

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

Gr. 3, Dr. H. Gruber

Name Andreas Roither Aufwand in h 5 h

Punkte \_\_\_\_\_ Kurzzeichen Tutor / Übungsleiter \_\_\_\_\_ / \_\_\_\_\_

**1. Telefonverzeichnis****(16 Punkte)**

Implementieren Sie ein elektronisches Telefonverzeichnis, welches es ermöglicht, Einträge zu speichern, abzurufen und zu löschen. Verwenden Sie dazu die folgenden Deklarationen:

```
CONST
    max = 10;
TYPE
    Entry = RECORD
        firstName: STRING[20];
        lastName:  STRING[30];
        phoneNumber: INTEGER;
    END; (*RECORD*)
    PhoneBook = ARRAY [1 .. max] OF Entry;
```

Das Telefonverzeichnis soll lückenlos gefüllt werden, auf eine Sortierung können Sie verzichten. Beim Löschen eines Eintrags ist die Lücke durch Verschieben der restlichen Einträge zu schließen. Zugriffe auf das Verzeichnis dürfen nur über die Prozeduren *AddEntry*, *DeleteEntry*, *SearchName*, *SearchNumber* und die Funktion *NrOfEntries* erfolgen (wählen Sie die Parameter passend):

- PROC. **AddEntry**(...);  
Erweitert das Verzeichnis um einen Eintrag. Bei einem Überlauf muss eine Fehlermeldung ausgegeben werden; das Verzeichnis darf in diesem Fall nicht verändert werden.
- PROC. **DeleteEntry**(...);  
Versucht, einen durch Vor- und Nachnamen gegebenen Eintrag zu entfernen. Ist dieser nicht vorhanden, so ist eine Fehlermeldung auszugeben.
- PROC. **SearchNumber**(...);  
Sucht einen Eintrag mit Vor- und Nachnamen. Ist kein solcher vorhanden, so ist eine Fehlermeldung auszugeben. Bei mehrfachen Einträgen sind alle auszugeben.
- PROC. **SearchName**(...);  
Sucht einen Eintrag nach der Telefonnummer. Ist kein Eintrag mit dieser Nummer vorhanden, so ist eine Fehlermeldung auszugeben.
- FUNC. **NrOfEntries**(...): INTEGER;  
Liefert die aktuelle Anzahl der Einträge.

**2. Feldverarbeitung mit offenen Feldparametern****(8 Punkte)**

Gegeben sind zwei beliebig große Felder *a1* und *a2*, die positive ganze Zahlen in aufsteigend sortierter Reihenfolge enthalten. Gesucht ist eine Pascal-Prozedur

```
PROC. Merge(a1, a2: ARRAY OF INT.; VAR a3: ARRAY OF INT.; VAR n3: INT.);
```

die ein aufsteigend sortiertes Feld *a3* liefert, das alle *n3* Zahlen enthält, die in *a1* oder in *a2* aber nicht in *a1* und *a2* vorkommen (vgl. *XOR*). Beachten Sie, dass in jedem Feld (also auch im Ergebnisfeld *a3*) Werte mehrfach vorkommen können. Im Fehlerfall (Feld *a3* würde überlaufen) soll *n3* auf -1 gestellt werden. *Beispiel*:

