Christoph Lüth <cxl>
Felix Beckwermert <foetus>
Michael Skibbe <mskibbe>
Dennis Walter <dw>
Rafael Trautmann <pirate>
Thomas Meyer <mclee>

2. Übungsblatt

Ausgabe: 04.11.02

Bearbeitungszeit: Zwei Wochen

3 *Kauf Dich Glücklich!* *10 Punkte*

Als frischdiplomierter Informatiker finden Sie Brot und Butter bei der Kaufhauskette *Ellstadt* (Motto: *Kauf' Dich Glücklich!*). Als erste Aufgabe steht dabei die Sanierung des firmeneigenen Lagerhaltungssystem an, dessen momentane Implementation noch aus der Steinzeit der Informatik stammt, als Rechner noch ganze Keller füllten und in COBOL programmiert wurden.

Das Lagerhaltungssystem soll das Artikellager der Firma verwalten. Dabei wird jeder Artikel durch eine eindeutige Artikelnummer gekennzeichnet. Eine Datenbank ordnet jeder dieser Nummern einen Namen und einen Preis (in Cent) zu. Das Warenlager ist dann ein Liste von Artikelnummern und die Anzahl der vorrätigen Artikel dieser Nummer. Das gesamte Kaufhaus[1] besteht aus der Datenbank, dem Warenlager, und dem Geldbestand:

```
type Name  = String
type Price = Int
type Code  = Int

type Database  = [(Code, Name, Price)]
type Warehouse = [(Code, Int)]
type Store     = (Database, Warehouse, Int)
```

1. Implementieren Sie zuerst die Datenbank mit den Funktionen

   ```
   lookupPrice :: Database -> Code-> Int
   lookupName  :: Database -> Code-> String
   ```

   welche den Namen und Preis eines Artikels liefern.

2. Implementieren Sie eine Funktion

   ```
   sell :: Store-> [Code]-> Store
   ```

   die eine Lieferung (bestehend aus einer Liste von Artikelnummern) verkauft, d.h. die Artikel werden aus dem Warenlager ausgetragen, und ihr Preis wird dem Bargeldbestand zugerechnet. Beachten Sie die Rand- und Fehlerfälle.

3. Implementieren Sie eine Funktion

   ```
   prnt :: Store-> String
   ```

[1] Wir abstrahieren hier einmal von der Tatsache, dass die Firma *Ellstadt* 73 Filialen in Deutschland und zudem eine Mehrheitsbeteiligung an einer Kaufhauskette in Südschweden besitzt.

die den Waren- und Bargeldbestand formatiert zurückgibt. Dabei sollen die Artikelnamen linksbündig in der ersten Spalte, und der Bestand rechtsbündig in der zweiten Spalte aufgereiht werden. Der Bargeldbestand wird darunter angegeben, natürlich in Euro und nicht Cent.

4. Der Marketingleiter von *Ellstadt* möchte wissen, welche Artikel zur Zeit besonders gut laufen. Erweitern Sie die Datenstrukturen so, dass Sie mittels folgender Funktion eine Übersicht über die Verkäufe geben können:

   ```
   salesChart :: Store-> String
   ```

   Die Funktion soll entweder eine (sehr einfache) Balkengraphik der Verkäufe zeichnen, oder die Artikel in Reihenfolge absteigenden Umsatzes ausgeben.

Tip: Wenn Funktionen Zeichenketten zurückgeben, werden diese auf der Kommandozeile formatiert angezeigt, und nicht ausgegeben:

```
Prelude> "\n" ++ "Hallo" ++ "\n"
"\nHallo\n"
```

Mit Hilfe der Funktion `putStrLn` können (nur auf der Kommandozeile!) Zeichenketten ausgegeben und nicht nur angezeigt werden:

```
Prelude> putStrLn ("\n" ++ "Hallo" ++ "\n")

Hallo
```

$\boxed{4}$ *Bauer Graham und sein Verfahren* — *10 Punkte*

Als der Farmer Ernest Mathew Graham 1853 in der damals noch weitgehend unbesiedelten Prärie östlich von Dead Man's Gulch, Arizona, seine Farm errichtete, stand er vor einem Problem. Damit seine Rinder nicht in den weiten Wilden Westen davonlaufen, musste er sein Weideland natürlich einzäunen. Nun war Stacheldraht allerdings teuer, und deshalb sollte der Zaun möglichst kurz sein, dabei aber alle wichtigen Punkte (Tränke, Ställe usw) des Weidelandes umfassen.

Graham, in seinem Vorleben Informatiker an der Universität Heidelberg bis er 1849 wegen der Teilnahme an den Märzunruhen 1848/49 vom Kurfürst persönlich vor die Wahl gestellt wurde, seinen Lehrstuhl entweder freiwillig oder durch vorzeitiges Ableben für eine Neubesetzung zur Verfügung zu stellen, konnte wegen einer während jener Märzunruhen erlittenen Fußverletzung nicht allzu weit laufen, und mußte daher ein Verfahren ersinnen, dass möglichst effizient ist.

