Prof. Dr. Th. Letschert
FB MNI

17. Oktober 2003

Aufgabenblatt 2 - NPP

Aufgabe 1

1. Starten Sie einen Browser, besuchen Sie die Seite `www.python.org`, sehen Sie sich dort etwas um und suchen Sie nach einem Python–Tutorium. Gibt es eine lokale Kopie des Tutoriums?

2. Welchen Namen hat der "Erfinder" von Python.

3. Stellen Sie fest, wie der Python–Interpretierer aktiviert wird und lassen Sie ihn $3 * 17 + 12$ berechnen.

4. Schreiben Sie folgendes Programm ("Skript") in eine Datei:

   ```
   x = input()
   y = input()
   p = x*y
   print ' Produkt: ', p
   ```

   und führen Sie es mit dem Python–Interpretierer aus. Kann das Programm auch Zeichenketten verarbeiten und in welchem Format werden sie eingegeben?

5. Welche Wirkung hat

   ```
   x = [[1,2,3],[4,5],[[[]]]]
   y = x[0]
   x,y = y,x
   ```

   Informieren Sie sich über die verwendeten Konstrukte.

Aufgabe 2

1. Sie definieren zwei Listen:

   ```
   >>> l1 = [1, 2, 3]
   >>> l2 = l1 + [4, 5]
   ```

   Jetzt ändern Sie die Wert–Zuordnung von `l1`, z.B. mit

   ```
   >>> l1[0] = 100
   ```

   wie verändert sich dadurch `l2`?
   Welche Auswirkung hat es, wenn der Wert von `l1` direkt an `l2` zugewiesen wird

   ```
   >>> l1 = [1, 2, 3]
   >>> l2 = l1
   >>> l1[0] = 100
   ```

Überlegen Sie welches Verhalten des Interpretieres Ihrer Meinung nach sinnvoll ist und testen Sie dann, wie er sich tatsächlich verhält.

Kommentieren/erklären Sie!

2. Was sind Tupel in Python? Kann dieses Beispiel mit Tupeln wiederholt werden?

3. Was versteht man unter "referenzieller Transparenz". Ist das etwas Positives oder Negatives? Hat C++ oder Python die Eigenschaft der referenziellen Transparenz.

4. Welche Ausgabe erzeugt folgendes Python–Programm:

```
i = 3
def fun(x):
    return i+x

l = [1, 2, 3, 4, 'Boo', 5]

for i in l:
    print fun(i)
```

Erläutern Sie mit diesem Beispiel im Vergleich zu einem äquivalenten C++–Programm die Konzepte der *statischen* (C++) und der *dynamischen* (Python) *Bindung*. Definieren Sie diese Begriffe. Was versteht man unter "statisch" und "dynamisch" im Bereich der Programmiersprachen? Ist die beobachtete dynamische Bindung mit dem Sprachkonzept von Python vereinbar?

Aufgabe 3

Folgendes C++–Programm legt in jedem Durchlauf einer Schleife jeweils eine neue Liste an und gibt die letzte Liste aus.

```
#include <list>
#include <string>
#include <iostream>

using namespace std;

class GenElem {
public:
  virtual ~GenElem () {}                              // warum ?
  virtual void print(ostream & os) = 0;
  virtual GenElem * clone()        = 0;
};

class StringElem : public GenElem {
public:
  StringElem (const string & s) : _s(s) {}
  StringElem * clone() { return new StringElem(_s); } // wieso ?
  void print(ostream & os) { os << _s; }
private:
  string _s;
};
```

```
class IntElem : public GenElem {
public:
  IntElem (int i) : _i(i) {}
  IntElem * clone() { return new IntElem(_i); }
  void print(ostream & os) { os << _i; }
private:
  int _i;
};

class Elem {
  friend ostream & operator<< (ostream &, const Elem &);
public:
  Elem (const Elem & elem) : _e(elem._e->clone()) {}  // weshalb ?
  Elem & operator= (const Elem & elem) {
    if ( &elem != this ) {
      delete(_e);
      _e = elem._e->clone();
    }
    return *this;
  }
  Elem (const string & s) : _e(new StringElem (s)) {}
  Elem (int i)            : _e(new IntElem(i)) {}
  ~Elem () { delete(_e); }                          // warum ?
private:
  GenElem * _e;
};

ostream & operator<< (ostream & os, const Elem & le) {
  le._e->print(os);
  return os;
}

typedef list<Elem> List;

int main() {
  List ll;
  for (int i = 0; i < 1000000; i++) {
    List l;
    l.push_back(Elem("H"));
    l.push_back(Elem("u"));
    l.push_back(Elem("g"));
    l.push_back(Elem("o"));
    l.push_back(Elem(" "));
    l.push_back(Elem("?"));
    l.push_back(Elem("!"));
    for (int j = 0; j < 20; j++) {
      l.push_back(Elem(j));
    }
    ll = l;                                         // was genau wird kopiert
  }
  for (List::iterator i = ll.begin();
       i != ll.end();
```

```
        ++i ) {
      cout << *i << endl;
    }
  }
```

Die Liste kann Elemente unterschiedlichen Typs aufnehmen. Das macht sie “generisch”. Erläutern Sie wie diese Generizität hier (in einer typisierten prozeduralen OO–Sprache mit Polymorphismus) implementiert wurde.

Erläutern Sie die Speicherverwaltungs–Mechanismen des Programms: welche Mechanismen werden von C++ selbst, welche werden von der Standardbibliothek und welche werden von diesem Programm geliefert?

