0. Zur Dokumentation

Da diese Aufgabe keinen Quellcode fordert, werde ich stattdessen versuchen meine Lösung und die Überlegungen, die zur ihr geführt haben, darzulegen.
Dabei werde ich Pseudo-C(++)-Code benutzen um die Beziehungen darzustellen, da Quellcode exaktere Aussagen ermöglicht als Fließtext.

1. Überlegungen zu den Teilaufgabe

1) Erstellen eines Kassenbons:

Ein Kassenbon muss folgende Informationen enthalten (als Ausgangspunkt habe ich einen LIDL-Kassenbon benutzt):

- Filiale
- Adresse der Filiale
- Datum und Uhrzeit

- eine Tabelle mit folgenden Inhalten:
  - Produktname
  - Gesamtpreis des Produkts (wenn mehrere Artikel gekauft werden, oder abgewogen werden muss, bzw Gutschrift bei Gesamtpreis < 0 bei der Rückgabe von Waren)
  - Anzahl der gekauften Artikel (bei der Rückgabe von Waren, wird hier eine negative Anzahl angegeben)
  - (bei nicht abgepackter Ware) Gewicht
  - Stückpreis (bzw Preis pro Stück)
  - (oder bei nicht abgepackter Ware) Preis pro kg

- Anzahl der Positionen auf der Liste (also Einträge in der Tabelle)
- Anzahl der gekauften Artikel (also Summe über "Anzahl der gekauften Artikel)
- Gesamtpreis
- wurde BAR oder mit einer Kundenkarte bezahlt? (EC-Karten würde das Modell unnötig verkomplizieren, da diese auch noch ein Unterschriftenfeld und weitere Daten benötigen würden, zB PIN)
- wenn BAR, dann vom Kunden gegebene Geldmenge, und Rückgeld
- wenn Kundenkarte, dann Kundenidentifikationsnummer (KundenID)

Einträge zur Mehrwertsteuer lasse ich weg, da auf Lebensmittel ja generell die gleiche Lebensmittelsteuer angewendet wird.

2) Ausdruck einer Liste aller Artikel, deren Bestand einen bestimmten Wert unterschreitet.

Der Ausdruck sollte folgenden Informationen enthalten:

- Artikelbestandswert, der von den folgenden Artikeln unterschritten wird

- eine Tabelle mit folgenden Inhalten:
  - Artikelname
  - Artikelanzahl (um zu sehen, wie akut der Bestandmangel im Lager ist)

3) Ausdruck einer Hitliste verkaufter Artikel, geordnet nach Produktgruppen

Der Ausdruck sollte folgende Informationen enthalten:

- Name des Monats für den die Hitliste erstellt wird
- Liste der Produktgruppen und Menge aller verkauften Produkte in einer

Produktgruppe
- Aufschlüsselung der Produktgruppen in Produkte und Produktverkäufe

4) Ausdruck von Adressetiketten für den Versand eines Werbebriefes

Die Etiketten müssen folgende Daten enthalten:

