Maple-Test zur LVA 'Computermathematik', SS 2011

Gruppe bunt

***** ANMERKUNGEN UND LÖSUNGSHINWEISE *****

AUFGABE	PUNKTE
1	/ 2
2	/ 3
3	/ 3
4	/ 4
5	/ 5
6	/ 6
7	/ 3
8	/ 4
SUMME:	/ 30

*** AUFGABE 1 (2 Punkte):**

(a) Erklären Sie jeden einzelnen Befehl.
(b) Welchen Typ hat das Objekt b ?
(c) Wie lautet der Wert von c ?

```
> r := 1..5; a:=[$r];
```

$$r := 1 .. 5$$
$$a := [1, 2, 3, 4, 5]$$

```
> b := map(x->x^2,a);
```

$$b := [1, 4, 9, 16, 25]$$

```
> c := add(x,x in b);
```

$$c := 55$$

*** Lösung:**

(a) r ist der Range 'von 1 bis 5', $r die daraus generierte exprseq, a die entsprechende Liste.
Komponentenweise Anwendung der Quadratfunktion auf a ergibt b.
c ist die Summe der Elemente aus b.

(b) b ist eine Liste (so wie a).

(c) c = Summe(x^2,x=1..5) = 55

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

(a) Erklären Sie, was sie hier sehen.

```
> a:=2*a;
  a;
Error, recursive assignment
```

$$a$$

```
>
```

(b) Erklären Sie, was sie hier sehen.

```
> f[x] := x^2;
  f[x],f[y];
  eval(f);
```

$$f_x := x^2$$

$$x^2, f_y$$

$$\mathrm{table}([x = x^2])$$

*** Lösung:**

(a) a:=2*a, aber dann wäre a=4*a, also a=8*a usw., es wird rekursiv 'unendlich oft' eingesetzt, unzulässig. Ein Objekt kann nicht durch sich selbst definiert werden. Vergleiche jedoch:

```
> a:=1; a:=2*a;
```

$$a := 1$$

$$a := 2$$

(b) Es wird eine Tabelle (table) mit einem Eintrag generiert. f[x] ist definiert, aber kein anderer Eintrag. Die Tabelle besteht aus einem einzigen Element.

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

Was gibt diese Prozedur aus, wenn Sie wie unten spezifiziert, aufgerufen wird?

```
> p := proc(n::nonnegint)
  if n=0 then
     return 0
  else
     return n-p(n-1)
  end if
  end proc:

> seq(p(i),i=1..8):
```

*** Lösung:**

```
> seq(p(i),i=1..8);
```

1, 1, 2, 2, 3, 3, 4, 4

Der genaue Beweis für dieses Verhalten (für beliebieges n) wäre vollständige Induktion.

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

Formulieren Sie eine Prozedur shift(L,m,x), die die ersten m Elemente aus einer gegebenen Liste L entfernt, die restlichen Elemente um m Positionen nach vorne verschiebt und die m frei werdenden Plätze am Ende mit den Werten x,x+1,x+2,...,x+m-1 auffüllt.

Verwenden Sie keine expliziten Schleifen. Überprüfung auf sinnvolle Eingabeparameter ist nicht erforderlich.
ACHTUNG: Eine Zuweisung der folgenden Form ist für Listen nicht zulässig:

```
> L[1..3] := [a,b,c];
Error, invalid subscript on left hand side of assignment
```

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

```
> shift:=proc(L::list,m,x)
  return [op(L[m+1..nops(L)]),seq(x+i-1,i=1..m)]
  end proc:

> shift([1,2,3,4,5],3,y);
```

$$[4, 5, y, y+1, y+2]$$

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

Formulieren Sie eine Prozedur <u>partint(f,g,a,b),</u> die zu gegebenen Funktionen f,g und Zahlen a,b die Formel für die partielle Ingration zurückgibt, d.h.

