Maple-Test zur LVA 'Computermathematik'

**** MIT LÖSUNGEN (Codes, Erläuterungen) ****

** GRUPPE WEISS **

7. Mai 2009

NAME, Vorname: .....

Matr.Nr.: .....

- **8 Aufgaben**
- **30 Punkte + ggf. 4 Bonuspunkte 'extra' (Aufgabe 8)**
- **Die Verwendung schriftlicher Unterlagen ist nicht gestattet.**
- **Unten zu formulierende Prozeduren:**
  generell ohne spezielle Parameterüberprüfung.
- **Bitte verwenden Sie jeweils nur das eine Blatt für ihre**
  Lösung. Keine Extra-Blätter beilegen.

==========

===== 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.

```
> n:=5;
```

$$n := 5$$

```
> X:=[seq(i,i=1..n)];
```

$$X := [1, 2, 3, 4, 5]$$

```
> for i from 1 to nops(X) do
      X[i]:=X[nops(X)-i+1]
  end do;
```

$$X_1 := 5$$
$$X_2 := 4$$
$$X_3 := 3$$
$$X_4 := 4$$
$$X_5 := 5$$

Lösung (Hinweise):

-- seq(...) konstruiert expression sequence, [...] ist die daraus entstehende Liste

-- nops (...) liefert Anzahl der Elemente der Liste X

-- sinmple Schleife

===== AUFGABE 2 (3 Punkte):

Was passiert bei nachfolgendem Aufruf p(k)?

```
> p:=proc(n)
  local i,x;
  x:=0;
  for i from 1 to n do
    x:=x+i
  end do
  end proc;
```

$$p := \mathbf{proc}(n)\ \mathbf{local}\ i, x;\ x := 0;\ \mathbf{for}\ i\ \mathbf{to}\ n\ \mathbf{do}\ x := x + i\ \mathbf{end\ do\ end\ proc}$$

```
> k:='k';
```

$$k := k$$

```
> p(k);
Error, (in p) final value in for loop must be numeric or character
```

Lösung:

-- Schleife nicht ausführbar (kein numerischer Wert für obere Grenze übergeben)

===== AUFGABE 3 (4 Punkte):

Schreiben Sie eine Prozedur p(L), die eine gegebene Liste L von integers analysiert, die Anzahl der ungeraden Primzahlen in L feststellt und diese Anzahl gemeinsam mit der Summe dieser Primzahlen in einer Liste (2 Elemente) zurückgibt.

Hinweis: - isprime(n::integer) testet auf Primzahleigenschaft (--> true/false)

Lösung: (Code der Prozedur p):

```
> p := proc(L::list)
  local iL, NL, number, counter, sump;
  NL := nops(L);
  counter := 0;
  sump := 0;
  for iL from 1 to NL do
    number := L[iL];
    if (number>2 and isprime(number)) then
      counter := counter + 1;
      sump := sump + number
    end if
  end do;
  return [counter,sump]
  end proc:
```

Beispiel:

```
> p([2,3,4,5,7,8,11,100]);
```

[4, 26]

===== 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: FAIL

-- L1, L2 haben gleiche Länge: [f,[*x*,*x*,...*x*]]
wobei *x* = true falls jeweilige Listenelemente übereinstimmen,
***x* = false sonst,**
und f = Anzahl der Nicht-Übereinstimmungen

Lösung (Code der Prozedur compare):

```
> compare := proc(L1::list,L2::list)
  local counter, iL, Lout;
  if (nops(L1)<>nops(L2)) then
    return FAIL
  else
    counter := 0;
    Lout := [seq(true,iL=1..nops(L1))]; # Initialisierung
    for iL from 1 to nops(L1) do
      if (L1[iL]<>L2[iL]) then
         Lout[iL] := false;
         counter := counter + 1
      end if
    end do;
    return [counter,Lout]
  end if
  end proc:
```

Beispiel:

```
> compare([1,2,3],[1,2,3,4]);
```

FAIL

```
> compare([A,B,C,D,E],[A,D,C,B,E]);
```

[2, [*true*, *false*, *true*, *false*, *true*]]

===== AUFGABE 5 (4 Punkte):

Schreiben Sie eine Prozedur horner(x,a), die
-- die Elemente a[0], a[1], ... eines gegebenen Arrays a
(n+1 Einträge, Startindex 0) als Koeffizienten eines
Polynoms interpretiert,
-- x als Auswertungsstelle interpretiert
-- und den Wert des Polynoms an der Stelle x
(also: summe von i=0...n: a[i]*x^i),
mittels des Horner-Schemas berechnet und zurückgibt.