<i>a1</i> =	2	4	4	10	15	15
<i>a2</i> =	3	4	5	10		
<i>a3</i> =	2	3	5	15	15	

und *n3* = 5

# Übung 5

## Aufgabe 1

### Lösungsidee

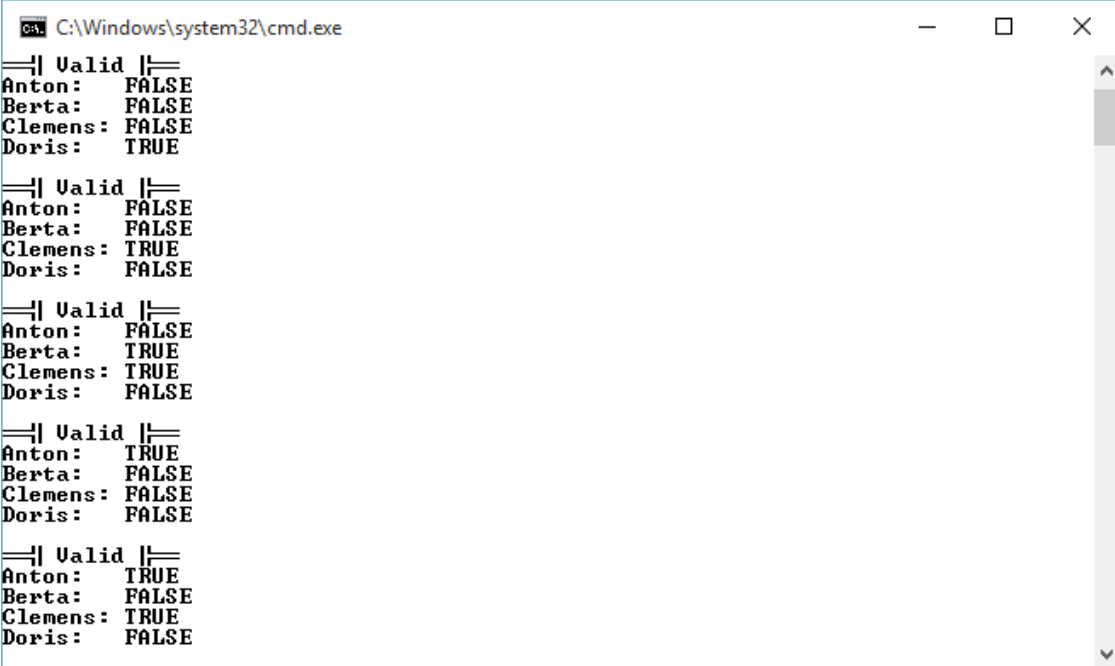
Es werden alle Fälle in denen Valid False zurückgeben muss, beachtet.

```

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