Integral_a^b f(x)*g(x) dx = F(b)*g(b) - F(a)*g(a) - Integral_a^b F(x)*g'(x) dx (*)

wobei F intern als Stammfunktion von f und g' als Ableitung von bestimmt wird.

Die beiden Integrale in der Formel (*) sollen in nicht-ausgewerteter Form zurückgegeben werden.

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

```
> partint := proc(f,g,a,b)
  local F,gstrich;
  F := unapply(int(f(x),x),x);
  gstrich := D(g);
  return Int(f(x)*g(x),x=a..b) =
         F(b)*g(b)-F(a)*g(a) - Int(F(x)*gstrich(x),x=a..b)
  end proc:

> partint(sin,exp,0,1);
```

$$\int_0^1 \sin(x)\,\mathbf{e}^x\,dx = -\cos(1)\,\mathbf{e} + 1 - \int_0^1 -\cos(x)\,\mathbf{e}^x\,dx$$

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

Formulieren Sie eine Prozedur ccsum(L::list,m::nonnegint), die die m-fach kumulierte Summe der Einträge in der Liste L als gleich lange Liste zurückliefert.
Im Fall m=0 ist L selbst zurückzugeben.

Aber: Sobald die Folge der Werte streng monoton wachsend wird, wird die Schleife abgebrochen, ggf. auch bereits mit der gegebenen Liste L. Überprüfen Sie die Monotonie im Code nur an einer Stelle, und zwar in möglichst einfacher Weise.

Anmerkung: Die kumulierte Summe von L ist LC := [L[1],L[1]+L[2],L[1]+L[2]+L[3],...]
Die doppelt kumulierte Summe von L ist die kumulierte Summe von LC, usw.

Hinweis: Schleife über m (sonst keinerlei Schleifen).
Lassen Sie L unverändert und verwenden Sie eine interne Liste für die Zwischenspeicherung.

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

```
> ccsum := proc(L::list,m::nonnegint)
  local C,i,k,n:=nops(L);
  C := L;
  for i from 1 to m do
    if add(signum(C[i]-C[i-1]),i=2..n)=n-1 then
       break
    end if;
    C := [seq(add(C[i],i=1..k),k=1..n)]
  end do;
```

```
  return C
  end proc:


> ccsum([1,0,1,0,1],0);
                                                [1, 0, 1, 0, 1]
> ccsum([1,0,1,0,1],1);
                                                [1, 1, 2, 2, 3]
> ccsum([1,0,1,0,1],2);
                                                [1, 2, 4, 6, 9]
> ccsum([1,0,1,0,1],3); # bereits vorher streng monoton
                                                [1, 2, 4, 6, 9]
```

*** AUFGABE 7 (3 Punkte):**

Lineare Algebra (Vektoren, Matrizen, ..) in Maple/LinearAlgebra:
Schreiben Sie alles relevante auf, was Ihnen zu diesem Thema einfällt..

