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

L Ö S U N G E N

* AUFGABE 1 (3 Punkte):

Finden Sie alle Syntaxfehler in der Prozedur:

```
> p : proc(x,y)
  local(a);
  a =: x+y;
  if (x is odd) then
    a =: -a;
  end then;
  return a;
  end proc:
Error, `(` unexpected
```

* Lösung:

6 Syntaxfehler: Korrekte Version:

```
> p := proc(x,y) # <--
  local a;       # <--
  a := x+y;      # <--
  if is(x,odd) then # <--
    a := -a; # <--
  end if;    # <--
  return a;
  end proc:
```

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

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

*** Lösung:**

```
> r := 5..1; # der 'range' von 5 bis 1 (rückwärts)
  whattype(r);
```

$$r := 5 .. 1$$

..

```
> A := [seq(n,n=r,-1)]; # Liste (n läuft von 5 abwärts bis 1)
```

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

```
> mul(a,a in A); # Produkt der Elemente
```

$$120$$

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

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

-- L1 leer und L2 leer: NULL

-- Ansonsten: L (als Liste)
wobei L[i] := max(L1[i],L2[i]), i=1,...
bzw. L[i] := FAIL falls L1[i] oder L2[i] nicht definiert.

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

```
> maxL := proc(L1,L2::list)
  local i,l1,l2, L:=[];
  l1,l2 := nops(L1),nops(L2);
  if (l1=0 and l2=0) then return NULL end if;
  for i from 1 to min(l1,l2) do
     L := [op(L),max(L1[i],L2[i])]
  end do;
  L:=[op(L),seq(FAIL,i=min(l1,l2)+1..max(l1,l2))];
  return L
  end proc:

> maxL([4,2,4,2],[2,3,2,3]);
                                [4, 3, 4, 3]
> maxL([1,2,3],[3,2,1,0]);
                                [3, 2, 3, FAIL]
> maxL([],[1,2,3,4]);
                                [FAIL, FAIL, FAIL, FAIL]
> maxL([],[]);
```

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

Schreiben Sie eine Prozedur cumsum(M), die zu einer gegebenen Menge M die Menge CM der kumulierten Summen der Elemente von M zurückliefert, dh. CM besteht aus den Elementen M[1], M[1]+M[2],... usw.
Falls M leer ist, ist auch die leere Menge zurückzugeben.

Vermeiden Sie unnötige Redundanz bei der Berechnung.

Achtung: Für eine Menge A ist eine Zuweisung der Form A[i] := ... unzulässig, da die Reihenfolge der Elemente bei Mengen nicht definiert ist.

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

```
> cumsum := proc(M::set)
  local cumvar:=0, i, CM:={};
  if M={} then return {} end if;
  for i from 1 to nops(M) do
     cumvar := cumvar+M[i];
     CM := CM union {cumvar}
  end do;
  return CM
  end proc:

> cumsum({1,2,3,4});
```

$$\{1, 3, 6, 10\}$$

```
> cumsum({a,b,c,c});
```

$$\{a, a+b, a+b+c\}$$

```
> cumsum({});
```

$$\{\ \}$$

* AUFGABE 5 (5 Punkte):

Schreiben Sie eine Prozedur fhalf(f::procedure,a,b,n), die zu einer gegebenen, als stetig und monoton wachsend vorausgesetzten Funktion eine Stelle x in [a,b] sucht mit der Eigenschaft f(x) = (f(a)+f(b))/2. (a<b wird angenommen; wird nicht extra überprüft.)

Lt. Vor. gilt f(a)<f(b), und es muss mindestens ein x mit der gewünschten Eigenschaft existieren. Setze [ax,bx]:=[a,b] und halbiere das Intervall [ax,bx] (Länge l=bx-ax). Stelle fest, in welchem der beiden Teilintervalle [ax,(ax+bx)/2] bzw. [(ax+bx)/2,bx] die Stelle x liegen muss. Setze demgemäß [ax,bx]:=[ax,(ax+bx)/2] bzw. [ax,bx]:=[(ax+bx)/2,bx].
Dies wird in einer Schleife iterativ n mal fortgesetzt, und zum Schluss wird das zuletzt erhaltene Intervall [ax,bx] als Liste zurückgegeben.

