Maple-Test zur LVA 'Computermathematik', SS 2010
16. April 2010 ** GRUPPE WEISS **

L Ö S U N G E N

*** AUFGABE 1 (3 Punkte):**

(a) Erklären Sie jeden einzelnen Befehl.
(b) Welchen Typ hat das Objekt X ?
(c) Geben Sie an, wie die Ausgabe aussieht.

*** Lösung:**

```
> n := 6;
```

$$n := 6$$

```
> r := 1..n;  # der 'range 1 bis n'
  whattype(r);
```

$$r := 1 .. 6$$

$$..$$

```
> X := {seq(i mod n/2,i=r)}; # Menge
```

$$X := \{0, 1, 2\}$$

```
> add(x,x in X); # Summe der Elemente
```

$$3$$

*** AUFGABE 2 (3 Punkte):**

Finden Sie alle Syntaxfehler in der Prozedur:

```
> p := proc(x;n)
  local i,y;
  y := 0;
  do i from 1 to n
     if is(even,y) then
        y =: y+x
     else
        y =: x-y
     end then
  end do;
  return y
  end proc;
Error, `;` unexpected
```

*** Lösung:**

6 Syntaxfehler: Korrekte Version:

```
> p := proc(x,n) # <--
  local i,y;
  y := 0;
  for i from 1 to n do  # <--
     if is(y,even) then # <--
        y := y+x # <--
     else
        y := x-y # <--
     end if # <--
  end do;
  return y
  end proc:
```

*** AUFGABE 3 (4 Punkte):**

Schreiben Sie eine Prozedur cum(L,type), die zu einer gegebenen Liste L die (gleich lange) Liste K der kumulierten Summen bzw. Produkte zurückliefert, dh. K[i] soll gleich sein zu L[1]+...+L[i] (falls type=`S`) bzw. L[1]*...*L[i] (falls type=`P`).

Vermeiden Sie unnötige Redundanz bei der Berechnung.

*** Lösung: (Code der Prozedur cum):**

```
> cum := proc(L,type)
  local cumvar,i,K;
  K := [seq(0,i=1..nops(L))];
  if type=`S` then
    cumvar := 0;
    for i from 1 to nops(L) do
       cumvar := cumvar+L[i];
       K[i] := cumvar
    end do
  elif type=`P` then
    cumvar := 1;
    for i from 1 to nops(L) do
       if L[i]=0 then # ab hier alles 0
          return K
       end if;
       cumvar := cumvar*L[i];
       K[i] := cumvar
    end do
  else
    return NULL
  end if;
  return K
  end proc:

> cum([1,2,3],`S`);
                                [1, 3, 6]
> cum([2,3,4,0,7],`P`);
                             [2, 6, 24, 0, 0]
> cum([],`P`);
                                  [ ]
> cum([1,2],`?`);
```

*** AUFGABE 4 (4 Punkte):**

Schreiben Sie eine Prozedur compare(L1::list,L2::list), die zu zwei gegebenen Listen L1, L2 folgendes zurückgibt:

-- L1, L2 haben verschiedene Länge: Abbruch mit Fehlermeldung

-- L1, L2 haben gleiche Länge: L (als Liste)
wobei L[i] := L1[i] falls L1[i]=L2[i],
L[i] := FAIL sonst

*** Lösung (Code der Prozedur compare):**

```
> compare := proc(L1,L2::list)
  local i, L:=[];
  if nops(L1)<>nops(L2) then
     error "L1,L2 nicht gleich lang!"
  else
     for i from 1 to nops(L1) do
        L:=[op(L),`if`(L1[i]=L2[i],L1[i],FAIL)]
     end do;
     return L
  end if
  end proc:

> compare([1,2,3,4],[1,3,2,4]);
                              [1, FAIL, FAIL, 4]
> compare([1,2,3,4],[]);
Error, (in compare) L1,L2 nicht gleich lang!
```

*** AUFGABE 5 (4 Punkte):**

Schreiben Sie eine Prozedur HesseMatrix(f::procedure,xi,eta),
die zu einer gegebenen reellwertigen Funktion f(x,y)
die Hesse-Matrix der 2.partiellen Ableitungen von f, d.h.

(f_xx f_xy)
(f_yx f_yy)

an der Stelle (x,y)=(xi, eta) als Matrix (2x2) zurückliefert.

