn die *Summe der Gewichte* der eingepackten Gegenstände das *Maximalgewicht* des Rucksacks *nicht überschreitet*.
- Der Wert einer Beladung ist die *Summe der Werte* aller eingepackten Gegenstände.
- Es soll jene Beladung ausgegeben werden, die *von allen zulässigen Beladungen den höchsten Wert* hat.

Für den ersten Versuch darfst du Gewicht und Wert aller Gegenstände fix ins Programm codieren (mittels *Initialisierung* in der Deklaration eines Arrays von Strukturen), das Maximalgewicht wird beim Programmaufruf auf der *Befehlszeile* angegeben.

Derartige Probleme können durch *systematisches Ausprobieren aller Möglichkeiten* mit Hilfe einer *rekursiven Funktion* gelöst werden.
Die **rekursive Lösungsidee (Backtracking)** ist in etwa Folgende:

- Man beginnt mit dem *leeren* Rucksack.
- Jede Ebene des rekursiven Aufrufs prüft *einen* Gegenstand:
  - Wenn der Gegenstand vom Gewicht her noch in den Rucksack *passt*, gibt man den Gegenstand *in den Rucksack* und macht einen *rekursiven Aufruf* für die restlichen Gegenstände.
  - Egal, ob der Gegenstand passt oder nicht, macht man einen *weiteren rekursiven Aufruf* für die restlichen Gegenstände, *ohne* den aktuellen Gegenstand in den Rucksack gepackt zu haben.
- Hat man *alle Gegenstände* entschieden (Rekursionstiefe = Anzahl der Gegenstände), prüft man, ob die aktuelle Beladung einen *höheren Wert* hat als die beste bisher bekannte Beladung. Wenn ja, speichert man sie als neue optimale Beladung.
- Nachdem der oberste Aufruf im Hauptprogramm zurückgekehrt ist, gibt man die gespeicherte optimale Beladung (Packliste, Gesamtgewicht und Gesamtwert) aus.

Gehe wie folgt vor:

- Deklariere einen **Strukturtyp**, der die Daten *eines* Gegenstandes enthält:
  - Seinen *Namen* (max. 31 Zeichen).
  - Sein *Gewicht* (eine Kommazahl).
  - Seinen *Wert* (auch eine Kommazahl).
  - Eine Ja-Nein-Variable, die anzeigt, ob der Gegenstand bei der *gerade probierten Beladungs-Variante* eingepackt wird oder heraußen bleibt.
  - Eine zweite Ja-Nein-Variable, die anzeigt, ob der Gegenstand bei der *besten bisher gefundenen Beladung* eingepackt wird oder heraußen bleibt.
- Deklariere ein **globales Array solcher Strukturen** und *initialisiere es fix* mit einigen Gegenständen samt Gewicht und Wert (die Ja/Nein-Variablen werden am Anfang *alle auf Nein* gesetzt).
- Deklariere weiters eine **globale `int`-Konstante**, die angibt, *wie viele Gegenstände* in deinem Array gespeichert sind (im Idealfall solltest du den Wert dieser Konstanten aus der Arraygröße *berechnen*, sodass er sich automatisch anpasst, wenn du bei der Initialisierung des Arrays neue Gegenstände hinzufügst oder Gegenstände entfernst).
- Schließlich brauchst du noch eine **globale `double`-Variable** für den *Gesamtwert der bisher besten Beladung*. Sie ist am Anfang **`0`**.
- Schreib weiters eine **Funktion**, die jedesmal aufgerufen wird, wenn man eine *neue gültige Beladung* ermittelt hat, d.h. für alle Gegenstände entschieden hat, ob sie eingepackt werden oder heraußen bleiben.

  Die Funktion hat den *Gesamtwert der eingepackten Gegenstände als Parameter* und keinen Returnwert.

  Ist der übergebene Gesamtwert *größer* als der in der globalen Variable gespeicherte Gesamtwert der besten bisher gefundenen Beladung, so wird erstens der *neue Wert in der globalen Variablen* für die beste Beladung gespeichert und zweitens in einer Schleife für alle Elemente des Arrays *die aktuelle Auswahl (Ja/Nein) als bisher beste Auswahl gespeichert*.

  Ist der angegebene Gesamtwert *kleinergleich* dem bisher besten Gesamtwert, kehrt die Funktion sofort zurück, ohne etwas zu tun:
  Die neue Beladungsvariante ist schlechter und wird *ignoriert*.
- Jetzt kommt die **rekursive Funktion zum Durchprobieren** aller Fälle.

  Die Idee ist folgende:
  - Jeder Aufruf probiert *beide Möglichkeiten* (einpacken oder heraußen lassen) *für **einen** Gegenstand* im Array.
  - Für die *Gegenstände weiter vorne* im Array (mit kleinerem Index) ist *schon festgelegt*, ob sie eingepackt werden oder heraußen bleiben.
  - Zum Durchprobieren der Möglichkeiten für die *Gegenstände weiter hinten* im Array wird die *Funktion rekursiv für den nächsten Gegenstand aufgerufen*.

Im Detail: Die Funktion hat *3 Parameter und keinen Returnwert*:
- Einen **`int`**: Den *Index* des zu probierenden Gegenstandes im Array.
- Einen **`double`**, der angibt, *für wieviel Gewicht im Rucksack noch Platz frei* ist.
- Noch einen **`double`**: Die *Summe des Wertes* der bisher *eingepackten* Gegenstände.

