☐ **Gr. 1,** Dr. D. Auer
☐ **Gr. 2,** Dr. G. Kronberger **Name** _____ **Aufwand in h** _____
☐ **Gr. 3,** Dr. H. Gruber

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

1. Ossi-Problem (5 Punkte)

Onkel Oskar (alias Ossi) lädt wieder einmal zu einer seiner faden Feten. Anton, Berta, Clemens und Doris können sich nicht einigen, wer von ihnen hin gehen darf/soll/muss. Sie beschließen aber:

1. Mindestens eine/r von ihnen muss hingehen, sonst ist Ossi (wieder einmal) beleidigt.
2. Anton geht auf keinen Fall zusammen mit Doris.
3. Wenn Berta geht, dann geht auch Clemens mit.
4. Wenn Anton und Clemens gehen, dann bleibt Berta auf jeden Fall zuhause.
5. Wenn Anton zuhause bleibt, dann geht entweder Clemens oder es geht Doris.

Gesucht ist ein Pascal-Programm, das alle möglichen Besuchergruppen für Ossis Geburtstagsfeier ausgibt.

Hinweis:

Solche "kombinatorischen Suchprobleme" lassen sich oft lösen, indem man in einem Feld (hier von Werten des Datentyps *BOOLEAN*) alle Möglichkeiten erzeugt und dann überprüft, ob die Bedingungen erfüllt sind.

Versuchen Sie, auf folgender Basis weiterzuarbeiten:

```
     (*… heading comment …*)
PROGRAM Ossi;

  TYPE
    Person   = (anton, berta, clemens, doris);
    Visitors = ARRAY[Person] OF BOOLEAN;

  VAR
    v: Visitors; (*v[p] = TRUE ? person p will attend Ossis party*)
    a, b, c, d: BOOLEAN;

  FUNCTION Valid(v: Visitors): BOOLEAN;
  BEGIN
    (*check all conditions and return TRUE if v holds a valid combination*)
  END; (*Valid*)

BEGIN (*Ossi*)
  FOR a := FALSE TO TRUE DO BEGIN
    v[anton] := a;
    FOR b := FALSE TO TRUE DO BEGIN
      v[berta] := b;
      FOR c := FALSE TO TRUE DO BEGIN
        v[clemens] := c;
        FOR d := FALSE TO TRUE DO BEGIN
          v[doris] := d;
          IF Valid(v) THEN BEGIN
            (*print results*)
          END; (*IF*)
        END; (*FOR*)
      END; (*FOR*)
    END; (*FOR*)
  END; (*FOR*)
END. (*Ossi*)
```

2. Sequentielle Suche und Auswertung boolescher Ausdrücke (1 + 2 Punkte)

Um in einem Feld *a*, das Werte unsortiert enthält, nach einem bestimmten Wert *x* zu suchen, geht man so vor, dass man von vorne beginnend jedes Feldelement mit dem gesuchten Wert vergleicht, bis man den Wert gefunden hat, oder das Feld zu Ende ist. Dieses Suchverfahren heißt *lineare* oder *sequentielle Suche*. Folgende Funktion verwendet dieses Verfahren:

```
FUNCTION IsElement(a: ARRAY OF INTEGER; x: INTEGER): BOOLEAN;
  VAR
    i: INTEGER;
BEGIN
  i := 0;
  WHILE (i <= High(a)) AND (a[i] <> x) DO BEGIN
    i := i + 1;
  END; (*WHILE*)
  IsElement := (i <= High(a));
END; (*IsElement*)
```

Hierbei ist relevant, ob der boolesche Ausdruck zur Steuerung der Schleife vollständig ausgewertet (*complete evaluation*) wird, oder ob die so genannte Kurzschlussauswertung (*short-circuit evaluation*) angewendet wird. Der FreePascal-Compiler erzeugt standardmäßig Code für die Kurzschlussauswertung. Mit der Compilerdirektive (*$B+*), die man am besten gleich zu Beginn des Programms platziert, kann man aber auf vollständige Auswertung umschalten (siehe in *user.pdf*, Appendix F, Seite 189), mit (*$B-*) kann man später wieder auf den Standardmodus zurück gehen. (Compileroptionen, die man beim Aufruf des Compilers, also in der Kommandozeile mitgeben könnte, gibt es für diesen Zweck leider nicht.)

a) Testen Sie obige Funktion mit und ohne Kurzschlussauswertung.

b) Schreiben die Funktion so um, dass sie auch mit vollständiger Auswertung funktioniert.

3. Arithmetik mit Rationalzahlen (4 * 3 + 2 + 2 * 1 Punkte)

Rationalzahlen sind Brüche, bei denen Zähler (*numerator*) und Nenner (*denominator*) ganze Zahlen (G) sind. Jede Rationalzahl r kann dargestellt werden als $r = a / b$ mit $a, b \in G$. Rationalzahlen können in Pascal z.B. mit folgendem Verbund-Datentyp modelliert werden:

```
TYPE
  Rational = RECORD                  (*     numerator     *)
    num, denom: INTEGER;             (*    -----------    *)
  END; (*RECORD*)                    (*    denominator    *)
```

Implementieren Sie vier Pascal-Prozeduren, die Rationalzahlen addieren, subtrahieren, multiplizieren und dividieren können. Jede dieser Prozeduren soll folgende Schnittstelle aufweisen:

```
PROCEDURE …(a, b: Rational; VAR c: Rational);
```

Vergessen Sie dabei nicht, zu kürzen, also Zähler und Nenner durch deren größten gemeinsamen Teiler zu dividieren, damit Überläufe des Datentyps *INTEGER* möglichst vermieden werden. Am besten machen Sie eine eigene Prozedur, die das Kürzen einer Rationalzahl (in Form eines Übergangsparameters) durchführt. Verwenden Sie eine "normalisierte Darstellung", bei der das Vorzeichen einer Rationalzahl im Zähler steht (Nenner immer größer 0) und die ganze Zahlen x in der Form $x / 1$ darstellt.

Außerdem soll für die Eingabe (in Form von Zähler und Nenner) und für die Ausgabe jeweils eine eigene Prozedur zur Verfügung gestellt werden.