Programmierparadigmen

Wintersemester 2012/13
Torsten Görg, Sandro Degiorgi

4. Übung
Einsendung bis spätestens 10.12.2012, 13:00 Uhr
Besprechungen am 12./13./14. Dezember 2012

Aufgabe 4.1 ..... 16 Punkte

Geben Sie zu jeder der folgenden Aussagen an, ob sie zutrifft oder nicht. Für die Bepunktung zählt der Gesamteindruck - es ist besser im Zweifelsfall keine Antwort zu geben als eine falsche.

1. Zur Minimierung des Speicherbedarfs ist eine verkettete Liste immer einem Array vorzuziehen.
2. Nur Java bietet mehrdimensionale Arrays.
3. Bei Programmiersprachen mit statischen Bindunsgkonzepten sind Deskriptoren unnötig und werden allenfalls fuer Optimierungen (z.B. in einem JIT) benötigt.
4. Die Konzepte `union` in C++ und `discriminated Records` in Ada sind semantisch äquivalent.
5. BestFit ist, wie der Name schon sagt, die beste bekannte Speicherverwaltungsstrategie.
6. Jedes Ausführungsmodell mit Single-Assignment Forderung kommt ohne Kontrollfluß aus.
7. Programmiersprachen mit statischen Aktivierungsblöcken erlauben keine Rekursion.
8. Bei einem parallelen Ausführungsmodell ist, dank schneller Cache Speicher, die Konsistenz der Daten gewährleistet.
9. Typdeskriptoren kommen bei statischer Typbindung nicht vor.
10. In C++ reicht es die Strukturäquivalenz der Typen zur Prüfung von Typäquivalenz zu nutzen.
11. Die offene Polymorphie von `templates` und `generics` kann und wird statisch aufgelöst.
12. Ada hat ein strikt monomorphes Typsystem.
13. C hat ein strikt monomorphes Typsystem.
14. Bindungsdeskriptoren werden immer zur Übersetzungszeit etabliert.
15. Dynamische Namesbindung ist deutlich ineffizienter als statische Namensbindung.
16. Das Konzept von Überladbarkeit ist zusammen mit den Objekt-orientierten Programmiersprachen entstanden.

Aufgabe 4.2 ..... 15 Punkte

Gegeben sei folgende Ada-Prozedur:

```
01  procedure Aufgabe_4_2 is
02     X : Integer;
03     type T1 is new Integer;
04     procedure P1 is
05        X : Integer := 23;
06        procedure P2( X : Integer ) is
07        begin
08           P1.X := X;
09        end P2;
10     begin
11        Aufgabe_4_2.P1.P2( X + 5 );
12     end P1;
13     M : T1 := 5;
14     N : T1;
15     type T2 is record
16        C : Float;
17        D : Integer;
18     end record;
19     R : T2 := ( 1.234, 7 );
20  begin
21     P1;
22     M := 3;
23     Aufgabe_4_2.X := M;
24  end Aufgabe_4_2;
```

1. Geben Sie zu obigem Ada-Programm alle Namens-, Typ- und Werte-Bindungen aller vorkommenden Variablen-Objekte an. Geben Sie weiter auch die Namensbindungen aller Prozeduren und Typdefinitionen an. Zu jeder Bindung sind auch die Nummern der Zeilen zu benennen, in denen diese Bindung gültig ist.
2. Wieviele Hierarchieebenen hat der hierarchische Namensraum in diesem Programm?
3. Geben Sie für alle in den Zeilen 08, 11, 21, 22 und 23 verwendeten Namen, die Variablen, formale Parameter oder Unterprogramme referenzieren, die Zeilennummern der Deklarationen an, an die diese Namen (statisch) gebunden sind.

Aufgabe 4.3 ..... 17 Punkte

Gegeben sei das folgende Ada Programm. Gehen Sie bei dieser Aufgabe davon aus, dass der Ada-Compiler einen Schalter zur Wahl der Namensbindung zur Verfügung stellt.

```
01  with Ada.Text_IO; use Ada.Text_IO;
02  procedure Namensbindung is
03    A : Integer := 1;
04    B : Integer := 2;
05    C : Integer := 3;
06    procedure Proc1 (X : Integer) is
07      A : Integer := X;
08    begin
09      Put_Line (Integer'Image (A + B + C));
10    end Proc1;
11    procedure Proc2 (X : Integer) is
12      B : Integer := X;
13    begin
14      Proc1 (A);
15      Put_Line (Integer'Image (A + B + C));
```

```
16    end Proc2;
17    procedure Proc3 (X : Integer) is
18      C : Integer := X;
19    begin
20      Proc2 (A);
21      Put_Line (Integer'Image (A + B + C));
22      Proc1 (A);
23    end Proc3;
24  begin
25    Proc1 (A);
26    Proc2 (A);
27    Proc3 (A);
28  end Namensbindung;
```

