

☐ Gr. 1, Dr. D. Auer☐ Gr. 2, Dr. G. Kronberger

Gr. 3, Dr. H. Gruber

Name Andreas Roither Aufwand in h 4 h

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.)

- Testen Sie obige Funktion mit und ohne Kurzschlussauswertung.
- 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
    num, denom: INTEGER;
  END; (*RECORD*)
(*      numerator      *)
(*      -----      *)
(*      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.

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.)

- Testen Sie obige Funktion mit und ohne Kurzschlussauswertung.
- 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
    num, denom: INTEGER;
  END; (*RECORD*)
(*      numerator      *)
(*      -----      *)
(*      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.

Übung 6

Aufgabe 1

Lösungsidee

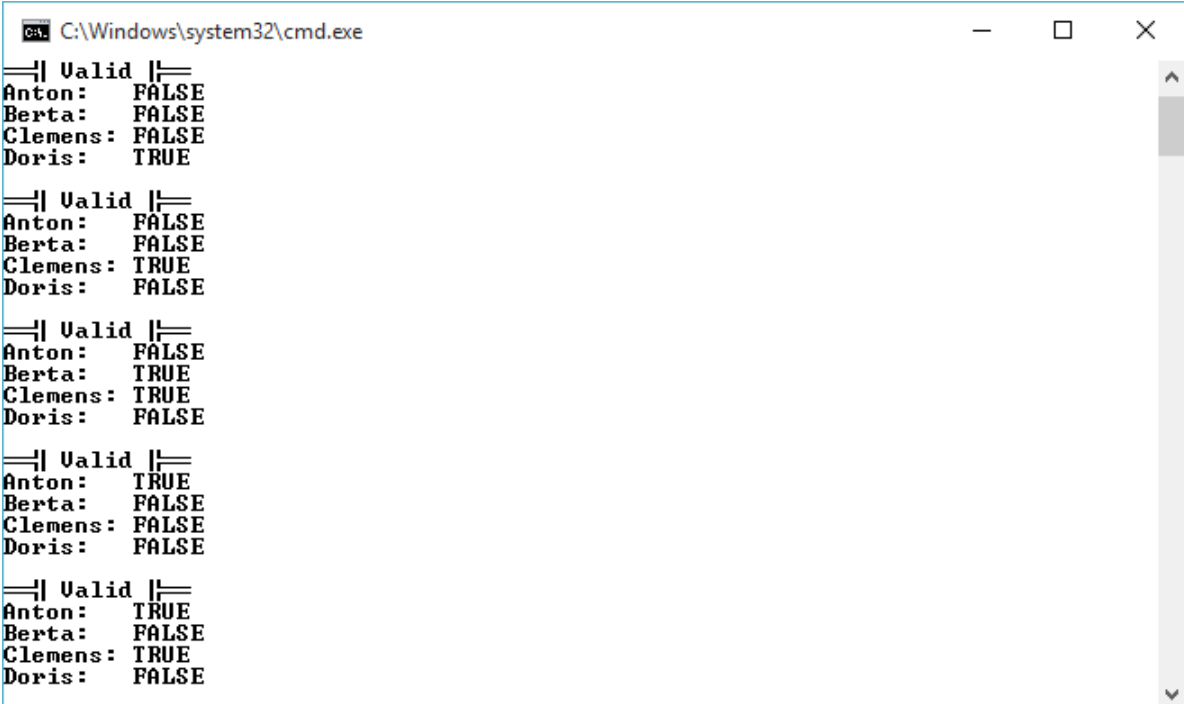
Es werden alle Fälle in denen Valid False zurückgeben muss, beachtet.

```

1 PROGRAM Ossi;
  TYPE
3   Person = (anton, berta, clemens, doris);
   Visitors = ARRAY[Person] OF BOOLEAN;
5
  VAR
7   v: Visitors; (*v[p] = TRUE ? person p will attend Ossis party*)
   a, b, c, d: BOOLEAN;
9  FUNCTION Valid(v: Visitors): BOOLEAN;
  BEGIN
11   (*Alle negativ kriterien, bei denen Valid nicht stimmt*)
   IF NOT v[anton] AND NOT v[clemens] AND NOT v[doris] AND NOT v[doris] THEN
   exit(False);
13   IF v[anton] AND v[doris] THEN exit(False);
   IF v[anton] AND v[clemens] AND v[berta] THEN exit(False);
15   IF v[berta] AND NOT v[clemens] THEN exit(False);
   IF NOT v[anton] AND v[clemens] AND v[doris] THEN exit(False);
17   Valid := True;
  END;
19
  BEGIN
21   FOR a := FALSE TO TRUE DO BEGIN
   v[anton] := a;
23   FOR b := FALSE TO TRUE DO BEGIN
   v[berta] := b;
25   FOR c := FALSE TO TRUE DO BEGIN
   v[clemens] := c;
27   FOR d := FALSE TO TRUE DO BEGIN
   v[doris] := d;
29   IF Valid(v) THEN BEGIN
   WriteLn(chr(205), chr(205), chr(185), ' Valid ', chr(204), chr(205), chr
(205));
31   WriteLn('Anton: ', v[anton]);
   WriteLn('Berta: ', v[berta]);
33   WriteLn('Clemens: ', v[clemens]);
   WriteLn('Doris: ', v[doris], #13#10);
35   END; (*IF*)
   END; (*FOR*)
37   END; (*FOR*)
   END; (*FOR*)
39   END; (*FOR*)
  END. (*Ossi*)