## Kombinationen

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

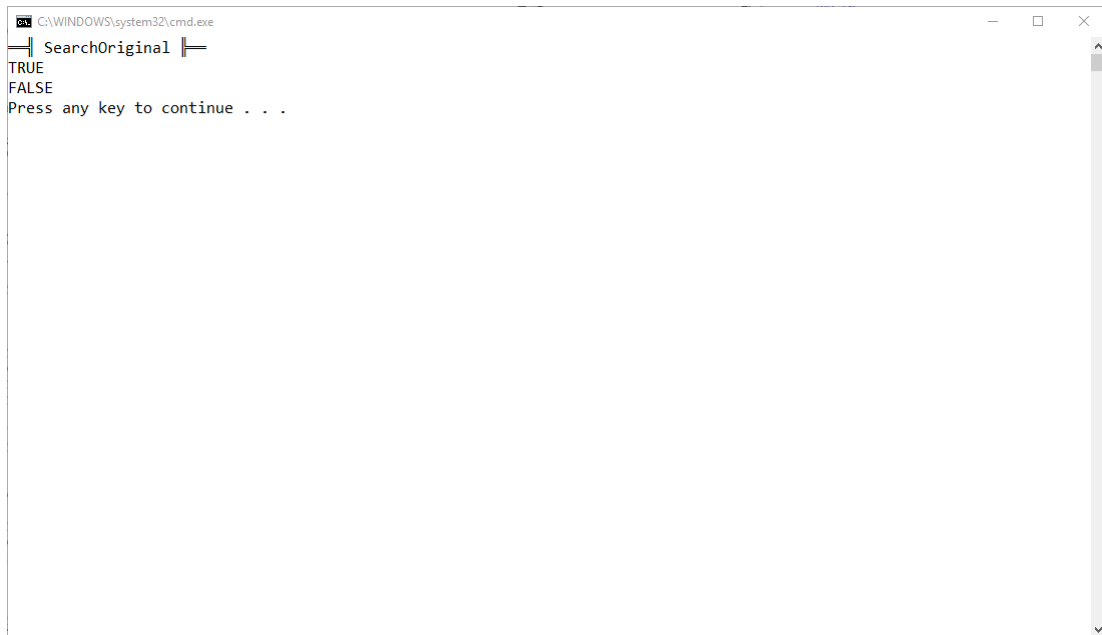


Abbildung 2: Testfälle Original ohne Directive

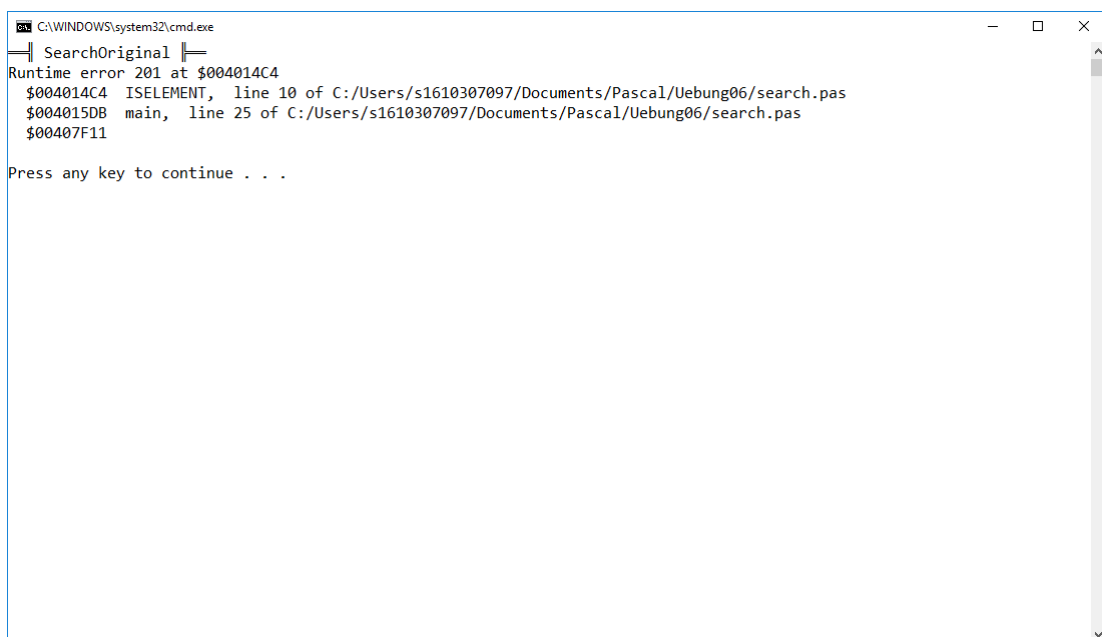


Abbildung 3: Testfälle Original mit Directive