- Name des Kunden
- Anschrift des Kunden

2. Implikationen für die Datenstrukturen

Für den Kassenbon wäre also folgende Struktur empfehlenswert:

```
struct Person {
	string name;
	string anschrift;
};

// es gibt genaue Bestimmungen zur Rundung, deshalb sollte ein Typ Währung
(Currency) definiert sein
// der dies berücksichtigt
typedef "X" Währung;

// wir brauchen auch einen Typen, der die Kundenidentifikationsnummer enthält
(Nummer der Kunderkarte)
typdef "X" KundenID;

struct Bon {
	// Name und Anschrift der Filiale werden in der Filialen Struktur
gespeichert
	// Zum Feststellen von Zeit und Datum gibt es Systemfunktionen

	// mit vector spiele ich auf std::vector an, und meine damit ein Array
variabler Größe
	vector< BonEintrag > einträge;

	// = einträge.size()
	unsigned anzahlPositionen;
	// = summe über eintrage[ i ].anzahlArtikel
	// ich benutze int (also signed statt unsigned), da ja vielleicht ein
Kunde schon gekaufte Waren
	// zurückgeben will
	int anzahlArtikel;

	Währung gesamtPreis;

	enum BezahlArt {
		BA_BAR,
		BA_KUNDENKARTE
	};

	BezahlArt bezahlArt;

	union {
		struct {
			Währung geld, rückgeld;
		} bar;
		struct {
			KundenID kundenID;
		} kundenKarte;
```

```
	};
};

// wir brauchen auch einen Typ, der den erwarteten Strichcode des Scanners
darstellt
typedef "X" ScannerCode;

// allgemeine Informationen zu einem Produkt
// diese könnten vielleicht hin und wieder von einem WAN-Server der
Supermarktkette heruntergeladen
// werden, auf jeden Fall sind diese Informationen nicht filialen-spezifisch
struct Produkt {
	string bezeichnung;

	unsigned produktNummer;
	// der der produktNummer entsprechende strichcode
	ScannerCode scannerCode;

	enum Typ {
		T_ABGEPACKT,
		T_ABGEWOGEN
	};

	Typ typ;
	union {
		Währung preisProStück;
		Währung preisProKG;
	};
};

// FilialenProduktInformation sollte schon deklariert werden, da es gleich
gebraucht wird, aber
// die Definition spare ich mir für später auf
struct FilialenProdukt;

// wir brauchen auch einen Typ, der das Gewicht speichert (< 0 bei Rückgabe,
obwohl sich hier die
// Frage stellt, ob das Geschäft abgewogenes zurücknehmen kann)
typedef "X" Gewicht;

struct BonEintrag {
	// statt einem Zeiger, kann auch ein absolute Index in ein Array, oder
etwas ähnliches gemeint
	// sein, ich verwende die Zeigerdarstellung nur um klar zu machen, dass es
sich um einen
	// Verweis auf eine spezifische Instanz handelt
	FilialenProdukt *produkt;

	Währung gesamtPreis;

	union {
		// < 0 bei Rückgabe eines abgewogenen Artikels
		Gewicht gewicht;
		// < 0 bei Rückgabe eines abgepackten Artikels
		int anzahlArtikel;
	};
};
```

Für die Hitliste und den Ausdruck der knappen Artikel (3) und 2)) ergeben sich die nachfolgenden Strukturen, wobei ich annehme, dass man bei der Hitliste an der Anzahl der verkauften Produkte in den letzten 30 Tagen (Wirtschaftsmonat)

interessiert ist. Ich schließe aus der Verwendung des Wortes "innerhalb eines Monates" in der Aufgabenstellung, ansonsten hätte "für ein Monat" ausgereicht.

```
const unsigned ANZAHL_TAGE = 30;

struct Statistik {
      // das Array wird jeden Tag um 1 nach links verschoben und der erste Tag
fällt damit weg
      // neue Einträge werden
      int anzahlProTag[ ANZAHL_TAGE ];

      int gesamtAnzahl;
}:

// auch hier kommt zuerst wieder eine allgemeine, filialen-unabhängige Version,
die nur
// statische Daten enthält
struct ProduktGruppe {
      string name;

      // mit set spiele ich auf std::set an und meine allgemein einen
automatisch sortierten Container
      // variabler Größe
      // Produkte der Gruppe
      set< Produkt* > produkte;
};

// FilialenGruppe soll standardmäßig nach gruppenVerkaufsStatistik.gesamtAnzahl
absteigend sortiert werden
struct FilialenGruppe {
      ProduktGruppe *gruppe;

      // Produkte der Gruppe
      // nach ->verkaufsStatistik.gesamtAnzahl automatisch sortieren
      set< FilialenProdukt* > produkte;

      Statistik gruppenVerkaufsStatisik;
};

// soll standardmäßig nach verkaufsStatistik.gesamtAnzahl absteigend sortiert
werden
struct FilialenProdukt {
      Produkt *produkt;

      unsigned stückImLager;
      Statistik verkaufsStatistik;
};
```

Und nun zu 4:
Da der Hersteller von Software normalerweise nicht weiß, welche Geschäftsfälle genau auftreten, sondern versuchen für alle Eventualitäten gewappnet zu sein, werde ich hier auch versuchen eine möglichst generische Datenstruktur zu entwickeln, die nicht nur bei Werbebriefen für Weinliebhaber genutzt werden kann.

```
struct KundenKarte {
      KartenID kartenID;

      Person kunde;

      // map bezieht sich auf std::map, und weist einem bestimmen Schüssel ein
```

```
Datenelement zu.
      // falls die Statistik nur für einen bestimmten Kunden interessant ist
      map< FilialenProdukt, int > gekaufteProdukte;
      // nach Gruppen geordnet, falls es eine Gruppe "Weine" gibt, wird dadurch
schon der Fall 4) sehr
      // vereinfacht
      map< FilialenProduktGruppe, int > gekauftVonProdukteGruppe;
};

Und zum Abschluss noch die Struktur, die alles speichert:

struct Filiale {
      Person info;

      set< Produkt > produkte;
      set< ProduktGruppen > produktGruppen;

      set< FilialenProdukt > filialenProdukte;
      set< FilialenProduktGruppen > filialenProduktGruppen;

      set< KundenKarten > kundenKarten;
};
```