```

Ossi.pas



```
C:\Windows\system32\cmd.exe

==| Valid |==
Anton:  FALSE
Berta:  FALSE
Clemens: FALSE
Doris:  TRUE

==| Valid |==
Anton:  FALSE
Berta:  FALSE
Clemens: TRUE
Doris:  FALSE

==| Valid |==
Anton:  FALSE
Berta:  TRUE
Clemens: TRUE
Doris:  FALSE

==| Valid |==
Anton:  TRUE
Berta:  FALSE
Clemens: FALSE
Doris:  FALSE

==| Valid |==
Anton:  TRUE
Berta:  FALSE
Clemens: TRUE
Doris:  FALSE
```

Abbildung 1: Alle möglichen Kombinationen

Testfall

Zeigt alle möglichen Kombinationen.

Aufgabe 2

Lösungsidee

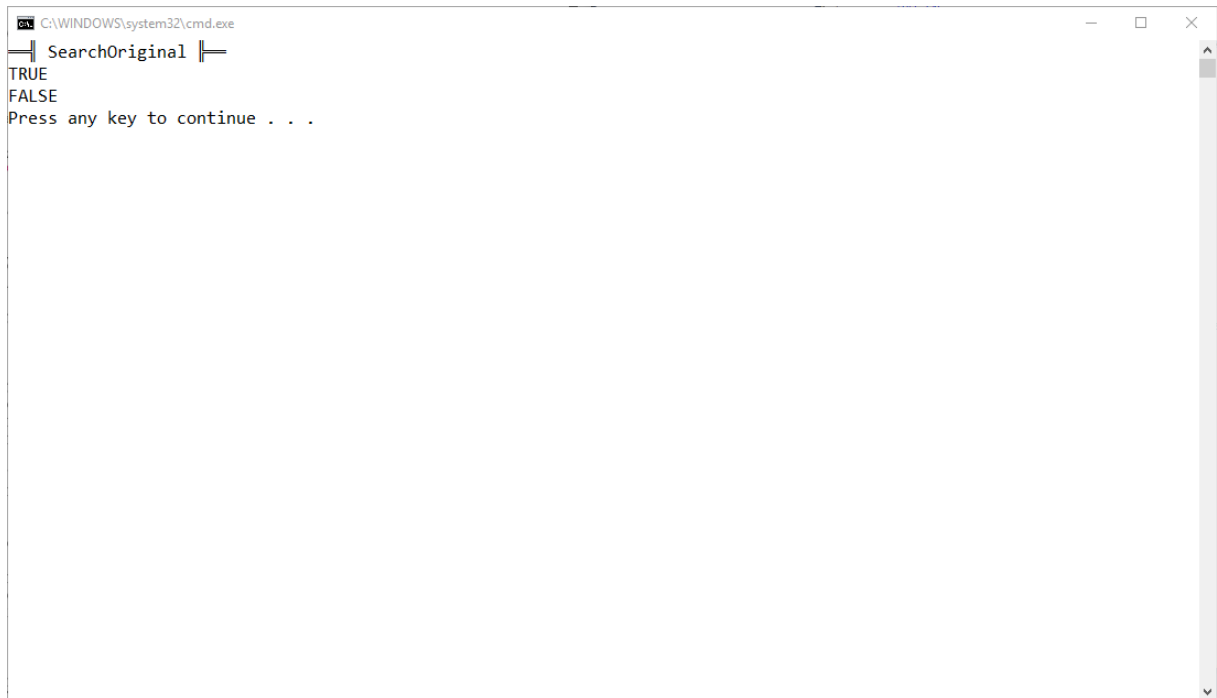
Die Funktion wird so umgeschrieben das kein Boolean als Rückgabewert verwendet wird. Stattdessen wird -1 und 1 verwendet. 1 wird semantisch als True angesehen und -1 wird semantisch als False angesehen.

```
(* $B+*)
2 PROGRAM search;
   FUNCTION IsElement(a: ARRAY OF INTEGER; x: INTEGER): BOOLEAN;
4     VAR
       i: INTEGER;
6     BEGIN
       i := 0;
       WHILE (i <= High(a)) AND (a[i] <> x) DO
8         BEGIN
10          i := i + 1;
12          END;
       IsElement := (i <= High(a));
14     END;

16 VAR arr: ARRAY [1 .. 5] OF INTEGER;
    BEGIN
18     arr[1] := 3;
19     arr[2] := 4;
20     arr[3] := 7;
21     arr[4] := 1;
22     arr[5] := 8;

24     WriteLn(chr(205), chr(205), chr(185), ' SearchOriginal ', chr(204), chr(205), chr