Erläutern Sie die Kopiersemantik des Programms: wie wird sie im Zusammenwirken von Sprache, STL, und diesem Programm realisiert.

Übersetzen Sie dieses Programm und lassen sie es laufen. Messen sie dabei die Laufzeit.

Folgendes Python–Programm führt die gleichen Aktionen aus wie das C++ Programm, inklusive Speicherverwaltung und Kopieraktionen:

```
import copy
for i in xrange(1000000):
    l=['H','u','g','0',' ','?','!']
    for j in xrange(20):
        l.append(j)
    ll = copy.deepcopy(l)
print ll
```

Informieren Sie sich über die verwendeten Konstrukte und erläutern Sie, welche Eigenschaften von Python es erlauben, dass dieses Programm so viel kompakter als die C++–Version ist. Erläutern Sie die Generizität der Liste, die Speicherverwaltung, sowie die Kopiersemantik. In Python wird die tiefe Listenkopie im Anwendungcode explizit angefordert. Ist das notwendig, hätte nicht eine einfache Zuweisung die gleiche Wirkung? Kopiert das C++–Programm tief. Wenn ja, wo ist das definiert oder ausprogrammiert?

Lassen Sie den Interpretierer das Python–Programm ausführen und messen Sie die Ausführungszeit.

Ein äquivalentes Java–Programm ist:

```
import java.util.Vector;
import java.util.Iterator;

public class listTest {
    public static void main(String[] args) {
        Vector ll = new Vector();
        for (int i = 0; i < 1000000; i++) {
            Vector l = new Vector();
            l.addElement("a");
            l.addElement("b");
            l.addElement("c");
            l.addElement("d");
            l.addElement("e");
            l.addElement("f");
            l.addElement("g");
```

```
            for (int j = 0; j < 20; j++) {
                l.addElement(new Integer(j));
            }
            ll = (Vector)l.clone();
        }
        Iterator i = ll.iterator();
        while ( i.hasNext() ) {
            System.out.println(i.next());
        }
    }
}
```

Erläutern Sie dessen Speicherverwaltung, die Realisation der generischen Listen und die Kopieraktionen im Vergleich zur C++– und Python–Version.

Messen Sie die Laufzeit des Programms.

Aufgabe 4

1. Welche Wirkung hat folgendes Python–Programm

```
def f(x):
    return 2*x
def g(x):
    return 3*x
def h(f, g):
    return lambda x:f(x)+g(x)

list = input()
for i in list:
    print h(f, g)(i)
```

   Mit welchen Eingaben kann das Programm arbeiten.

2. Welche Wirkung hat folgendes Programm bei der Eingabe von `[1, 'Hallo', 3]`:

```
def f(x):
    return x+x
def g(x):
    return 3*x
list = input()
for i in list:
   print ( lambda x : (lambda p, q : p(x)+q(x))(f, g)) (i)
```

3. Transformieren Sie obiges Beispiel in ein äquivalentes Programm in dem die Aufrufe von `f` und `g` durch Lambda–Ausdrücke ersetzt werden.

4. Analysieren Sie die Typstruktur und berechnen Sie die Werte und Typen folgender Ausdrücke:

   $(\lambda z.(\lambda y.y + z))(2)$

   $(\lambda x.(\lambda y.(\lambda x.x + y))(x + 1))(5)$

   $(\lambda f.f(\lambda x.x + x))(\lambda g.g(g(1)))$

   $(\lambda f.f(\lambda x.x))(\lambda h.h(1))$

5. Übersetzen Sie, so weit das möglich ist, diese Ausdrücke in die Notation von Python und lassen Sie den Python–Interpretierer ihren Wert berechnen. Welche Aktionen sind mit dem jeweiligen Wert möglich: Zuweisung an eine Variable, Ausgabe, Aufruf als Funktion (mit welchem Argument beispielsweise)? Geben Sie wenn möglich äquivalente einfachere Ausdrücke oder Funktionsdefinitionen an.

6. Definieren Sie informal und in Python eine Funktion $compose$, derart, dass $compose(f, g)$ eine Funktion h liefert mit $h(x) = f(g(x)) = compose(f, g)(x)$.

7. Definieren Sie die bekannte Fibonacci-Funktion mit Parametermustern.

Aufgabe 5

Es sei eine Funktion f wie folgt definiert

$$\mathbf{letrec}\ f\ =\ \lambda\, x\,.\,\mathbf{if}\ x = 0\,\mathbf{then}\,1\,\mathbf{else}\,f(x)$$

1. Welchen Wert hat der Ausdruck

   $(\lambda x, y\,.\,\mathbf{if}\ x = 0\,\mathbf{then}\,1\,\mathbf{else}\,y)(0, f(2))$?

   Hängt dieser Wert von der Auswertungsreihenfolge ab?

2. Kann f in Python definiert werden?

3. Kann, mit diesem f, der Ausdruck

   $(\lambda x, y\,.\,\mathbf{if}\ x = 0\,\mathbf{then}\,1\,\mathbf{else}\,y)(0, f(2))$

   in Python geschrieben und eventuell sogar ausgewertet werden?

4. Welche Auswertungsstrategie hat Python: Wie nennt man sie? Welche Alternativen gibt es? Ist die Auswertungsstrategie von Python “normal” – d.h. so wie die von C++ oder Java? Ist sie effizient? Wie hätten Sie die Sprache in dieser Hinsicht konzipiert?