1. Benennen Sie die grundsätzlichen Nachteile von statischer Namensbindung und grundsätzlichen Nachteile von dynamischer Namensbindung.
2. Bestimmen Sie die Ausgabe des Programms unter der Voraussetzung von a) statischer Namensbindung bzw. b) dynamischer Namensbindung.
3. Charakterisieren Sie für die Ausführung dieses Programms den Gültigkeitsbereich, die Sichtbarkeit und die Lebensdauer der Variablen `A`, `B` und `C` unter der Voraussetzung von a) statischer Namensbindung bzw. b) dynamischer Namensbindung.
4. Modifizieren Sie das gegebene Programm so, dass es sich unabhängig von der Art der Namensbindung immer so verhält a) wie zuvor bei der statischen Namensbindung bzw. b) wie zuvor bei der dynamischen Namensbindung.

Aufgabe 4.4 ..... 11 Punkte

Bei rekursiv definierten Typen stößt die Strukturäquivalenz an ihre Grenzen. In der Vorlesung wurden zwei Möglichkeiten vorgestellt, dieses Problem zu lösen. Untersuchen Sie dazu den folgenden Pseudocode.

```
type T1;
type T2 = record
            c1 : integer;
            c2 : T1;
          end record;
type T1 = pointer to T2;
type T3 = record
            c3 : integer;
            c4 : T1;
          end record;
type T4 = pointer to T2;
type T5 = record
            c5 : integer;
            c6 : T4;
          end record;

a : T1;
b : T2;
c : pointer to T2;
d : T3;
e, f : pointer to T3;
g : T4;
h : T5;
i : T3;
```

1. Welche der Variablen `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h` und `i` sind entsprechend der Strukturäquivalenz zueinander typäquivalent? Setzen Sie dabei zunächst voraus, dass zur Lösung der Rekursionsproblems Strukturen nicht strukturell sondern entsprechend der Namensäquivalenz verglichen werden, wie z.B. in C/C++.
2. Geben Sie die strukturellen Typäquivalenzklassen erneut an, wobei diesmal Zeiger nicht strukturell sondern entsprechend der Namensäquivalenz verglichen werden.

Aufgabe 4.5 ..... 21 Punkte

Gegeben sei folgendes Python-Programm. Beachten Sie, dass Python eine Skriptsprache mit Duck-Typing ist. Information zur Objektorientierung und zum Duck-Typing in Python finden Sie z.B. unter `http://www.tutorialspoint.com/python/python_classes_objects.htm`.

```
01   class Rectangle:
02      def draw( self ):
03         print "Drawing a rectangle (width=", self.width, ", \
04                                     height=", self.height, ")."
05
06   class Circle:
07      def draw( self ):
08         print "Drawing a circle (radius=", self.radius, ")."
09
10   obj1 = Rectangle()
11   obj1.width = 500
12   obj1.height = 300
13   obj2 = Circle()
14   obj2.radius = 123
15
16   obj1.draw()
17   obj2.draw()
```

1. Welche grundsätzlichen Unterschiede bestehen zwischen Duck-Typing und strenger, statischer Typbindung?
2. Welche Typen sind mit den durch `obj1`, `obj2`, `width`, `height` und `radius` bezeichneten Variablen-Objekten und welche mit den in ihnen gespeicherten Werten assoziiert?
3. Welche Methoden werden in dem obigen Python-Programm in den Zeilen `16` und `17` aufgerufen? Und welche Ausgaben liefert das Programm bei seiner Ausführung?
4. Implementieren Sie ein zu dem gegebenen Python-Programm äquivalentes Java-Programm, wobei Sie die in Java notwendigen Typinformationen in Form von expliziten Deklarationen, durch Hinzufügen eines gemeinsamen Interfaces und mit Attributangaben in den Klassen ergänzen. Typisieren Sie `obj1` und `obj2` mit dem neuen Interface-Typ.
5. Um den Duck-Typing-Mechanismus für Objektattribute nachzubilden, soll nun eine Java-Klasse `DynamicAttributeSet` mit den Methoden `getattr`, `hasattr`, `setattr` und `delattr` unter Verwendung einer Java-`HashMap` ausprogrammiert werden (in Anlehnung an die gleichnamigen in Python verfügbaren Operationen). Ein Gerüst der Klasse mit Spezifikationen der Methoden finden Sie in `DynamicAttributeSet.java` auf der Homepage.
6. Erstellen Sie eine weitere Java-Portierung der gegebenen Python-Programms, die (im Gegensatz zu Aufgabenteil 4.) die Duck-Typing-Methodenaufrufe mit Hilfe von Java-Reflection nachbildet. Ferner sind die Objektattribute und lokalen Variablen mit der Klasse `DynamicAttributeSet` aus dem vorigen Aufgabenteil zu verwalten, so dass Attribut- und Variablenzugriffe durch Methodenaufrufe ersetzt sind. Hinweise zur Java Reflection API finden Sie unter `http://docs.oracle.com/javase/tutorial/reflect/index.html`.