Übungen zu Objektorientierte Programmierung WS 98/99

A.1) Namensäquivalenz, Strukturäquivalenz (2 min, 2 Punkte) richtig falsch

1.1) In Oberon wird bei Zuweisungen grundsätzlich Namensäquivalenz geprüft.
1.2) In Oberon gibt es neben reiner Namensäquivalenz auch noch Strukturäquivalenz, da beim Zuweisen an eine Prozedurvariable nur die Übereinstimmung der Parameterlisten geprüft wird.
1.3) In Oberon bieten Open-Array-Parameter die Flexibilität, daß als Aktualparameter beliebig lange Arrays (gleicher Dimensionen) übergeben werden können, deren Elementtypen strukturäquivalent sind (z.B. Übergabe einer Variable str32: ARRAY 32 OF CHAR an einen Parameter s: ARRAY OF CHAR).
1.4) Strukturäquivalenz bietet mehr Freiheiten, Namensäquivalenz schränkt die Freiheit des Programmierers stärker ein.

A.2) Polymorphismus, Erweiterung (2 min, 2 Punkte) richtig falsch

2.1) Polymorphismus ist grundsätzlich die Möglichkeit, daß eine Variable zur Übersetzungszeit auf mehrere Objekte zeigen kann.
2.2) Polymorphismus ist grundsätzlich die Möglichkeit, daß eine Variable auf Objekte verschiedener Typen verweisen kann.
2.3) Ein erweiterter Typ T1 kann mehr Felder enthalten als sein Basistyp T (T1 ist spezieller als T). Ebenso ist die Menge der T1-Objekte größer als die Menge der T-Objekte.
2.4) In Oberon können nur Zeigervariablen polymorph sein.

A.3) Typprüfung, Typtest und Type-Guard (2 min, 2 Punkte) richtig falsch

3.1) In Oberon werden nach der statischen Typprüfung zur Übersetzungszeit auch zur Laufzeit dynamische Typprüfungen durchgeführt.
3.2) In Oberon wird beim Methodenaufruf aufgrund des dynamischen Typs entschieden, ob das Empfängerobjekt die Methode versteht.
3.3) Bei der dynamischen Typprüfung (Typtest) wird zur Laufzeit überprüft, ob der statische Typ einer Variable einem gewünschten Typ entspricht.
3.4) Der Typeguard führt einen Typtest durch und garantiert (falls der Typtest zur Laufzeit erfolgreich durchgeführt werden konnte), daß der Ausdruck vom gewünschten Typ ist.

A.4) Fragen zur dynamische Bindung (2 min, 2 Punkte) richtig falsch

4.1) Dynamische Bindung heißt, daß erst zur Laufzeit feststeht, was ausgeführt wird.
4.2) Der Aufruf von Methoden (typgebundenen Prozeduren) ist in Oberon die einzige Möglichkeit dynamischer Bindung. Der Aufruf von Prozedurvariablen erfolgt über statische Bindung.
4.3) In Oberon wird der statische Typ einer Variable verwendet, um zu entscheiden ob ein Methodenaufruf möglich ist. Der dynamische Typ einer Variable wird verwendet, um zu entscheiden, welche konkrete Methode zur Laufzeit aufgerufen wird.
4.4) Supercalls, Klassenprozeduren und Prozeduraufrufe (ausgenommen Prozedurvariablen) können statisch gebunden werden.

A.5) Statischer Typ, dynamischer Typ, statische Bindung, dynamische Bindung (2 min, 2 Punkte) richtig falsch

5.1) Beim Typtest (z.B. t IS T2) wird der dynamische Typ überprüft.
5.2) Bei der Typzusicherung (z.B. t(T2)) wird der statische Typ überprüft.
5.3) Dynamisch allozierte Objekte (am Heap) sowie statische allozierte Objekte (modulglobal) und automatisch allozierte Objekte (lokal) haben für sich gesehen immer denselben statischen und dynamischen Typen.
5.4) Der dynamische Typ wird bei der Prüfung von Zugriffen auf Recordfelder verwendet (z.B. nach einer Zuweisung p0 := p1 sind über p0 nur mehr die Felder entsprechend dem dynamischen Typ von p0 zugreifbar).

B) Beispielprogramm (10 min, 10 Punkte)

Geben Sie an den gekennzeichneten Stellen die dynamischen Typen an. Veranschaulichen Sie die Programmausführung durch Skizzen der erzeugten Datenstrukturen. Geben Sie die bei der Ausführung des Modulrumpfs erzeugte Ausgabe an. Wenn das Programm syntaktische Fehler enthält, zeichnen Sie diese an. Wenn das Programm zur Laufzeit Fehler (Traps) produziert, kennzeichnen Sie diese Stellen und beschreiben Sie kurz den Grund dafür.

```
MODULE DynamicBinding;

TYPE
   T0 = RECORD END ; P0 = POINTER TO T0;
   T1 = RECORD (T0) END ; P1 = POINTER TO T1;
   T2 = RECORD (T1) END ; P2 = POINTER TO T2;

VAR
   p0: P0; p1: P1; p2: P2;

PROCEDURE (VAR t: T0) Dump;
BEGIN Out.String("T0.Dump"); Out.Ln
END Dump;

PROCEDURE (VAR t: T1) Dump;
BEGIN t.Dump^; Out.String("T1.Dump"); Out.Ln
END Dump;

PROCEDURE (VAR t: T2) Dump;
BEGIN t.Dump^; Out.String("T2.Dump"); Out.Ln
END Dump;

PROCEDURE X (s: T0; VAR t: T0; p: P0; VAR q: P0);
   VAR p1: P1; p2: P2; t1: T1; t2: T2;
BEGIN
   s.Dump; t.Dump; p.Dump; q.Dump;
   NEW(p1); p := p1; NEW(p2); q := p2; s := t2; t1 := t(T1)
END X;

BEGIN
   NEW(p0); NEW(p1); NEW(p2);
   (*  Dynamic type of p0:____   dynamic type of p1:____   dynamic type of p2:____  *)
   p0.Dump; p1.Dump; p2.Dump;
   X(p0^, p1^, p2, p0);
   (*  Dynamic type of p0:____   dynamic type of p1:____   dynamic type of p2:____  *)
   p0.Dump; p1.Dump; p2.Dump
END DynamicBinding.
```