Die Aufteilung in statische einerseits und dynamische, filialen-spezifische Datenstrukturen andererseits, hat den Vorteil, dass, falls es sich um eine Supermarktkette handelt, die Daten in verschiedenen Dateien gespeichert werden können und bei Preisänderungen ganz einfach die Datendatei neu bezogen werde und in den Speicher geladen werden müsste. Die Datenstrukturen unterstützen diese Herangehensweise. Die Strukturen enthalten explizit alle Daten, die zum Erstellen der Ausdrücke nötig sind, d.h. der Algorithmus zum Erstellen der Strukturen muss Sorge dafür tragen, die Daten richtig zu bestimmen.

3. Erstellung der Ausdrücke

1) Erstellen des Kassenbons:

Zum Erstellen des Kassenbons wie er in 1. beschrieben ist, muss man also Filiale.info ausgeben (Filialenname und Anschrift) und dann den Rest der Daten in Bon. Bei den Einträgen muss der Verweis auf FilialenProdukt aufgelöst werden um auf den Preis pro kg, etc. zugreifen zu können.

2) Erstellen der Liste der Artikel, deren Bestand einen gewissen Wert X unterschreitet:

Es wird durh alle Elemente von filialenProdukte iteriert und diejenigen für die stückImLager < X ist werden zum Ausdruck hinzugefügt.

3) Erstellen der Hitliste

Es wird durch filialenProduktGruppen, das standardmäßig nach gruppenVerkaufsStatistik.gesamtAnzahl absteigend sortiert ist, iteriert und der Name und gruppenVerkaufsStatistik.gesamtAnzahl entsprechend ausgegeben. Dann wird jeweils durch produkte der FilialenProduktGruppe iteriert und die gesamtAnzahl der verkaufsStatistik ausgegeben.

4) Ausdrucken der Adressetiketten

Es wird einfach durch kundenKarten iteriert und für jeden Kunden festgestellt, wie viele Weine er gekauft hat, entweder durch gekauftVonProdukteGruppe, falls es eine Gruppe "Weine" gibt, oder durch Summieren von gekaufteProdukte für die verschiedenen Produktnamen der Weine.

4. Bewertung des Geschäftsfalles #4

Betrachten wir zuerst die Vorteil:
Im Vergleich zu normalen Massenwerbesendungen hat eine solche zielgerichtete Werbesendung einige Vorteile.
Zuerst einmal hat sie höhere Erfolgschancen, da die Wahrscheinlichkeit größer ist, dass die Adressaten an den Angeboten interessiert sind, da sie aufgrund ihres Kaufverhaltens ausgewählt wurden. Normalerweise haben Massenversendungen einen relativ geringen Wirkungsgrad, da die meisten Empfänger nicht daran interessiert sind. Hier liegt der Fall anders, da nur 'bekannte' Weinkäufer angeschrieben werden.
Das Unternehmen spart dadurch Kosten und ist gleichzeitig effizienter, die Kunden sind vielleicht über die Informationen erfreut, da sie automatisch darüber benachrichtigt wurden und begrüßen den Service, der auf sie "speziell" zugeschnitten ist.
Andererseits sammelt aber der Supermarkt, dadurch sehr spezielle Kundendaten (- in meiner Lösung oben werden ja alle Einkäufe gespeichert um Datenmaterial für solche spezifischen Auswertungen zu bieten -) wahrscheinlich ohne direktes Wissen des Kunden. Normalerweise werden solche Daten höchstens anonymisiert erhoben, um nicht gegen Datenschutzbestimmungen zu verstoßen.
Die Frage, die sich stellt, ist: Wie "gläsern" wollen die Kunden sein?
Ist es wünschenswert, dass so viele Informationen über das Kaufverhalten erhoben werden?
Wie verhält das Unternehmen mit den Daten - gibt es sie vielleicht weiter an andere Firmen?
Insgesamt kann man diesen Geschäftsfall und die damit verbundenen Datenerhebungen gemischt bewerten:
Das Endresultat an sich ist effizienter und für alle vorteilhafter, der Weg dorthin ist aber fragwürdig, da nicht sicher ist, in wie weit die Kunden mit den Erhebungen einverstanden sind und ob sie überhaupt davon in Kenntnis gesetzt wurden, als sie ihre Kundenkarte bekamen.