Programmierparadigmen

Wintersemester 2012/13
Torsten Görg, Sandro Degiorgi

3. Übung
Einsendung bis spätestens 26.11.2012, 13:00 Uhr
Besprechungen am 5./6./7. Dezember 2012

Aufgabe 3.1 ..... 14 Punkte

In fast jeder Programmiersprache ist das Definieren und Verwenden von Array-Datentypen vorgesehen, so auch in Ada, Java und C++. Recherchieren Sie in den folgenden Sprachspezifikationen, was jeweils zu mehrdimensionalen Arrays, zu möglichen Array-Indexbereichen und darüber, ob Array-Objekte auf dem Stapel oder auf der Halde allokiert werden, festgelegt ist. Geben Sie zu jeder der drei Sprachen auch jeweils eine Eigenschaft von Array-Typen oder -Objekten an, in der sie sich von den anderen beiden Sprachen unterscheidet.

1. Ada 2005: `http://www.adaic.org/standards/05rm/RM-Final.pdf`
   Schauen Sie sich hier insbesondere Kapitel 3.6 an.
2. Java: `http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf`
   Schauen Sie sich hier insbesondere Kapitel 10 an.
3. C++: `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf`
   Schauen Sie sich hier insbesondere Kapitel 8.3.4 an.

Aufgabe 3.2 ..... 18 Punkte

Im Folgenden sind mehrere Typdefinitionen gezeigt, die jeweils entweder in Ada, in Java oder in C++ formuliert sind. Identifizieren Sie bei jeder Typdefinition zunächst, in welcher dieser drei Sprachen sie ausgedrückt ist, und geben dann möglichst äquivalente Typdefinitonen jeweils in den beiden anderen Sprachen an.

1.
```
enum Chromatisch {c, cis, d, dis, e, f, fis, g, gis, a, ais, b}
enum Dauer {Ganz, Halb, Viertel, Achtel, Sechzehntel}
struct Note {
   Chromatisch zwoelfer;
   Dauer takt_dauer;
   bool hoerbar;
};
```

2.
```
type Sechzehn new Integer range 1..16;
type ZwoelfTakter is new Integer range 1..12;
type Takt is array(Sechzehn) of Note;
type Takte is array (ZwoelfTakter) of Takt;
```

```
type Arrangement is record
   titel      : String(1..42);
   jahr       : Integer;
   interpret  : String(1..42);
   schema     : Takte;
   improv     : Boolean;
end record;
```

3.
```
class Takt {
   public int noten[];
   public Takt()   { noten = new int[16]; }
}
class Blues {
   public Takt takte[];
   public Blues()  { takte = new Takt[12]; }
}
```

4.
```
enum KoordinatenArt { kartesisch, polar };
struct Position {
   KoordinatenArt Art;
   union {
      struct {
         float x;
         float y;
      };
      struct {
         float winkel;
         float abstand;
      };
   };
};
```

Aufgabe 3.3 ..... 18 Punkte

Implementieren Sie für folgende physikalischen Formeln Ada-Funktionen, die jeweils die Größe auf der linken Seite der Formelgleichung berechnen und die dafür notwendigen variablen Werte für die rechte Gleichungsseite als Funktionsparameter entgegennehmen. Entwerfen Sie dazu für die physikalischen Größen jeweils geeignete Typdefinitionen, so dass Ihre Berechnungsfunktionen möglichst sicher und zuverlässig angewendet werden können und potentielle Fehler beim Aufrufen dieser Funktionen möglichst schon durch das Typsystem verhindert werden. Hinweis: Die Umsetzung einer Formel muss nicht zwingend in nur einer Ada-Funktion erfolgen, sondern kann ggf. auch durch mehrere Funktionen realisert werden.

1. $P = U \cdot I$
   mit P in Watt (W), U in Volt (V) und I in Ampere (A)

2. $Q = I \cdot t$
   mit Q in Coulomb (C), I in Ampere (A) und t in <Zeiteinheit>, <Zeiteinheit> ist Sekunden (s), Minuten (min) oder Stunden (h)

3. $\vec{E} = \vec{F}/Q$
   mit $\vec{E} = (E_x, E_y, E_z)$, $\vec{F} = (F_x, F_y, F_z)$, Vektorkomponenten je in Newton (N), Q in Coulomb (C)

4. $s = v \cdot t$
   mit s in Meter (m), v in <Wegeinheit>/<Zeiteinheit> und t in <Zeiteinheit> (siehe 2.), <Wegeinheit> ist Meter (m), Fuß (ft) oder Seemeilen (sm)