        (205));
25     WriteLn(IsElement(arr, 8));
26     WriteLn(IsElement(arr, 10));
    END.
```

search.pas



```
C:\WINDOWS\system32\cmd.exe
SearchOriginal
TRUE
FALSE
Press any key to continue . . .
```

Abbildung 2: Testfälle Original ohne Directive



```
C:\WINDOWS\system32\cmd.exe
SearchOriginal
Runtime error 201 at $004014C4
$004014C4 ISELEMENT, line 10 of C:/Users/s1610307097/Documents/Pascal/Uebung06/search.pas
$0040150B main, line 25 of C:/Users/s1610307097/Documents/Pascal/Uebung06/search.pas
$00407F11
Press any key to continue . . .
```

Abbildung 3: Testfälle Original mit Directive

```
1  (*$B+*)
PROGRAM search;
3  (*1 und -1 Stellen den Wahrheitswert dar*)
  FUNCTION IsElement(a: ARRAY OF INTEGER; x: INTEGER): Integer;
5    VAR
      i: INTEGER;
7    BEGIN
      FOR i := 0 to High(a) DO
9        IF a[i] = x THEN Exit(1);

11     IsElement := -1;
      END;
13
14  VAR arr: ARRAY [1 .. 5] OF INTEGER;
15  BEGIN
16    arr[1] := 3;
17    arr[2] := 4;
18    arr[3] := 7;
19    arr[4] := 1;
20    arr[5] := 8;
21
22    WriteLn(chr(205), chr(205), chr(185), ' SearchAltered ', chr(204), chr(205), chr
      (205));
23    WriteLn(IsElement(arr, 8));
24    WriteLn(IsElement(arr, 10));
25  END.
```

