PRG1x & ADE1x	Üb. zu Einf. in die Programmierung und Element. Alg. u. Datenstrukt.	WS 20/21, Übung 6

Abgabetermin: Mi, 18. 11. 2020

☐ **Gr. 1,** Dr. D. Auer **Name** Angelos Angelis **Aufwand in h** 6

☒ **Gr. 2,** Dr. G. Kronberger

☐ **Gr. 3,** Dr. S. Wagner **Punkte** _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

1. 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 := Low(a);
  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 (*$B+*), die man am besten gleich zu Beginn des Programms platziert, kann man aber auf vollständige Auswertung umschalten (siehe in *doc/user.pdf*, Appendix F, Seite 213), mit (*$B-*) kann man später wieder auf den Standardmodus zurückgehen. (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.

2. Namenskonvertierung (5 Punkte)

Wenn Bezeichner für Variablen, Funktionen etc. aus mehreren Wörtern bestehen, werden die Wortgrenzen meist durch eine spezielle Schreibweise sichtbar gemacht. In Pascal-Programmen schreibt man z.B. den Anfangsbuchstaben ab dem zweiten Wort groß (z.B. `maxStringLength` und `MoveToFront`), und in C-Programmen verwendet man häufig den Unterstrich als Trennzeichen (z.B. `max_string_length` und `Move_to_front`).

Implementieren Sie eine Pascal-Prozedur, die einen gegebenen Bezeichner (vom Typ STRING) in C-Schreibweise in die Pascal-Schreibweise konvertiert. Berücksichtigen Sie, dass auch mehrere Unterstriche zwischen den Wörtern stehen können.

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

Rationale Zahlen sind Brüche, bei denen Zähler (*numerator*) und Nenner (*denominator*) ganze Zahlen (G) sind. Jede rationale Zahl r kann dargestellt werden als $r = a / b$ mit $a, b \in G$. Rationale Zahlen 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 rationale Zahlen **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 rationalen Zahl (in Form eines Übergangsparameters) durchführt. Verwenden Sie eine "normalisierte Darstellung", bei der das Vorzeichen einer rationalen Zahl im Zähler steht (Nenner immer größer 0) und die eine ganze Zahl 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.

Hinweise:

1. Geben Sie für alle Ihre Lösungen immer eine „Lösungsidee“ an.
2. Dokumentieren und kommentieren Sie Ihre Algorithmen.
3. Bei Programmen: Geben Sie immer auch Testfälle ab, an denen man erkennen kann, dass Ihr Programm funktioniert, und dass es auch in Fehlersituation entsprechend reagiert.

Lösungsidee:

Da bei der complete evaluation die ganze Zeile und damit auch das „`(a[i] <> x)`" überprüft wird, obwohl „`(i <= High(a))`" nicht wahr ist, muss man darauf achten dass nicht über das array „a" hinaus überprüft wird. Das habe ich mit einer FOR Schleife gemacht.

Quellcode:

```
(*$B+*)
PROGRAM SeqSearch;

TYPE IntArray = ARRAY[1..3] OF INTEGER;

FUNCTION IsElement(a: ARRAY OF INTEGER; x: INTEGER): BOOLEAN;
VAR
  i: INTEGER;
BEGIN
  i := Low(a);
  FOR i := LOW(a) TO HIGH(a) DO BEGIN
    IF (a[i] <> x) THEN BEGIN
     i := i + 1;
    END;
  END; (* WHILE *)
  IsElement := (i <= High(a));
END; (* IsElement *)

VAR
arr : IntArray;
BEGIN
  arr[1] := 1;
  arr[2] := 2;
  arr[3] := 3;
  WriteLn(IsElement(arr, 3));
  WriteLn(IsElement(arr, 4));
END
```

Testfall:

```
TRUE
FALSE
```

Lösungsidee:

Mit einer FOR Schleife prüft man jeden CHAR des Strings ob der CHAR ein Unterstrich ist und den Löschen den Nächsten Char mit den entsprechenden Großbuchstaben ersetzen. Man sollte ebenfalls am Anfang der Schleife überprüfen, ob nochmal ein Unterstrich kommt und löschen.

Quelltext:

```
PROGRAM Namenskonvertierung;

CONST
CapitalToLower = 32;

PROCEDURE CToPascal(VAR s :String);
VAR
i,LowerChar :INTEGER;
BEGIN
  FOR i := 1 TO Length(s) DO BEGIN
    IF s[i] = '_' THEN BEGIN
      IF s[i+1] = '_' THEN BEGIN
       Delete(s,i,1);
       Dec(i)
      END
      Else BEGIN
      LowerChar := Ord(s[i+1])-CapitalToLower;
      Delete(s,i,2);
      Insert(CHAR(Ord(LowerChar)),s,i)
      END;
    END;
  END;
END;

VAR
s : STRING;
BEGIN
WriteLn('Bitte C Bezeichner eingeben um ihn auf die Pascal-
Schreibweise zu konvertieren');
Read(s);
CToPascal(s);
WriteLn('Pascal Konvertierung: ',s);
END.
```

Testfall1)

```
Bitte C Bezeichner eingeben um ihn auf die Pascal-Schreibweise zu konvertieren
max_string_length
Pascal Konvertierung: maxStringLength
```

Testfall2)

```
Bitte C Bezeichner eingeben um ihn auf die Pascal-Schreibweise zu konvertieren
Move_to_front
Pascal Konvertierung: MoveToFront
```

Testfall3)

```
Bitte C Bezeichner eingeben um ihn auf die Pascal-Schreibweise zu konvertieren
test_______tomany_____underscores
Pascal Konvertierung: testTomanyUnderscores
```

Lösungsidee:

Eingabe:

Wird erstmal der Zähler dann der Nenner abgefragt und gelesen und dann wird überprüft ob der Nenner kleiner 0 ist. Falls ja wird ein error ausgegeben.

