

# Konzepte der Programmiersprachen

Sommersemester 2007

Eduard Wiebe

## Übungsblatt 3

### Organisatorisches

Dieses Übungsblatt wird am 8. Juni besprochen.

### Aufgabe 3.1: Laufzeitprüfung

Schutz gegen fehlerhaftes Überschreiben des Speichers und Einhaltung der in Deklarationen gegebenen Zusicherungen über Wertebereiche kann durch entsprechende Übersetzungs- und Laufzeitprüfungen erreicht werden.

Fügen Sie in das eingegrenzte Ada-Programmstück („von hier“ – „bis hier“) alle Prüfungen ein, die den notwendigen Laufzeitprüfungen entsprechen, um derartige Sicherheit zu garantieren.

Die Prüfungen sollen von der folgenden Form sein:

```
if Verletzungsbedingung then raise Error; end if;
```

Nehmen Sie dabei an, dass die statische Typprüfung erfolgreich war und dass alle beteiligten Variablen Werte besitzen, die ihren (Sub-)Typen entsprechen, aber zur Übersetzungszeit noch nicht bekannt sind.

```
type Transaktions_Art is (Einzahlung, Auszahlung);
type Waehrung is new Integer;
type Konto_Nummer is new Integer range 0 .. 10**9;
type Buchungseintrag (Art : Transaktions_Art) is
  record
    Konto : Konto_Nummer;
    case Art is
      when Einzahlung =>
        Gutschrift : Waehrung range 0 .. 10**6;
        ...
      when Auszahlung =>
        Abzug : Waehrung range -10**6 .. 0;
        ...
    end case;
  end record;
type Buchung is access Buchungseintrag;
type Buchungs_Liste is
  array (Integer range 0 .. Vorgang_Max) of Buchung;

procedure Verarbeite
  (Tagesliste : Buchungs_Liste; Vorgang : Integer;
   Konto      : Konto_Nummer; Betrag  : Waehrung;
   Kosten     : Waehrung) is
begin
  ...
  ----- von hier
  Tagesliste (Vorgang).Konto := Konto;
  Tagesliste (Vorgang).Gutschrift := Betrag - Kosten;
  ----- bis hier
  ...
end Verarbeite;
```

### **Aufgabe 3.2: Freund-Konzepte**

In neueren Programmiersprachen treten sogenannte „Freund“-Konzepte auf.

1. Welche generelle Semantik wird durch sie realisiert?
2. Welches Benutzerproblem wird durch ein solches Konzept gelöst?
3. Skizzieren Sie ein Beispiel, in dem das Freund-Konzept sinnvoll angewandt wird.

### **Aufgabe 3.3: Vererbung, Varianz**

Im Bereich der objektorientierten Programmierung gibt es die Begriffe der Kovarianz, Invarianz und Kontravarianz der Signaturen von Methoden.

1. Charakterisieren Sie mit einigen Sätzen jeden dieser Begriffe.
2. Welche Zusammenhänge gibt es mit der Präfix- und der Parameternotation für Methodenaufrufe?
3. Welche Vor- und Nachteile hat Kovarianz für ererbte Methodenimplementierungen?

### **Aufgabe 3.4: Wertesemantik und lokale Variablen**

In einer Programmiersprache sei folgende Semantik gegeben:

- Es gibt Klassen, die Unterklassen besitzen können. Unterklassen können gegenüber ihren Oberklassen zusätzliche Datenattribute und Operationen besitzen.
- In Unterprogrammen sind lokale Variablen zugelassen, deren deklariertes Typ eine Klasse ist. Klassen sind auch als deklarierter Typ von globalen Variablen und Parametern sowie Funktionsresultaten erlaubt.
- Die Zuweisung hat Wertesemantik. Den obigen Variablen können Instanzen einer beliebigen Unterklasse zugewiesen werden, wobei alle Datenattribute der Unterklasse erhalten bleiben.

Daraus ergeben sich allerdings einige Probleme.

1. Welches Implementierungsproblem tritt bei der Realisierung solcher lokaler Variablen auf? Wie kann es ohne Änderung der Sprachsemantik gelöst werden? Gibt es unvermeidbare Nachteile?
2. Können die Schwierigkeiten aus der vorigen Teilaufgabe durch Änderung der Sprachsemantik vermieden werden, ohne dabei die „typischen“ Charakteristika objektorientierter Sprachen zu verlieren?

### Aufgabe 3.5: Klassen in C++

Das folgende C++-Programm berechnet Primzahlen. Nach welchem Algorithmus funktioniert dieses Programm? Beschreiben Sie die verwendete Klassenhierarchie. Welche Regeln gibt es in C++ den Zugriff auf die Komponenten einer Klasse?

```
#include <iostream>

class Knoten {
public:
    Knoten *Eingabe;
    Knoten (Knoten *s) { Eingabe = s; }
    virtual int Ausgabewert () { return 0; }
};

class Generator : public Knoten {
    int Wert;
public:
    int Ausgabewert(void) { return Wert++; }
    Generator(int Startwert) : Knoten (0) { Wert = Startwert; }
};

class Filter : public Knoten {
    int Faktor;
public:
    int Ausgabewert ();
    Filter (Knoten *s, int f) : Knoten(s) { Faktor = f; }
};

int Filter::Ausgabewert () {
    int n;
    do {
        n = Eingabe->Ausgabewert();
    } while (!(n % Faktor));
    return n;
}

int main(void) {
    Generator C (2);
    Knoten *Sieb = &C;
    int Primzahl, max;
    std::cout << "Primzahl-Sieb in C++\n\nWieviele Primzahlen? ";
    std::cin >> max;
    for (int i = 0; i < max; i++) {
        Primzahl = Sieb->Ausgabewert();
        std::cout << Primzahl << " ";
        Sieb = new Filter (Sieb, Primzahl);
    }
    std::cout << "\n";
    return 0;
}
```