Die Maßeinheiten können in folgender Weise umgerechnet werden:
$W = V \cdot A$, $C = A \cdot s$, $1h = 60m = 3600s$, $ft = 0,30479m$, $sm = 1852m$, $N = 10^3 g \cdot m/s^2$ und $N/C = N/(A \cdot s) = (V \cdot N)/(W \cdot s) = V/m$

Aufgabe 3.4 ..... 12 Punkte

Analysieren Sie die folgenden, in Pseudocode formulierten Typ- und Variablendefinitionen. Welche der Variablen `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`, `j`, `k`, `l` und `m` sind zueinander typäquivalent? Geben Sie die Typäquivalenzen zum einen unter der Voraussetzung von reiner struktureller Typäquivalenz (7 Punkte) und zum anderen für reine Namens-Typäquivalenz (5 Punkte) an.

```
type Musikinstrument = record
                          hersteller : string(1..100);
                          baujahr    : integer;
                       end record;

type Automobil = record
                    hersteller : string(1..100);
                    baujahr    : integer;
                 end record;

type Geigen   = pointer to array [1..12] of Musikinstrument;
type Menschen = pointer to array [1..12] of record
                                               name : string(1..100);
                                               age  : integer;
                                            end record;
type Link_1   = pointer to array [1..12] of record
                                               location : string(1..100);
                                               members_count : integer;
                                            end record;
type Garagen_Part = array [1..12] of Automobil;
type Link_2 = pointer to Garagen_Part;

a : Musikinstrument;
b : Link_1;
```

```
c : pointer to array [1..12] of Automobil;
d, e : Link_1;
f : pointer to Garagen_Part;
g : Link_2;
h : record
       x : string(1..100);
       y : integer;
    end record;
i, j : pointer to array [1..11] of Automobil;
k : Link_2;
l : pointer to array [1..11] of Musikinstrument;
m : Garagen_Part;
```

Aufgabe 3.5 ..... 18 Punkte

Im Rahmen dieser Aufgabe soll die Heap-Speicherverwaltung mittels des Reference-Counting-Mechanismus betrachtet werden. Auf der Veranstaltungs-Homepage finden Sie dazu einige vorbereitete Ada-Quelltextdateien.

1. Vervollständigen Sie die Implementierung des Reference-Counting-Mechanismus, indem Sie in `reference_counting.adb` die Operationen `Finalize`, `Assign`, `Get_Child_Ref` und `Set_Child_Ref` entsprechend der Kommentierung in der Paketspezifikation (`reference_counting.ads`) und unter Verwendung der Hilfsprozeduren `Increment_References_Count` und `Decrement_References_Count` ausprogrammieren.
   Hinweis: Informationen zu Controlled-Types in Ada finden Sie z.B. unter `http://www.adaic.org/resources/add_content/standards/05rm/html/RM-7-6.html`.

2. Fügen Sie nun im Paket `Reference_Counting` eine Prozedur `Dump_Ref_Count_Objects` hinzu, die zu allen aktuell durch das `Reference_Counting`-Paket verwalteten Objekten den Wert ihres `Value`-Attributs und die Anzahl der auf das Objekt verweisenden Referenzen auf der Console ausgibt, und implementieren diese. Bauen Sie dazu im Paket `Reference_Counting` einen Mechanismus ein, der sich merkt, welche Objekte aktuell verwaltet werden.

3. Auf der Homepage befindet sich in `reference_counting_test.adb` ein Testprogramm für die Reference-Counting-Implementierung. Welche Ausgaben liefern die Aufrufe von `Dump_Ref_Count_Objects` an den mit `A`, `B` und `C` markierten Programmstellen? Erklären Sie zu allen ausgegebenen Referenzzählerwerten, wie diese zu Stande gekommen sind. Wenn Sie die ersten beiden Aufgabenteile nicht gelöst haben, überlegen Sie sich, welche Ausgaben erscheinen müssten.

4. Konstruieren Sie ein Beispielprogramm, das über das `Reference_Counting`-Paket Objekte allokiert, die auch nach Freigabe aller Variablen mit Referenzen, die von außen auf diese Objekte gezeigt haben, durch den Reference-Counting-Mechanismus bis zum Programmende nicht abgeräumt werden. Machen Sie dieses Programmverhalten durch geeignete Aufrufe von `Dump_Ref_Count_Objects` sichtbar.