Das kürzen wird genau so programmiert wie es in der Aufgabe beschrieben wird.

Addieren: Der Ergebnisszähler wird errechnet in dem man das Produkt von bruch1zähler und bruch2nenner, mit dem Produkt von bruch1nenner und bruch2zähler addiert. Für den Ergebnisnenner muss man einfach beide Nenner multiplizieren

Subtrahieren: Genau wie addieren nur beim Zähler muss man subtrahieren anstatt zu addieren

Multiplikation: Zähler mit Zähler und Nenner mit Nenner multiplizieren

Division: Bruch1 mit dem Kehrwert von Bruch2 multiplizieren

Quelltext:

```
PROGRAM ratZahlen;

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

PROCEDURE ReadInput(VAR RatZahl : Rational);
BEGIN
  WriteLn('Bitte Zähler eingeben');
  ReadLn(RatZahl.num);
```

```
  WriteLn('Bitte Nenner eingeben');
  ReadLn(RatZahl.denom);
  IF RatZahl.denom = 0 THEN BEGIN
    WriteLn('error');
    Halt;
  END ELSE IF RatZahl.denom < 0 THEN BEGIN
    RatZahl.denom := Abs(RatZahl.denom);
    RatZahl.num := - RatZahl.num;
  END;
END;

PROCEDURE Kuerzen(VAR c : RATIONAL);
VAR
i,Highest,Smallest : INTEGER;
BEGIN
IF c.num >= c.denom THEN BEGIN
  Highest := c.num;
  Smallest := c.denom;
END
ELSE
begin
  Highest := c.denom;
  Smallest := c.num;
end;
  FOR i := Highest DOWNTO 0 DO BEGIN
    IF (Highest MOD i = 0) AND (Smallest MOD i = 0) THEN BEGIN
      c.num := c.num DIV i;
      c.denom := c.denom DIV i;
      EXIT;
    END;
  END;
END;

PROCEDURE RatSum(a,b : Rational; VAR c : RATIONAL);
BEGIN
  c.num := (b.denom * a.num) + (a.denom * b.num);
  c.denom := a.denom * b.denom;
  Kuerzen(c);
END;

PROCEDURE RatDif(a,b : Rational; VAR c : RATIONAL);
BEGIN
  c.num := (b.denom * a.num) - (a.denom * b.num);
  c.denom := a.denom * b.denom;
  Kuerzen(c);
```

```
END;

PROCEDURE RatDiv(a,b : Rational; VAR c : RATIONAL);
BEGIN
 c.num := a.num * b.denom;
 c.denom := a.denom * b.num;
 Kuerzen(c);
END;

PROCEDURE RatPro(a,b : Rational; VAR c : RATIONAL);
BEGIN
 c.num := a.num * b.num;
 c.denom := a.denom * b.denom;
 Kuerzen(c);
END;

PROCEDURE WriteRecord(a:Rational);
BEGIN
  Write(a.num,'/',a.denom,' ');
END;

VAR
RatZahl1 : Rational;
RatZahl2 : Rational;
RatErg: Rational;
BEGIN
ReadInput(RatZahl1);
ReadInput(RatZahl2);

WriteLn('Summieren von: ');
WriteRecord(RatZahl1);
WriteRecord(RatZahl2);
RatSum(RatZahl1,RatZahl2,RatErg);
Write('Ergebniss: ');
WriteRecord(RatErg);
WriteLn;
WriteLn;

WriteLn('Subtrahieren von: ');
WriteRecord(RatZahl1);
WriteRecord(RatZahl2);
RatDif(RatZahl1,RatZahl2,RatErg);
Write('Ergebniss: ');
WriteRecord(RatErg);
WriteLn;
```

```
WriteLn;

WriteLn('Multiplizieren von: ');
WriteRecord(RatZahl1);
WriteRecord(RatZahl2);
RatPro(RatZahl1,RatZahl2,RatErg);
Write('Ergebniss: ');
WriteRecord(RatErg);
WriteLn;
WriteLn;

WriteLn('Dividieren von: ');
WriteRecord(RatZahl1);
WriteRecord(RatZahl2);
RatDiv(RatZahl1,RatZahl2,RatErg);
Write('Ergebniss: ');
WriteRecord(RatErg);
END.
```

Testfall1)

```
Bitte Zaehler eingeben
4
Bitte Nenner eingeben
8
Bitte Zaehler eingeben
7
Bitte Nenner eingeben
8
Summieren von:
4/8 7/8 Ergebniss: 11/8

Subtrahieren von:
4/8 7/8 Ergebniss: -3/8

Multiplizieren von:
4/8 7/8 Ergebniss: 7/16

Dividieren von:
4/8 7/8 Ergebniss: 4/7
```

Testfall2)

```
Bitte Zaehler eingeben
-2
Bitte Nenner eingeben
5
Bitte Zaehler eingeben
4
Bitte Nenner eingeben
8
Summieren von:
-2/5 4/8 Ergebniss: 1/10

Subtrahieren von:
-2/5 4/8 Ergebniss: -9/10

Multiplizieren von:
-2/5 4/8 Ergebniss: -1/5

Dividieren von:
-2/5 4/8 Ergebniss: -4/5
```

Testfall3)

```
Bitte Zaehler eingeben
0
Bitte Nenner eingeben
0
error
```

Testfall4)

```
Bitte Zaehler eingeben
0
Bitte Nenner eingeben
1
Summieren von:
0/1 0/1 Ergebniss: 0/1

Subtrahieren von:
0/1 0/1 Ergebniss: 0/1

Multiplizieren von:
0/1 0/1 Ergebniss: 0/1

Dividieren von:
0/1 0/1 error
```