Abstrakt gesehen handelt es sich hierbei um das Problem, für eine Menge von Punkten die sogenannte *konvexe Hülle* zu finden. Das von Graham ersonnene Verfahren, der *Graham scan*, funktioniert in zwei Schritten:

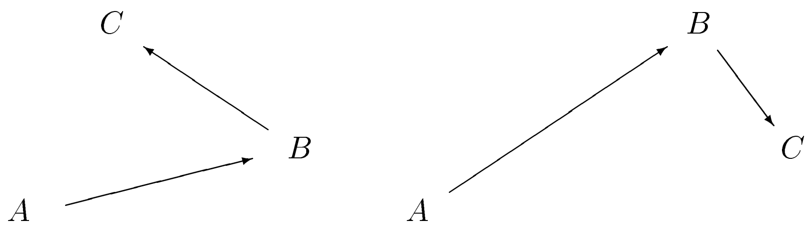
1. Gegeben eine Menge von Punkten P konstruieren wir zuerst ein einfaches Polygon Q durch alle Punkte von P. Dabei heißt ein Polygon *einfach*, wenn sich keine Kanten überschneiden.
2. Danach werden alle konkaven Winkel aus Q entfernt.

Implementieren Sie eine Funktion

```
convexHull :: [Point]-> Poly
```

welche die konvexe Hülle mittels Graham scan berechnet.

Zentrales Hilfsmittel hierbei ist die *Orientierung* von Punkten. Sei $P = (x, y)$ ein Punkt, dann schreiben wir x_P für seine x-Koordinate und y_P für seine y-Koordinate. Zentral für die folgenden Betrachtungen ist die *Orientierung* eines Punkttripels (A, B, C). Wir sagen (A, B, C) ist *positiv orientiert*, wenn die Punkte A, B und C gegen den Uhrzeigersinn durchlaufen werden (1, links), und *negativ*, wenn dies nicht der Fall ist (1, rechts).



(1)

Die Orientierung kann durch die Determinante $D_{A,B,C}$ bestimmt werden:

$$\begin{aligned} D_{A,B,C} &= (y_B - y_A)(x_C - x_B) - (y_C - y_B)(x_B - x_A) \\ &= y_B x_C - y_A x_C + y_A x_B - y_C x_B + y_C x_A - y_B x_A \end{aligned}$$

Falls $D_{A,B,C} < 0$, dann ist die Orientierung positiv, und falls $D_{A,B,C} > 0$, dann ist die Orientierung negativ. Falls $D_{A,B,C} = 0$, liegen A, B und C auf einer Geraden.

Hinweise:

1. Zerlegen Sie das Problem in zwei Funktionen

```
simplePoly :: [Point]-> Poly
scan       :: Poly-> Poly
```

2. Um ein einfaches Polygon zu konstruieren, wählen Sie als Bezugspunkt zum Beispiel den linken, unteren Punkt P der Liste, und sortieren die restlichen Punkte der Liste entgegen dem Uhrzeigersinn, d.h. aufsteigend bezüglich einer wie folgt definierten Ordnung auf den Punkten:
$$Q < R \Leftrightarrow D_{P,Q,R} < 0$$

3. Um alle konkaven Winkel zu entfernen, wird die eigentliche konvexe Hülle schrittweise aufgebaut. Angefangen mit den ersten drei Punkten des simplen Polygons in umgekehrter Reihenfolge wird der Punkt des simplen Polygons wird zu der konvexen Hülle hinzugefügt, danach wird diese *normalisiert*:

   - Wenn die konvexe Hülle weniger als drei Punkte enthält, ist sie normalisiert;
   - Wenn die konvexe Hülle drei Punkte oder mehr enthält, und die Orientierung der ersten drei Punkte ist positiv (oder null), dann entferne den mittleren Punkt (dann wird dieser nämlich von einem konkaven Winkel eingeschlossen), und normalisiere weiter;
   - Wenn die konvexe Hülle drei Punkte oder mehr enthält, und die Orientierung der ersten drei Punkte ist negativ, dann ist sie normalisiert.

   Man beachte, dass sich dadurch, dass von dem simplen Polygon von vorne abgenommen, und vorne an die konvexe Hülle angefügt wird, die Reihenfolge der Punkte umkehrt. Man kann natürlich auch hinten anfügen (das ist allerdings ineffizienter); dann dreht sich die Orientierung um, d.h. wenn die Orientierung der ersten drei Punkte negativ ist, wird ein konkaver Winkel eingeschlossen.

Das Sortieren hat hierbei den Aufwand $O(n \log n)$. Der Zweite Schritt, die Normalisierung, hat linearen Aufwand, da jeder Punkt einmal in die Hülle aufgenommen und höchstens einmal wieder herausgenommen werden muß. Damit ist der Gesamtaufwand des Graham scan $O(n \log n)$.

Dieses ist Version 1.0 von 4. November 2002.