*** Lösung (Code der Prozedur HesseMatrix):**

(Geben Sie auch ein Beispiel dafür, wie die Prozedur aufzurufen ist.)

```
> HesseMatrix := proc(f::procedure,xi,eta)
  local H,i,j,x,y;
  H := Matrix([[diff(f(x,y),x,x),diff(f(x,y),x,y)],
               [diff(f(x,y),y,x),diff(f(x,y),y,y)]]);
  for i from 1 to 2 do
  for j from 1 to 2 do
    H[i,j] := subs(x=xi,y=eta,H[i,j])
  end do
  end do;
  # oder statt der Doppelschleife:
  # eleganter so (subtilere Syntax!)
  H := map2(subs,{x=xi,y=eta},H);
  return H;
  end proc:

> HesseMatrix((x,y)->(x+2*y)^3,x,y);
```

$$\begin{bmatrix} 6x+12y & 12x+24y \\ 12x+24y & 24x+48y \end{bmatrix}$$

```
> HesseMatrix((x,y)->(x+2*y)^3,1,2);
```

$$\begin{bmatrix} 30 & 60 \\ 60 & 120 \end{bmatrix}$$

*** AUFGABE 6 (5 Punkte):**

Schreiben Sie eine Prozedur fpiter(f::procedure,a,b,n)
die für eine gegebene Funktion f(x) eine Fixpunkt-Iteration ausführt:
Gesucht ist ein `Fixpunkt' von f, d,h. ein x in [a,b] mit der Eigenschaft x=f(x).
Die Iterationsvorschrift lautet: x_0 := (a+b)/2, und

x_i := f(x_(i-1)), i=1,...,n

Zurückzugeben sind alle Iterierten x_0,...,x_n in einem Array X.
Exakte Arithmetik (keine Rundung mit evalf o.ä.).

! Sollte man jedoch während der Iteration ein x_i finden mit x_i=f(x_i), so wird die Iteration abgebrochen, und es wird anstelle des Arrays X der betreffende Wert x_i zurückgegeben. Falls x_i<a oder x_i>b eintritt, wird FAIL zurückgegeben.

*** Lösung (Code der Prozedur fpiter):**

```
> fpiter := proc(f::procedure,a,b,n)
  local fx,i, X:=Array(0..n);
  X[0] := (a+b)/2;  fx := f(X[0]);
  if fx=X[0] then return X[0] end if;
  for i from 1 to n do
     if (fx<a or fx>b) then return FAIL end if;
     X[i] := fx;  fx :=f(X[i]);
     if fx=X[i] then return X[i] end if;
  end do;
  return X
  end proc:

> fpiter(x->1/2-x,0,1/2,3);
```

$$\frac{1}{4}$$

```
> X:=evalf(fpiter(x->1/2-x^2,0,1,5));
```

$$X := \text{Array}(0 .. 5, \{(0) = 0.5000000000, (1) = 0.2500000000, (2) = 0.4375000000, (3) = 0.3085937500, (4) = 0.4047698975, (5) = 0.3361613301\}, \textit{datatype} = \textit{anything}, \textit{storage} = \textit{rectangular}, \textit{order} = \textit{Fortran\_order})$$

```
> evalf(solve(x=1/2-x^2,x))[1];
```

$$0.3660254038$$

```
> fpiter(x->3+x,0,1,2);
```

$$\textit{FAIL}$$

*** AUFGABE 7 (4 Punkte):**

Beschreiben Sie möglichst präzise die wichtigsten Datenstrukturen in Maple (mindestens drei), samt wesentlichen Eigenschaften und Art der Verwendung.

*** Lösung:**

... z.B. set, list, Array

*** AUFGABE 8 (3 Punkte):**

Was tut die folgende Prozedur M ?
-- Erklären Sie genau, was M bei korrektem Aufruf zurückliefert.
-- Was bedeutet "gefunden." bzw. "nicht gefunden"?
-- Um welche Aussage der Analysis geht es hier?

```
> M := proc(f::procedure,a,b)
  local m,x;
  m := int(f(x),x=a..b)/(b-a);
  x := evalf(solve(f(x)=m));
  if evalb(x=NULL) then
    print ("nicht gefunden");
    return FAIL;
  else
    print("gefunden");
    return x;
  end if;
  end proc:
```

*** Lösung:**

1. Mittelwertsatz der Integralrechnung

```
> M(sin,0,1);
```

"gefunden"

0.4776547629

```
> f := x->piecewise(x<0,-1,2); # unstetig
```

$f := x \rightarrow \text{piecewise}(x < 0, -1, 2)$

```
> M(f,-1,2);
```

"nicht gefunden"

FAIL