Hinweis: -- Horner-Schema: Auswertung gemäß
a[0] + x * (a[1] + x * (a[2] + ...))
Überlegen Sie die Organisation der Schleife!
-- ArrayNumElems(...) liefert Anzahl der Elemente eines Arrays

Lösung (Code der Prozedur horner):

```
> horner := proc(x,a)
  local i, n, polval;
  n := ArrayNumElems(a)-1;
  polval := a[n];
  for i from n-1 to 0 by -1 do
    polval := x*polval + a[i]
  end do;
  return polval;
  end proc:
```

Beispiel:

```
> 1+2*y+4*y^3;
```

$$1 + 2\,y + 4\,y^3$$

```
> horner(y,Array(0..3,[1,2,0,4]));
```

$$y\,(4\,y^2 + 2) + 1$$

```
> expand(%);
```

$$1 + 2\,y + 4\,y^3$$

```
> horner(sqrt(2),Array(0..3,[1,2,0,4]));
```

$$10\,\sqrt{2} + 1$$

===== AUFGABE 6 (4 Punkte):

Schreiben Sie eine Prozedur newton(xstart,f::procedure,n,epsilon),
die für eine gegebene skalare reelle Funktion f(x)
n Schritte des Newton-Verfahrens durchführt (Nullstellensuche).
Die Iterationsvorschrift lautet: x[0]:=xstart, und

x[i+1] := x[i] - f(x[i]) / f'(x[i]), i=0,1,...n-1

Zurückzugeben sind alle Iterierten x[0],...x[n] in einem Array x.
Rechnung in Gleitpunktarithmetik (evalf(...)).

Der Parameter epsilon soll festlegen, wann
(nach Auswertung der Ableitungen mittels evalf)
die Iteration als 'gescheitert' abzubrechen ist, d.h.:

Eine Ableitung |f'(x[i])| < epsilon führt zu einem Abbruch (mit error).

Lösung (Code der Prozedur newton):

```
> newton := proc(xstart,f::procedure,n,epsilon)
  local fprime, i, x;
  x:=Array(0..n); # wird so mit 0 initialisiert
  x[0] := xstart;
  for i from 0 to n-1 do
    fprime := evalf(D(f)(x[i]));
    if (abs(fprime) < epsilon) then
       error "Iteration failed"
    else
       x[i+1] := evalf(x[i]-f(x[i])/fprime);
    end if;
  end do;
  end proc:
```

Beispiel (Quadratwurzel aus 2):

```
> newton(2,x->x^2-2,3,1e-10);
```

$$1.41421568627450980392156862745$$

```
> evalf(sqrt(2));
```

$$1.41421356237309504880168872421$$

```
> %-%%;
```

$$-0.21239014147551198799032 \cdot 10^{-5}$$

```
> newton(0,x->x^2-2,3,1e-10);
Error, (in newton) Iteration failed
```

===== AUFGABE 7 (4 Punkte):

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

Lösung:

siehe VO

===== AUFGABE 8 (4 Punkte):

Was tut die folgende Prozedur p ?
Erklären Sie genau, was p bei korrektem Aufruf zurückliefert.
Erklären Sie auch die error-Meldung.

```
> p:=proc(f::procedure,N::posint)
  local a,i,t;
  a:=Array(0..N+1);
  for i from 0 to N+1 do
     a[i]:= (D@@i)(f)(0)/i!;
     if (simplify((D@@i)(f)(t)=0)) then
        return i-1,a[0..i-1]
     end if;
  end do;
  error ("N too small? You may try to increase N.");
  end proc:
```

Lösung (Hinweis):

- p stellt fest, ob f ein Polynom vom (Maximal)Grad N ist.
- Grundlage: Test der höheren Ableitungen auf =0 (identisch 0 !);
  Berechnung der Koeffizienten nach dem Taylor'schen Satz.

"error": Falls N nicht groß genug gewählt, unentscheidbar. (Höherer Grad? kein Polynom?)

Beispiel:

```
> f:=x->1+x+5*x^5;
```

$$f := x \to 1 + x + 5\,x^5$$

```
> p(f,4);
Error, (in p) N too small? You may try to increase N.
```

```
> p(f,5);
```

$$5, \mathrm{Array}(0 .. 5, \{(0) = 1, (1) = 1, (5) = 5\}, \mathit{datatype} = \mathit{anything}, \mathit{storage} = \mathit{rectangular}, \mathit{order} = \mathit{Fortran\_order})$$

Achtung: Das ist nicht 100% verlässlich.. Beispiel:

```
> f := x->cos(3*arccos(x));
```

$$f := x \to \cos(3 \arccos(x))$$

```
> expand(f(x));
```

$$4\,x^3 - 3\,x$$

```
> p(f,3);
Error, (in p) N too small? You may try to increase N.
```

... wird von obigem Code nicht erkannt. (Das wäre eine weitere Aufgabe, deutlich subtiler.)