Martin-Luther-Universität Halle-Wittenberg
Fachbereich Mathematik und Informatik
Dr. R. Picht

Telefon: +49 (345) 55-24 756
Email: picht@informatik.uni-halle.de

Halle, 09. Juni 2006

Programmiersprachen (SS 2006)

Übungsserie 10

Aufgabe 1 (Paket)

Das folgende Ada-Paket faßt einige physikalische Konstanten zusammen:

```
package physics is
 c: constant Float := 3.0e+8;
 G: constant Float := 6.7e-11;
 c: constant Float := 6.6e-3;
end physics;
```

Welche Bindungen treten auf?
Wie kann auf die Komponenten des Pakets zugegrifffen werden?

Aufgabe 2 (Rekursion)

Betrachtet werde der folgende Pascal-Block:

```
type Natural = 0..maxint;
var n, w : Natural;

procedure print(num, width : Natural);
    var digit : 0..9;
    begin
      if width = 0 then goto 9;
      if num>=10 then print(num div 10, width - 1);
      digit := num mod 10;
      write(output, chr(ord('0') +digit))
    end;
```

begin
...;
print(n, w);
...;
9:
...
end

Habe w den Wert 3 und n gleich 2006.
Was passiert, wenn der Sprung ausgeführt wird?

Aufgabe 3 (Modul)

Gegeben ist folgendes Ada-Paket mit trigonometrischen Funktionen:

```
package trig is
  function sin(x:Float) return Float;
  function cos(x:Float) return Float;
end trig;

package body trig is
 pi:constant Float:= 3.1416;

 function norm(x:Float) return Float is
   ...; --Berechne x modulo 2*pi


 function sin(x:Float) return Float is
   ...; --Berechne den Sinus von norm(x)


 function cos(x:Float) return Float is
   ...; --Berechne den Cosinus von norm(x)

end trig;
```

Was wird exportiert?
Welche Variablen bleiben verborgen?
Wie werden die exportierten Funktionen aufgerufen?

Aufgabe 4 (Modul)

Das folgende generische Paket kapselt in Ada eine beschränkte Warteschlange von Zeichen.

```
generic
   capacity : in Positive;
package queue_class is
   procedure append ( newitem : in Character);
   procedure remove ( olditem : out Character);
end queue_class;
```

```
package body queue_class is
   items : array (1..capacity) of Character;
   size, front, rear : Integer range 0..capacity;
   procedure append ( newitem : in Character) is
   ...; – Füge newitem am Schwanz der Schlange ein
   procedure remove ( olditem : out Character ) is
   ...; – Entferne olditem vom Anfang der Schlange
begin
...; – Leere die Warteschlange
end queue_class;
```

a) Ist das Paket parametrisiert und falls ja, wie geschieht die Parameterübergabe?

b) Welche Variablen/Objekte werden exportiert?

c) Welche Variablen/Objekte sind verborgen?

d) Das obige Paket enthält ein freies Vorkommen des Parameters Character, der den Typ von Einträgen der Warteschlange bezeichnet.
Das Paket hängt eigentlich nicht davon ab, ob die Einträge tatsächlich Zeichen sind. Verallgemeinern Sie, indem Sie auch in Hinsicht auf den Typ Item dieser Einträge parametrisieren.

e) Geben Sie einen qualifizierten Zugriff an, in dem Sie ein ' * ' an eine Warteschlange anhängen.

Aufgabe 5 (opaquer Typ)

Die Verwendung eines opaquen Typs in Modula-2 erfordert im allgemeinen den Export von `Assign-` und `Equal`-Prozeduren. Warum?

Schreiben Sie `Assign-` und `Equal`-Prozeduren für den folgenden Modul *ComplexNumbers* (Modula-2 ist der Nachfolger von Pascal, verwenden Sie eine pascal-ähnliche Syntax). Vervollständigen Sie den Implementierungsmodul.

```
  DEFINITION MODULE ComplexNumbers;

  TYPE COMPLEX;

  PROCEDURE Add(x, y: COMPLEX): COMPLEX;
  PROCEDURE Subtract(x, y: COMPLEX): COMPLEX;
  PROCEDURE Multiply(x, y: COMPLEX): COMPLEX;
  PROCEDURE Divide(x, y: COMPLEX): COMPLEX;
  PROCEDURE Negate(z: COMPLEX): COMPLEX;
  PROCEDURE MakeComplex(x, y: REAL): COMPLEX;
  PROCEDURE RealPart(z: COMPLEX): REAL;
  PROCEDURE ImaginaryPart(z: COMPLEX): REAL;

  END ComplexNumbers;

  IMPLEMENTATION MODULE ComplexNumbers;

  FROM Storage IMPORT ALLOCATE;

  TYPE  COMPLEX = POINTER TO ComplexRecord;
        ComplexRecord = RECORD
       re, im : REAL;
END;
  PROCEDURE Add(x, y: COMPLEX): COMPLEX;
  VAR t: COMPLEX;
  BEGIN
    NEW(t);
    t^.re := x^.re + y^.re:  (*  x^ Bezugsvariable von Zeigervariable x  *)
    t^.im := x^.im + y^.im;
    RETURN t;
  END Add;

  PROCEDURE Subtract(x, y: COMPLEX): COMPLEX;

  PROCEDURE Multiply(x, y: COMPLEX): COMPLEX;

  PROCEDURE Divide(x, y: COMPLEX): COMPLEX;

  PROCEDURE Negate(z: COMPLEX): COMPLEX;

  PROCEDURE MakeComplex(x, y: REAL): COMPLEX;

  PROCEDURE RealPart(z: COMPLEX): REAL;

  PROCEDURE ImaginaryPart(z: COMPLEX): REAL;
```

Schreiben Sie außerdem ein Modul *ComplexUser*, das den Typ `COMPLEX` benutzt, indem es seine Funktionen aus dem Modul *ComplexNumbers* importiert. Erzeugen Sie zwei komplexe Zahlen und multiplizieren Sie diese.

Aufgabe 6 (ADT)

Nehmen Sie an, Sie wollten einen Typ definieren, dessen Werte rationale Zahlen sind, mit *genau* arbeitenden arithmetischen Operationen (wie Addition). In ML könnten Sie dafür die folgenden Vereinbarungen schreiben:

```
datatype rational = rat of ( int * int)


val zero = rat (0, 1)
and one = rat (1, 1) ;
fun op ++ (rat (m1, n1): rational, (rat (m2, n2): rational) =
      rat (m1*n2 + m2*n1, n1*n2)
```

Welche unliebsame Eigenschaft hat diese Repräsentation?
Betrachten Sie dazu einen Vergleich von zwei rationalen Zahlen?

Besprechung ist am 13. Juni 2006.