Aufgabe 2: Fallunterscheidungen Prädikatsfunktionen Gemischte Daten

Allgemeine Regeln

- Die in der Vorlesung besprochenen **Vorgehensweisen** und **Programmierregeln** müssen angewendet werden.

- Das gilt immer, auch ohne explizite Erwähnung.

- Oft gibt es eine kompaktere Lösung mit "**fortgeschritteneren**" Programmkonstrukten, als mit den oft "**elementaren**" deren Anwendung in den jeweiligen Aufgaben geübt werden soll.

- Das ist mir durchaus bekannt.

- Sie sollen aber lernen, was mit einem vorgegebenen "Werkzeugkoffer" möglich ist.

- Oft hat man es in der Praxis mit vorgeschriebenen, aber leider etwas "**ausdrucksschwachen**" Programmiersprachen zu tun.

  Das ist manchmal etwas mühsam. Aber auch damit muß man umgehen können.

- Also halten Sie sich bitte an die Vorgaben für die jeweils zulässige **Verwendung von Sprachkonstrukten**.

- Es wird Ihren Lernfortschritt und die dazu notwendige Tiefe des Verständnisses befördern.

Binäre Fallunterscheidungen mit (_?_:_)

Verwenden Sie **Konditionaloperatoren** (auch wenn es anders geht)

Minimum und Maximum

Schreiben Sie jeweils eine Funktion

min_int(int1, int2) bzw.

max_int(int1, int2)

mit jeweils zwei Integer-Argumenten, welche das Minimum bzw. das Maximum der Argumente liefert.

Element eines Intervalls

Schreiben Sie eine 3-stellige Prädikatsfunktion

within?(val, lower, upper)

mit Integer-Argumenten, die prüft, ob val innerhalb des Zahlenbereichs lower bis upper (jeweils einschließlich) liegt.

Prädikatsfunktionen

Definition einiger Prädikate für Temperaturen

Schreiben Sie **vier Prädikatsfunktionen**, die Temperaturen (als Integer kodiert) klassifizieren.

zu_kalt?(temp),
zu_warm?(temp),
angenehm?(temp),
unangenehm?(temp)

Dabei soll gelten, daß Temperaturen zwischen

16 und **22 Grad einschließlich**

als **angenehm** empfunden werden.

Hinweise:

Diese Prädikate definieren jeweils Teilmengen der Menge der Temperaturen.

Verwenden Sie **logische Verknüpfungen** für eine kompakte Notation !

Übungen zur Transformation von Prädikaten

Transformieren Sie die Prädikatsfunktion

angenehm()

in mehrere **äquivalente Formen.**

Das geht am einfachsten, indem Sie die Formen als **Kommentare** schreiben, und nur jeweils eine Form für den Test **aktiv** schalten.

- mit **and** (ohne **or** aber evtl. mit **not**)
- mit **or** (ohne **and** aber evtl. mit **not**)
- nur mit **Konditionaloperator** (ohne jegliche logische Verknüpfung, also ohne and, or, not)

Testen Sie diese Transformationen !

Das nennt man einen **Regressionstest.**

Man prüft damit, ob nach einer **Programmtransformation** oder nach einer **Programmerweiterung** noch alles funktioniert.

Komplexere Fallunterscheidungen

Summe von größten Quadraten

Schreiben Sie eine Funktion, die drei Integer **konsumiert** und die Summe der Quadrate der beiden größeren Zahlen **produziert**:

larger_sum_square(val1, val2, val3)

Hinweise zur Vorgehensweise

- Sie dürfen ihre vorher selbst programmierten Funktionen verwenden.

  Das macht manches einfacher und vor allem **lesbarer**.

- Das eigentliche Problem besteht darin, die beiden größten Elemente **{max1,max2}** aus der Menge **{val1,val2,val3}** zu finden.

- Also "**wünschen**" Sie sich eine **Hilfsfunktion**, die dieses leistet, und verwenden diese.

  Das abschließende quadrieren ist dann nur noch ein "Peanuts"-Problem.

- Die Hilfsfunktion muß ein "Paar" von Werten produzieren. Dieses Paar können Sie (wie bei der Clock-Aufgabe) mit dem **[ ]-Operator** erzeugen.

Hinweise zur Konstruktion von Testfällen (Pfadabdeckung)

- Entwickeln Sie eine **kompakte** Menge von Testfällen, die alle strukturell möglichen **Kombinationen** abdecken.
- Fragen: Wie viele sind das? Wie konstruiert man diese Menge ?
- Mit diesen Testfällen würden alle alternativen "**Pfade**" durch ihr Programm wenigstens einmal durchlaufen.
- Das nennt man beim Testen "**Pfadabdeckung**".
- Dieses ist oft wegen der "**kombinatorischen Explosion**" der Möglichkeiten kaum zu erreichen.
- Hier ist es aber noch einfach machbar.
- Also üben wir das mal.

Funktionen auf gemischten Daten

Gemischte Daten: Zwei Repräsentationen von Wochentagen

- Repräsentation durch die Zahlen von 1 für Montag bis 7 für Sonntag (DayNum)

  DayNum ::= Nat :: (1..7)

- Repräsentation durch die Symbole :Mo für Montag bis :So für Sonntag (DaySym)

  DaySym ::= {:Mo, :Di, :Mi, :Do, :Fr, :Sa, :So}

- Die Repräsentationen sind also Teilmengen von Nat bzw. Sym

- Der **gemischte Typ** Day ist die **disjunkte Vereinigung** (oder „**Summe**“) von beiden.

  Day ::= (DayNum | DaySym)

Schmale und breite Typprädikate

day_num? ::= Any -> Bool

day_sym ::= Any -> Bool

day? ::= Any -> Bool

Das Prädikat day? fragt also, ob ein Datenobjekt entweder zu DayNum oder zu DaySym gehört.

Schmale und breite Konversionsfunktionen

Spezifizieren und implementieren Sie **schmale** Konversionsfunktionen zwischen den Typen

day_num_to_day_sym ::=

day_sym_to_day_num ::=

Spezifizieren und implementieren Sie **breite** Konversionsfunktionen, die beide Repräsentationen konsumieren können

to_day_sym ::=

to_day_num ::=

Schmale und breite Funktionen für Nachfolger und Vorgänger

Spezifizieren und implementieren Sie je zwei Funktionen für jede Repräsentation, die (zyklisch) jeweils den vorigen oder nächsten Tag berechnen.

day_num_succ ::=

day_sym_succ ::=

day_sym_succ ::=

day_sym_pred ::=

Die breiten Funktionen sollen jede Repräsentation (d.h. Day) konsumieren können, und das Ergebnis in der jeweiligen Repräsentation der Eingabe abliefern.

Dadurch haben wir nur noch zwei Funktionen:

day_succ ::=

day_pred ::=