searchwithoutbool.pas

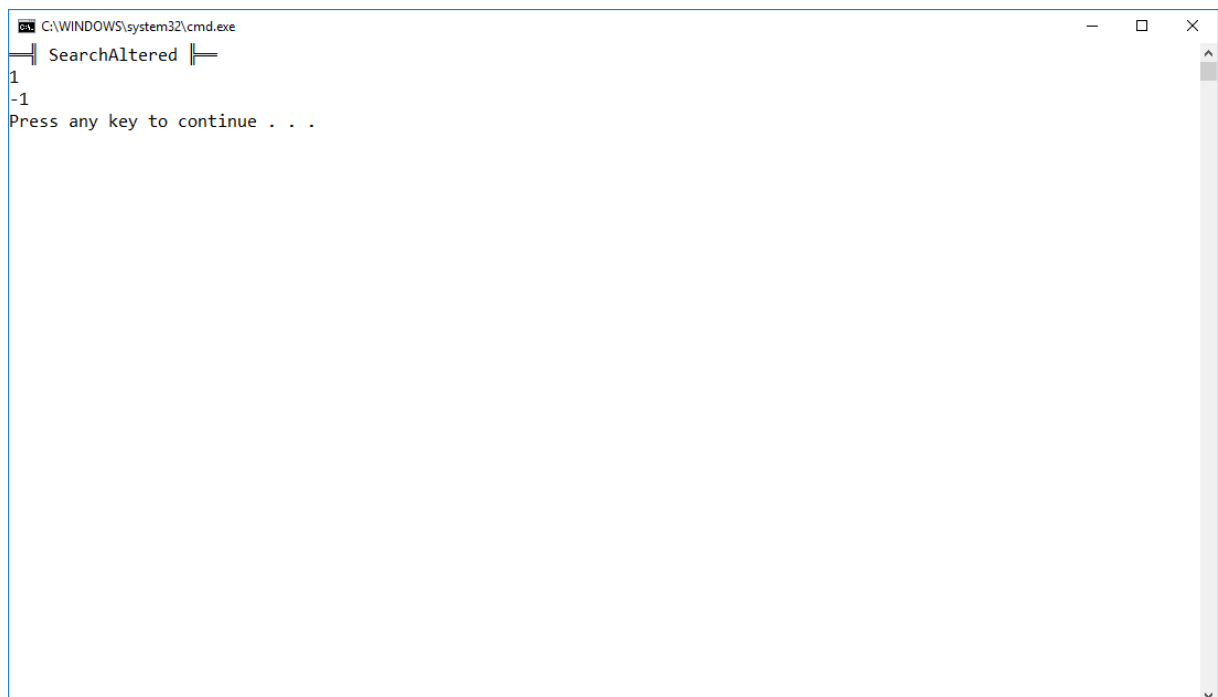


Abbildung 4: Testfälle Suchen mit Directive

Testfälle

Es werden zwei Zahlen gesucht, eine ist vorhanden die andere nicht (1 wenn gefunden, -1 wenn nicht).

Aufgabe 3

Lösungsidee

Es soll mit Rational Zahlen gerechnet werden. Dazu werden 4 verschiedenen Procedures verwendet die Multiplizieren, Dividieren, Addieren und Subtrahieren. Nach jeder Operation wird der ggT (größter gemeinsamer Teiler) gesucht und der Bruch vereinfacht. Zusätzlich wird das Rationale Ergebnis ausgegeben.

```

1 PROGRAM rationalcalc;
3   TYPE
4     Rational = RECORD
5       num, denom: INTEGER;
6     END;
7
8   (* Function to get the greatest common divisor *)
9   FUNCTION ggT (a,b : INTEGER) : INTEGER;
10  VAR remainder: INTEGER;
11  BEGIN
12    WHILE NOT (b=0) DO
13    BEGIN
14      remainder := a MOD b;
15      a := b;
16      b := remainder;
17    END;
18    ggT :=a;
19  END;
20
21  (* Proc to simplify given Rational *)
22  PROCEDURE simplify (Var a : Rational);
23  var divisor : Integer;
24  begin
25    divisor := ggT(a.num, a.denom);
26    a.num := a.num div divisor;
27    a.denom := a.denom div divisor;
28  end;
29
30  PROCEDURE rationalAdd(a, b: Rational; VAR c: Rational);
31  BEGIN
32    IF NOT (a.denom = b.denom) THEN
33    BEGIN
34      c.num := a.num * b.denom + b.num * a.denom;
35      c.denom := a.denom * b.denom ;
36    END
37    ELSE
38    BEGIN
39      c.num := a.num + b.num;
40      c.denom := a.denom;
41    END;
42    simplify(c);
43  END;
44
45  PROCEDURE rationalSub(a, b: Rational; VAR c: Rational);
46  BEGIN
47    IF NOT (a.denom = b.denom) THEN
48    BEGIN