*** Lösung:**

```
> with(LinearAlgebra);
```

[*&x, Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix, BidiagonalForm, BilinearForm, CARE, CharacteristicMatrix, CharacteristicPolynomial, Column, ColumnDimension, ColumnOperation, ColumnSpace, CompanionMatrix, ConditionNumber, ConstantMatrix, ConstantVector, Copy, CreatePermutation, CrossProduct, DARE, DeleteColumn, DeleteRow, Determinant, Diagonal, DiagonalMatrix, Dimension, Dimensions, DotProduct, EigenConditionNumbers, Eigenvalues, Eigenvectors, Equal, ForwardSubstitute, FrobeniusForm, GaussianElimination, GenerateEquations, GenerateMatrix, Generic, GetResultDataType, GetResultShape, GivensRotationMatrix, GramSchmidt, HankelMatrix, HermiteForm, HermitianTranspose, HessenbergForm, HilbertMatrix, HouseholderMatrix, IdentityMatrix, IntersectionBasis, IsDefinite, IsOrthogonal, IsSimilar, IsUnitary, JordanBlockMatrix, JordanForm, KroneckerProduct, LA_Main, LUDecomposition, LeastSquares, LinearSolve, LyapunovSolve, Map, Map2, MatrixAdd, MatrixExponential, MatrixFunction, MatrixInverse, MatrixMatrixMultiply, MatrixNorm, MatrixPower, MatrixScalarMultiply, MatrixVectorMultiply, MinimalPolynomial, Minor, Modular, Multiply, NoUserValue, Norm, Normalize, NullSpace, OuterProductMatrix, Permanent, Pivot, PopovForm, QRDecomposition, RandomMatrix, RandomVector, Rank, RationalCanonicalForm, ReducedRowEchelonForm, Row, RowDimension, RowOperation, RowSpace, ScalarMatrix, ScalarMultiply, ScalarVector, SchurForm, SingularValues, SmithForm, StronglyConnectedBlocks, SubMatrix, SubVector, SumBasis, SylvesterMatrix, SylvesterSolve, ToeplitzMatrix, Trace, Transpose, TridiagonalForm, UnitVector, VandermondeMatrix, VectorAdd, VectorAngle, VectorMatrixMultiply, VectorNorm, VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip*]

```
>
> Vector,Matrix;
```

Vector, Matrix

```
> # etc.
```

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

Ein bisschen Analysis:
Nehmen Sie an, f steht für eine monoton wachsende Funktion.
-- Was ist der Zweck der Prozedur XXX, und was bedeuten die Argumente?
-- Was bedeutet die error-Meldung?
-- YYY ist eine 'Treiber-Prozedur' für XXX. Was ist der Zweck, d.h. wieso ruft man nicht gleich XXX auf?

```
> XXX := proc(f::procedure,value,a,b,epsilon,max_versuche,verbose,count)
  local  x:=0.5*(a+b), deviation;
  deviation := f(x)-value;
  if max_versuche=0 then
     error "Fehlschlag"
  else
     if (verbose='v') then
        printf("%3d  x=%20.12e, deviation=%10.2e \n",count,x,deviation)
     end if;
     if abs(deviation)<=abs(epsilon) then
        return x,deviation
     else
        `if`(deviation<0,
             XXX(f,value,x,b,epsilon,max_versuche-1,verbose,count+1),
             XXX(f,value,a,x,epsilon,max_versuche-1,verbose,count+1)
         )
     end if
  end if
  end proc:
```

```
  YYY:= proc(f::procedure,value,a,b,epsilon,max_versuche,verbose)
  XXX(f,value,a,b,epsilon,max_versuche,verbose,1)
  end proc:
```

*** Lösung:**

Die Prozedur XXX sucht im Intervall [a,b] nach einer Stelle für die f(x) dem vorgegebenen Wert value möglichst nahe kommt (Abweichung maximal epsilon).
Methode: Binäre Suche (Intervallhalbierung) unter der Annahme einer monoton wachsenden Funktion.
Falls verbose='v' angegeben wird, wird ein Protokoll der Suche ausgegegeben.

Rückgabewert: x, Abweichung f(x)-value

error-meldung: Nach max_versuchen Intervallhalbierungen kein Erfolg.

XXX ruft sich rekursiv auf und zählt den Parameter count hoch bis zu max_versuche.
YYY hat eine einfachere Parameterliste: count wird automatisch mit 1 initialisiert und XXX mit count=1 aufgerufen.

```
> YYY(x->x^2,0.5,0,1,1e-2,10,'v');
  1  x=  5.000000000000e-01, deviation= -2.50e-01
  2  x=  7.500000000000e-01, deviation=  6.25e-02
  3  x=  6.250000000000e-01, deviation= -1.09e-01
  4  x=  6.875000000000e-01, deviation= -2.73e-02
  5  x=  7.187500000000e-01, deviation=  1.66e-02
  6  x=  7.031250000000e-01, deviation= -5.62e-03
```

0.703125, -0.0056152344

```
>
```