

Übungen zur Vorlesung
"Objektorientiertes Programmieren im Wissenschaftlichen Rechnen"

Dr. Olaf Ippisch, Ole Klein
ole.klein@iwr.uni-heidelberg.de

Abgabe am 30. 04. 2013 in der Vorlesung

Auf diesem Übungsblatt soll der Umgang mit Pointern und dynamischer Speicherverwaltung wiederholt werden.

ÜBUNG 1 POINTER

`i` habe den Typ `int`, und `p` den Typ `int *`. Welcher der folgenden Ausdrücke ist korrekt bzw. nicht korrekt? Geben Sie zu den korrekten Ausdrücken den Typ an.

- | | |
|----------------------------|------------------------------|
| • <code>i + 1</code> | • <code>&p</code> |
| • <code>*p</code> | • <code>p + 1</code> |
| • <code>*p + 3</code> | • <code>&p == i</code> |
| • <code>&i == p</code> | • <code>**(&p)</code> |
| • <code>i == *p</code> | • <code>*p + i > i</code> |

2 Punkte

ÜBUNG 2 DESTRUKTOR

Welche der folgenden Aussagen ist richtig? Der Destruktor einer Klasse `C` ist dafür verantwortlich ...

- ...alle Objekte der Klasse `C` wegzuräumen.
- ...Objekte der Klasse `C` im Heap wegzuräumen.
- ...alle Komponenten von Objekten der Klasse `C` wegzuräumen.
- ...Komponenten von Objekten der Klasse `C`, die im Heap liegen, wegzuräumen.
- `delete` ist nichts anderes, als eine spezielle Art den Destruktor aufzurufen: `x` sei vom Typ `C*`, dann ist `delete x`; das Gleiche wie `(*x).~C()`

2 Punkte

ÜBUNG 3 NEW & DELETE

1. Warum ist:

```
int * get_int1 () {  
    int * p;  
    p = new int;  
    return p;  
}
```

eine sinnvolle Methode einen Verweis auf neue `int` Variable zu erzeugen, dagegen

```
int * get_int2 () {  
    int i;  
    int * p = &i;  
    return p;  
}
```

völlig ungeeignet?

2. Es seien die Definitionen / Anweisungen:

```
int * p;  
p = new int;  
*p = 17;
```

ausgeführt worden. Warum sind die darauf folgenden Aufrufe

```
p = 0;  
delete p;
```

unsinnig, dagegen aber

```
delete p;  
p = 0;
```

sinnvoll?

4 Punkte

ÜBUNG 4 VERKETTETE LISTE

An dem einfachen Beispiel einer verketteten Liste üben wir das Zusammenspiel von Konstruktoren, Destruktoren und Zeigern.

Wir wollen eine verkettete Liste programmieren, welche eine beliebige Anzahl von `int` Werten speichern kann. Eine Liste besteht aus einem Objekt der Klasse `List`, die auf eine Folge von Objekten der Klasse `Node` verweist. In den Knoten sind die Listenelemente in einer Komponente `int value` gespeichert und einen Pointer `Node * next` zeigt auf den nächsten Knoten. Das Ende der Liste ist daran erkennbar, dass der `next` pointer den Wert `0` hat.

1. Was ist besonderes daran, wenn ein Pointer den Wert `0` hat?
2. Implementieren Sie die Klasse `Node`. Achten Sie dabei darauf, dass alle Member-Variable immer initialisiert sind.
3. Implementieren Sie die Klasse `List` mit folgende Methoden:

```
class List {
public:
    List (); // create an empty list
    ~List (); // clean up the list and all nodes
    Node * first() const; // return a pointer to the first entry
    Node * next(const Node * n) const; // return a pointer to the node after n
    void append (int i); // append a value to the end of the list
    void insert (Node * n, int i); // insert a value before n
    void erase (Node * n); // remove n from the list
};
```

`List` muss zusätzlich den Anfang der Liste speichern. Um sicher zu stellen, dass nicht versehentlich die Listenstruktur außerhalb der Klasse `List` geändert wird, soll der `next` Pointer der `Node` Klasse `private` sein, `value` ist `public`, damit man die Werte auch lesen und schreiben kann. Damit `List` auf den `next` Pointer der `Node` Klasse zugreifen kann, muss folgende Zeile in die Deklaration der `Node` Klasse eingefügt werden: `friend class List;`
Stellen Sie außerdem sicher, dass der Destruktor alle `Node` Objekte wieder löscht.

4. Testen Sie ihre Implementierung mit folgendem Programm:

```
int main () {
    List list;
    list.append(2);
    list.append(3);
    list.insert(list.first(), 1);
    for (Node * n = list.first(); n != 0; n = list.next(n))
        std::cout << n->value << std::endl;
    return 0;
}
```

5. Was passiert, wenn man die Liste kopiert? Und was passiert, wenn die beiden Listen wieder gelöscht werden?

```
int main () {
    List list;
    list.append(2);
    ...
    List list2 = list;
    return 0;
}
```