```

```

49      c.num := a.num * b.denom - b.num * a.denom;
      c.denom := a.denom * b.denom ;
51  END
  ELSE BEGIN
53      c.num := a.num - b.num;
      c.denom := a.denom;
55  END;
  simplify(c);
57  END;

59  PROCEDURE rationalMult(a, b: Rational; VAR c: Rational);
  BEGIN
61      c.num := a.num * b.num;
      c.denom := a.denom * b.denom;
63
      simplify(c);
65  END;

67  PROCEDURE rationalDiv(a, b: Rational; VAR c: Rational);
  BEGIN
69      c.num := a.num * b.denom;
      c.denom := a.denom * b.num;
71      simplify(c);
  END;

73
  (*Liest a und b ein*)
75  PROCEDURE input(VAR a, b: Rational);
  BEGIN
77  REPEAT
      Write('a num: ');
79      ReadLn(a.num);
      Write('a denom: ');
81      ReadLn(a.denom);

83      Write('b num: ');
      ReadLn(b.num);
85      Write('b denom: ');
      ReadLn(b.denom);

87
      IF (a.denom = 0) OR (b.denom = 0) THEN
89          WriteLn('a denom und b denom muessen groeser 0 sein')
      UNTIL (a.denom > 0) AND (b.denom > 0)
91  END;

93  (*Gibt eine Rationale Zahl aus*)
  PROCEDURE output(a: Rational);
95  BEGIN
      WriteLn(a.num, '/', a.denom);
97  END;

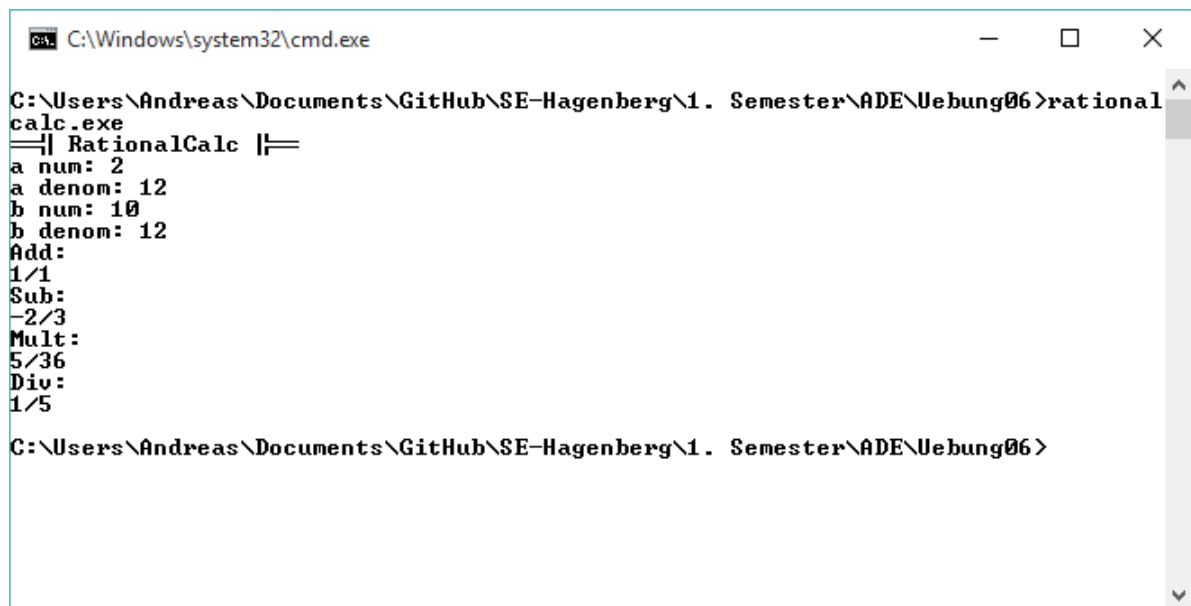
99  VAR a, b, c: Rational;
  BEGIN
101  WriteLn(chr(205), chr(205), chr(185), ' RationalCalc ', chr(204), chr(205), chr
      (205));
      input(a,b);

103
      WriteLn('Add: ');
105  rationalAdd(a,b,c);

