Aribas

Kommandozeilen – Befehle:

Wichtig: Alle Befehle (Zuweisungen, Berechnungen) müssen mit `.` abgeschlossen werden.

`:=`	Zuweisung
`a**b`	Potenzieren
`a mod b`	Modulo-Rechnung
`a div b`	Ganzzahlige Division ohne Rest
`random(n)`	Zufallszahl zwischen `0` (inklusive) und `n` (exklusive)
`load("bsp")`	Laden der Funktionen aus der Datei bsp.ari
_	Zugriff auf das Ergebnis des letzten Befehls
__	Zugriff auf das Ergebnis des vorletzten Befehls
___	Zugriff auf das Ergebnis des vorvorletzten Befehls
Ctrl-F1	letzer Befehl erscheint in der Kommandozeile
Ctrl-F2	vorletzer Befehl erscheint in der Kommandozeile
Ctrl-F3	drittletzer Befehl erscheint in der Kommandozeile

Programmieren neuer Funktionen:

Aufbau:

```
function(<Eingabeparameter>):<Rückgabetyp>
var
      <Variablendeklaration>
begin
      <Anweisungen>
end;
```

`<Eingabeparameter>` und `<Variablendeklaration>`:

`<Variablenname>:<Typ>;`
mehrere Variablen gleichen Typs werden durch `,` getrennt;
die Angaben zu verschiedenen Typen werden durch `;` getrennt

`<Typ>` und `<Rückgabetyp>`:

```
integer
real
```

`<Anweisungen>`:

Anweisungen werden durch `;` getrennt;
alle Variablen müssen deklariert werden;

Fallunterscheidung:

```
if <Bedingung> then
      <Anweisungen>
else
      <Anweisungen>
end;
```

(der `else`-Zweig kann entfallen; es muss immer ein `end;` stehen)

Schleifen:

```
for <Variable> := <Start> to <Ende> do
      <Anweisungen>
end;

while <Bedingung> do
      <Anweisungen>
end;
```

Vergleiche: gleich: `=`, ungleich: `<>` oder `/=`

Ausgaben z.B.: `writeln("Wert der Variablen a: ",a)`

Kommentare werden in `(*` … `*)` eingeschlossen;
durch `#` wird der Rest der Zeile zum Kommentar

Aribas-Beispielprogramm:

```
(* Funktion zur Berechnung von n! *)
function fak(n: integer):integer
var
      k,erg: integer;
begin
      erg := 1;
      for k:=1 to n do
            erg := erg*k;
      end;
      return erg;
end;

(* Funktion zur Berechnung von exp(x) *)
function expo(x: real):real
var
      erg, summand, posx, posxn: real;
      n : integer;
begin
      # bei x<0: berechne exp(-x)
      #        und nimm anschliessend den Kehrwert
      if x<0 then
            posx := -x; # posx = positives x = |x|
      else
            posx := x;
      end;

      # Berechne exp(x) durch die Potenzreihenentwicklung:
      # exp(x) = 1 + x + x^2/fak(2) + x^3/fak(3) + ...

      # Initialisierungen
      erg := 1.0;
      n := 1;
      posxn := posx;    # posxn enthaelt |x|**n

      # Berechnung des Summanden zu n=1
      summand := posxn;

      while (summand > 0.00001) do
            writeln("Summand: ",summand);
            erg := erg+summand;
            n := n+1;
            posxn := posxn*posx;
            summand := posxn / fak(n);
      end;

      # bei x<0 muss der Kehrwert genommen werden
      if x<0 then
            erg := 1/erg;
      end;

      return erg;
end;
```