Rechnung in Gleitpunktarithmetik (evalf), weil sonst im allgemeinen der numerische Vergleich der Funktionswerte nicht durchführbar ist.

* Lösung (Code der Prozedur fhalf):

```
> fhalf := proc(f::procedure,a,b,n)
  local ax:=a,bx:=b, fm,i,xm;
  fm := (f(a)+f(b))/2;
  for i from 1 to n do
     xm := (ax+bx)/2;
     if (evalf(f(xm))<=evalf(fm)) then
        ax := xm
     else
        bx := xm
     end if
  end do;
  return [ax,bx]
  end proc:
> evalf(fhalf(x->x^2,0,1,10));
```

$$[0.7070312500, 0.7080078125]$$

Anmerkung: Ein Vergleich wie <= funktoniert 'direkt' nur für einfachste algebraische Ausdrücke. Im allgemeinen muss man das in Gleitpunktarithmetik durchführen:

```
> if (sqrt(2)<sqrt(3)) then dings else bums end if;
Error, cannot determine if this expression is true or false: 2^(1/2) < 3^(1/2)

> if evalb(sqrt(2)<sqrt(3)) then dings else bums end if;
Error, cannot determine if this expression is true or false: 2^(1/2) < 3^(1/2)

> if (sqrt(2.)<sqrt(3.)) then dings else bums end if;
```

dings

*** AUFGABE 6 (4 Punkte):**

Schreiben Sie eine Prozedur gradnorm(f::procedure,xi,eta,zeta),
die zu einer gegebenen reellwertigen Funktion f(x,y,z), f: R^3 -> R,
die euklidische Länge des Gradienten
(= Vektor der partiellen Ableitungen von f), d.h.

Länge des Vektors (f_x, f_y, f_z)

an der Stelle (x,y,z)=(xi, eta,zeta) zurückliefert.

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

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

```
> gradnorm := proc(f::procedure,xi,eta,zeta)
  local gradient,i,x,y,z,X;
  X := x,y,z;
  gradient := Vector([diff(f(X),x),diff(f(X),y),diff(f(X),z)]);
  for i from 1 to 3 do
     gradient[i] := subs(x=xi,y=eta,z=zeta,gradient[i])
  end do;
  # oder statt der Schleife:
  # eleganter so (subtilere Syntax)
  gradient := map2(subs,{x=xi,y=eta,z=zeta},gradient);
  return sqrt(add(gradient[i]^2,i=1..3));
  end proc:

> gradnorm((x,y,z)->x^3+y^3+z^3,x,y,z);
```

$$3\sqrt{x^4+y^4+z^4}$$

```
> gradnorm((x,y,z)->x^3+y^3+z^3,1,2,3);
```

$$21\sqrt{2}$$

*** AUFGABE 7 (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 x,y;
  y := (f(b)-f(a))/(b-a);
  x := evalf(solve(D(f)(x)=y));
  if evalb(x=NULL) then
    print ("nicht gefunden");
    return FAIL;
  else
    print("gefunden");
    return x;
  end if;
  end proc:
```

*** Lösung:**

1. Mittelwertsatz der Differentialrechnung

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

"gefunden"

0.5707963268

```
> m(x->1,-1,1); # Trivialbeispiel
  # Aber: 0=0 - solve kann damit nichts anfangen
```

"nicht gefunden"

FAIL

*** AUFGABE 8 (4 Punkte):**

Beschreiben Sie die wichtigsten Steuerkonstrukte für die Programmierung in Maple. Angabe einer möglichst allgemeinen Syntax wird besonders gewertet; konkrete Beispiele sind O.K.

*** Lösung:**

... if, for..do, while..do + Varianten