Als erstes prüft die Funktion, ob *schon alle Gegenstände entschieden* sind
(wie erkennt man das an der Nummer des zu probierenden Gegenstandes?).
Wenn ja, ruft sie die *oben beschriebene Funktion* zur Prüfung
auf eine neue optimale Beladung auf und kehrt dann gleich zurück.

Dann prüft die Funktion, ob der *aktuelle Gegenstand noch in den Rucksack passt*.
Wenn ja, wird er *als eingepackt markiert*.
Dann wird die Funktion *rekursiv für die restlichen Gegenstände*
(d.h. mit der *nächsten* Gegenstands-Nummer) aufgerufen.
Welche Werte musst du dem rekursiven Aufruf
für das *noch freie Gewicht* und die *Summe des Wertes* übergeben,
wenn du gerade einen Gegenstand dazugepackt hast?

Und zuletzt muss die Funktion in jedem Fall noch die *andere Möglichkeit* probieren
(egal, ob der Gegenstand hineingepasst hat oder nicht):
Den aktuellen Gegenstand *als nicht eingepackt markieren*
und wieder einen *rekursiven Aufruf* zum Probieren aller Möglichkeiten
der restlichen Gegenstände machen (da der Gegenstand ja *nicht* eingepackt wurde,
ändert sich diesmal am noch freien Gewicht und am Wert
der eingepackten Gegenstände *nichts*).

- Zuletzt kommt das **Hauptprogramm**:
  Es wird mit einer *Kommazahl auf der Befehlszeile* aufgerufen,
  nämlich dem *maximalen Beladungsgewicht* des Rucksackes
  (prüfe, ob wirklich eine *positive* Zahl angegeben wurde!).

  Dann wird die *Funktion zum Durchprobieren* aller Möglichkeiten aufgerufen,
  und zwar für den *ersten* Gegenstand im Array sowie mit der auf der Befehlszeile
  angegebenen Zahl für das freie Gewicht und mit 0 für den bisherigen Gesamtwert
  der schon eingepackten Gegenstände (es wurde ja noch nichts eingepackt).

  Zuletzt geht das Hauptprogramm das Array mit einer *Schleife* durch und
  gibt *genau die Gegenstände* aus, die *bei der optimalen Beladung eingepackt* werden.
  Summiere dabei auch den *Gesamtwert* und das *Gesamtgewicht* der Beladung auf
  und gib sie aus.

Erweiterungen

1. Baue einen *Aufrufzähler* (welche Art von Variable?) in dein Programm ein und
   gib am Ende des Programmes aus, wie viele rekursive Aufrufe gemacht wurden.
2. Versuche, die Anzahl der rekursiven Aufrufe durch zusätzliche Programmlogik zu
   reduzieren. Dazu gibt es viele Strategien, eine recht gute *Optimierung* ist folgende:
   - In der *Struktur* für jeden Gegenstand wird *zusätzlich* sein *Wert pro Gewicht*
     (also sein Wert dividiert durch sein Gewicht) gespeichert.

- Das Array aller Gegenstände wird *vor Beginn* der Lösungssuche nach diesem Wert pro Gewicht *sortiert* (z.B. **`qsort`** verwenden!), und zwar so, dass die *in Relation zum Gewicht wertvolleren Gegenstände zuerst* betrachtet werden.
- Bevor man prüft, ob man den nächsten Gegenstand noch in den Rucksack gibt, berechnet man jedesmal eine *obere Schranke für den maximal noch möglichen Wert* der aktuellen Beladung: Diese ist "Wert der bisher im Rucksack befindlichen Gegenstände" + "Wert pro Gewicht des aktuellen Gegenstandes" * "Restkapazität des Rucksacks". (Warum?)
- Ist diese *obere Grenze* für den *größtmöglichen Wert* der aktuellen Beladung *kleinergleich* dem globalen Wert der *bisher besten gefundenen Beladung*, kann man *sofort zurückkehren*, ohne diesen und weitere Gegenstände zu betrachten: Der aktuelle Aufruf kann *keine bessere Beladung* als die schon bekannte mehr liefern.

3. Lies die Werte und Gewichte der *Gegenstände aus einem File* ein, anstatt sie fix im Programm vorzugeben:
   - Der *Filename* wird auf der *Befehlszeile* angegeben.
   - Der File enthält *pro Gegenstand eine Zeile* mit *Name, Gewicht und Wert* (verwende **`fscanf`**, das Format für **`double`**-Werte ist **`%lf`**).
   - Du darfst in deinem Programm ein *globales Array fixer Größe* und damit eine *fixe obere Schranke* für die *Anzahl* der Gegenstände codieren (prüfe bei jedem Lesen, ob sie überschritten wird!).
     Es ist nicht notwendig, statt dem Array fixer Größe für die Gegenstände ein beim Lesen mittels **`realloc`** wachsendes Array oder eine verkettete Liste zu bauen (das wäre die Super-Extra-Lösung). Auch für die *Namen* der Gegenstände darfst du eine *fixe Maximallänge* vorgeben.
   - Du solltest *Formatierungsfehler* im Eingabefile erkennen und melden, aber du musst nicht versuchen, sie zu überspringen und weiterzulesen. (Wie erkennt man, wie viele Werte das **`scanf`** erfolgreich gelesen und gespeichert hat? Wie erkennt man, dass die Eingabedatei zu Ende ist?)