```

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;
12      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
23          (205));
24      WriteLn(IsElement(arr, 8));
25      WriteLn(IsElement(arr, 10));
26  END.
```

searchwithoutbool.pas

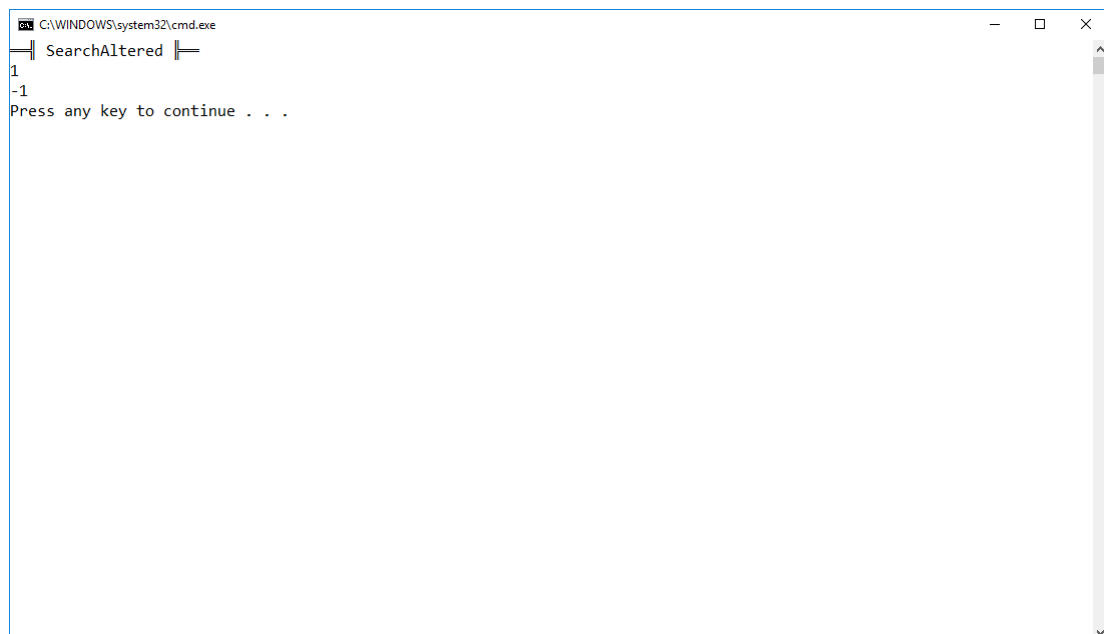


Abbildung 4: Testfälle Suchen mit Directive

## Testfälle

## 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



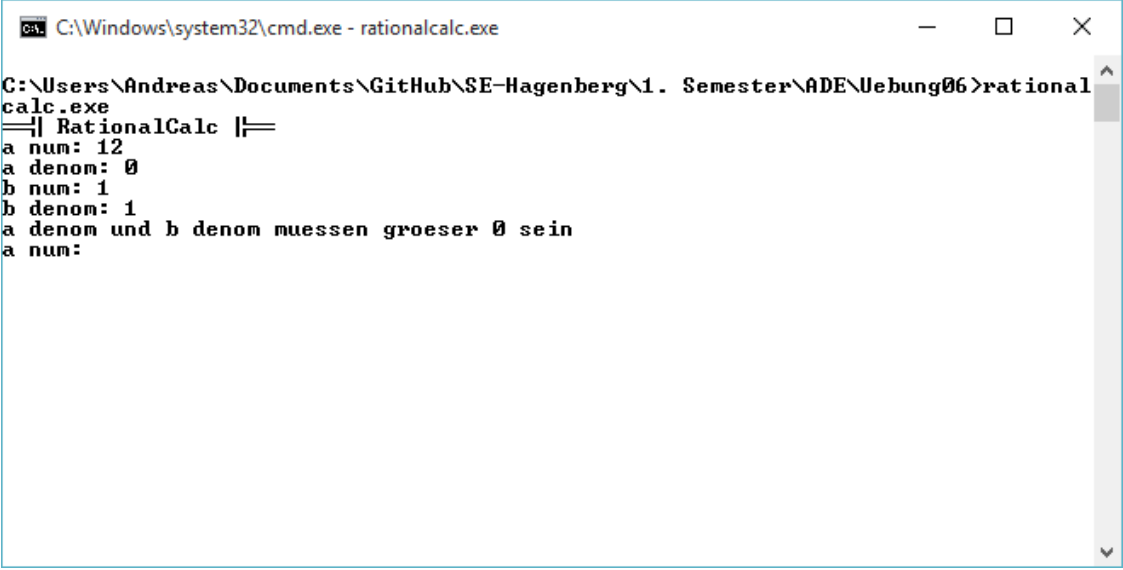
```
C:\Windows\system32\cmd.exe  
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1. Semester\ADE\Uebung06>rational  
calc.exe  
==| RationalCalc |==  
a num: 2  
a denom: 12  
b num: 10  
b denom: 12  
Add:  
1/1  
Sub:  
-2/3  
Mult:  
5/36  
Div:  
1/5  
C:\Users\Andreas\Documents\GitHub\SE-Hagenberg\1.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