```

```
107 output(c);  
WriteLn('Sub: ');  
rationalSub(a,b,c);  
109 output(c);  
WriteLn('Mult: ');  
111 rationalMult(a,b,c);  
output(c);  
113 WriteLn('Div: ');  
rationalDiv(a,b,c);  
115 output(c);  
END.
```

rationalcalc.pas



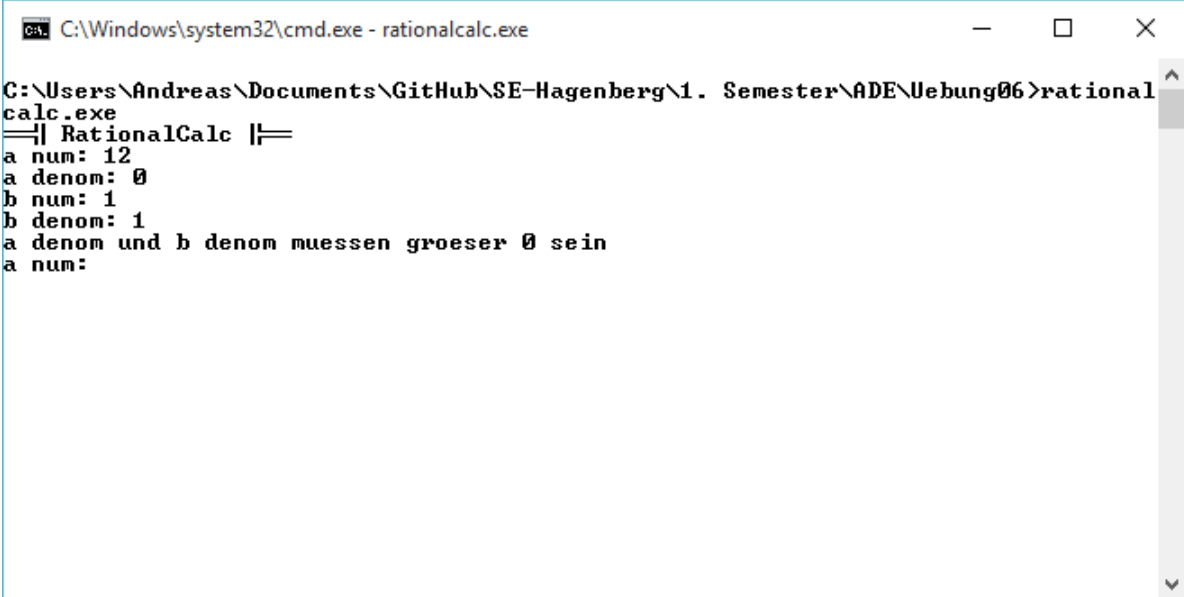
```
C:\Windows\system32\cmd.exe  
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung06>rational  
calc.exe  
==|| RationalCalc ||==  
a num: 2  
a denom: 12  
b num: 10  
b denom: 12  
Add:  
1/1  
Sub:  
-2/3  
Mult:  
5/36  
Div:  
1/5  
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung06>
```

Abbildung 5: Testfall rationalcalc



```
C:\Windows\system32\cmd.exe  
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung06>rational  
calc.exe  
==|| RationalCalc ||==  
a num: -5  
a denom: 2  
b num: -2  
b denom: 4  
Add:  
-3/1  
Sub:  
-2/1  
Mult:  
5/4  
Div:  
5/1  
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung06>
```

Abbildung 6: Testfall rationalcalc



```
C:\Windows\system32\cmd.exe - rationalcalc.exe

C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung06>rational
calc.exe
==| RationalCalc |==
a num: 12
a denom: 0
b num: 1
b denom: 1
a denom und b denom muessen groeser 0 sein
a num:
```

Abbildung 7: Testfall rationalcalc

Testfälle

Die Testfälle zeigen das Rechnen mit zwei verschiedenen Brüchen. Der letzte Testfall zeigt das solange eingelesen wird bis gültige Werte eingegeben wurden.