Extra-Übung AIK: "Rucksack packen"

Gegeben sind mehrere *Gegenstände*. Jeder Gegenstand hat ein *Gewicht* und einen *Wert*.

Weiters ist das *maximale Beladungs-Gewicht* eines Rucksacks (oder eines Lastwagens, …) gegeben. Dieses Maximalgewicht wird im Normalfall *kleiner* als die Summe der Gewichte aller Gegenstände sein, d.h. es können *nicht alle Gegenstände eingeladen* werden.

Gesucht ist ein Programm, das die *optimale Beladung* ermittelt, d.h. jene *Auswahl von Gegenständen*, deren *Gesamtgewicht kleinergleich dem Maximalgewicht* ist und deren *Gesamtwert möglichst hoch* ist.

Derartige Probleme werden durch *systematisches Ausprobieren aller Möglichkeiten* mit Hilfe einer *rekursiven Funktion* gelöst.

Gehe wie folgt vor:

- Deklariere einen **Strukturtyp**, der die Daten *eines* Gegenstandes enthält:
  - Seinen *Namen* (max. 31 Zeichen).
  - Sein *Gewicht* (als 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* 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).
- 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 Möglichkeit* 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 Lösung 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.
  - Die *Gegenstände weiter vorne* im Array (mit kleinerem Index) sind *schon festgelegt*.
  - 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 Lösung eingepackt* werden (gib auch den *Gesamtwert* und das *Gesamtgewicht* der Beladung aus!).

Anmerkung:
Bei den Studenten gibt es dieselbe Aufgabe mit Zusatzaufgaben:
- Mitzählen der rekursiven Aufrufe
- Optimierung der Gegenstands-Reihenfolge zur Reduzierung der Aufrufs-Zahl
- Einlesen der Gegenstands-Daten aus einer Datei