Prof. Dr. Th. Letschert
FB MNI

21. Oktober 2003

Aufgabenblatt 3 - NPP

Aufgabe 1

1. Sind Python, C++, Java Sprachen mit dynamischer oder mit statischer Typisierung.
2. Warum gelten Programme in Sprachen mit dynamischer Typisierung als fehlerträchtig.
3. Erläutern Sie, wie in Python und in C++ jeweils die "typfreie" Funktionen `add` realisiert wird. Gehen Sie dabei auf Speicherung und Verarbeitung von Typinformationen in den verschiedenen Phasen der Verarbeitung ein.

```
Python:                     C++:

def add(x, y):              template<typename T>
    return x + y            T add (T x, T y) {
                                return x+y;
                            }
```

4. Sind Templates oder Überladung in C++ Beispiele für dynamische Typisierung?
5. Wurden in der Vorlesung "Compilerbau" Sprachen mit statischer oder solche mit dynamischer Typisierung besprochen?
6. Warum wird oft die Meinung vertreten, dass Sprachen mit dynamischer Typisierung speziell (oder nur) für das *rapid prototyping* geeignet sind?
7. Warum gelten die Autoren von Programmen in Sprachen mit dynamischer Typisierung gelegentlich als von etwas minderer Geistesstärke, obwohl sie oft sehr produktiv sind.
8. Kann eine Liste in Python eine Mischung aus Zahlen und Listen von Zahlen enthalten?
9. Mit welchem Typsystem müssen Sie ein C++–Programm austatten, wenn es eine Liste enthalten soll, in der eine Mischung aus Zahlen und Listen von Zahlen vorkommt? Erläutern Sie mit einem UML–Diagramm!

Aufgabe 2

1. Schreiben Sie in Python eine Funktion die folgender Definitionen in Lamba–Notation entspricht:

   $\textbf{let } f1 \ = \ \lambda x.\lambda f.f(x)$

   und geben Sie eine geeignete Anwendung dieser Funktion an.
2. Schreiben Sie in Python eine Implementierung von

   $\textbf{let } genFunAdder \ = \ \lambda f.\lambda x, y.f(x) + f(y)$
3. Informieren Sie sich über die `map`–Funktion von Python. Geben Sie ein Beispiel an, in dem diese Funktion benutzt wird. Welchen Typ hat `map`?

Aufgabe 3

1. Folgendes Beispiel enthält eine Funktion `map`, deren Funktionsparameter als Objekt "verkleidet" wurde. Modifizieren Sie das Beispiel weiter, derart, dass es keinerlei freie Funktionen mehr enthält. Machen Sie also auch `map` zu einem "funktionalen Objekt".

```
template<typename T>
class FO {
public:
  void operator() (T x) {
    cout << x << endl;
  }
};

void map (list<int> l, FO<int> fo) {
  for ( list<int>::iterator i = l.begin();
        i != l.end();
        ++i )
    fo(*i);
}

int main() {
  list<int> l;
  for ( int i = 0; i<10; ++i)
    l.push_back(i);
  FO<int> fo;
  map (l, fo);
}
```

2. Welche Rolle spielt & in folgender Definition (warum ist es erforderlich), insbesondere wenn `FuncObj<T>` eine Basisklasse darstellt?

```
template<typename T>
void map (list<T> l, FuncObj<T> & f) {
  for ( typename list<T>::iterator i = l.begin();
        i != l.end();
        ++i )
    f(*i);
}
```

3. Geben Sie ein Python–Programm an, das folgendem C++–Programm entspricht

```
class Add {
public:
  Add (int p_x) : x(p_x) {}
  int operator() (int y) { return x+y; }
private:
  int x;
};
Add genAdd (int x) {
  return Add(x);
}
int main() {
```

```
  Add add2 = genAdd(2);
  cout << add2(3) << endl;
}
```

Aufgabe 4

1. Betrachten Sie folgendes Python–Programm:

```
def compose(f, g):
    return lambda x: g(f(x))

fl = [lambda x: 2*x, lambda x: 3*x, lambda x: 4*x]

F = lambda x:x
for f in fl:
    F = compose(F, f)

print F(2)
```

   Welche Wirkung hat es, was passiert?

2. Vergleichen Sie das obige mit dem folgenden Programm. Hat es die gleiche Wirkung? Sollte es die gleiche Wirkung haben?

```
def compose(f, g):
    return lambda x: g(f(x))

F = lambda x:x
for i in [2, 3, 4]:
    F = compose(F, lambda x : i*x)

print F(2)
```

3. Funktionale Objekte werden auch in Python als *Closures* erzeugt, d.h. als Ausdrücke, die in einer bestimmten Umgebung ausgewertet werden. Diese Umgebung bestimmt die Bedeutung (Bindung) der freien Bezeichner. Welche Bezeichner sind im zweiten Beispiel frei und wie werden sie gebunden?

4. Geben Sie zum zweiten Python–Programm ein äquivalentes C++–Programm an, in dem funktionale Objekte dynamisch erzeugt und dynamisch, mit einer Funktion `compose`, zu neuen Funktionsobjekten verknüpft werden.

5. Die funktionalen Objekte in Ihrem C++–Programm sind ebenfalls als *Closures* zu verstehen. Wie sieht es hier mit der Bindung aus. Liefert Ihr C++–Programm den Wert des ersten, oder des zweiten Python–Programms, oder eventuell einen völlig anderen Wert?