

Zeiger (Pointer), Referenzen und const

Tobias Sielhorst

31 October 2006

chair for computer aided medical procedures

department of computer science | technische universität münchen

Typen

Ein Zeiger (MyObject*)

- zeigt auf einen Speicherbereich
- Es gibt Nullzeiger (Nullpointer)

```
MyObject object;  
MyObject* objectPointer;  
  
MyObject& ref = object;
```

Eine Referenz (MyObject&)

- ist wie ein neuer Name für ein bestehendes Objekt
- Es gibt keine Nullreferenz

JAVA kennt nur die Mischung aus beiden

Umwandlung

- Die gleichen Zeichen werden bei Umwandlungen benutzt
- & vor einem Objekt bedeutet

„Ich hätte gern den Speicherort vom Objekt“

- * vor einem Zeiger (bzw. Speicherort) bedeutet

„Ich hätte gern die Referenz (bzw. das Objekt) zum Zeiger“

```
MyObject object;  
MyObject* objectPointer;  
  
objectPointer = &object;  
object = *objectPointer;
```

Wozu das ganze? Geschwindigkeit!

- Objekte beliebige Größe
- Pointer → 4 Byte
Referenzen → 4 Byte

```
Telefonbuch t;
string suchbegriff;
Eintrag e;
e = suchfunktion(suchbegriff, t);
```

```
Eintrag suchfunktion(string s, Telefonbuch t)
{ // kopiert beim Aufruf das ganze Telefonbuch
  ...
  return s;
}
```

```
Eintrag suchfunktion(string& s, Telefonbuch& t)
{ // kopiert nur 4 Byte
  ...
  return s;
}
```

```
Eintrag& suchfunktion(string& s, Telefonbuch& t)
{ // wenn man den Eintrag ändert, ändert sich
  // auch der im Telefonbuch!
  ...
  return s;
}
```

Warum also Zeiger?

```
MyParentClass* mp;  
MyClass mc;  
  
mp = &mc; // geht  
  
MyParentClass& mr = mc; //Kompilierfehler
```

- In vielen Fällen möchte man ein Objekt über eine höhere Vererbungsklasse (parent class) ansprechen. Dies geht nur mit Zeigern

- Effizientes Programmieren mittels Zeigerarithmetik

```
// ein Byte Array (10MByte) um den Wert 5 erhöhen  
char* array = new char[10000000];  
for (int i=0; i <10000000;i++){  
    array[i]+=5;  
}  
delete[] array;
```

- folgendes Programm ist x-mal schneller

```
char* array = new char[10000000];  
char* endOfArray = array+10000000;  
for (char* i=array; i <endOfArray;i++){  
    (*i)+=5;  
}  
delete[] array;
```

Der Keller und die Halde (Stack und Heap)

- Es gibt zwei Möglichkeiten wo man Speicher für seine Objekte bekommt:

```
MyObject mo1; // Stack  
MyObject* mo2 = new MyObject(); // Heap
```

- Beim Stack wird der Speicher wieder überschrieben, nachdem die Variable aus dem lokalen Namensraum verschwindet
- Beim Heap wird der Speicher erst wieder freigegeben, wenn man die Funktion `delete` aufruft
- Bei JAVA kommt alles auf den Heap und wenn es nicht mehr gebraucht wird vom GarbageCollector wieder freigegeben



Tipps zu Zeigern (Pointer) und Referenzen

- Man nimmt Pointer, wenn man Vererbungsverhältnisse zur Laufzeit ausnutzen möchte
- Man nimmt Pointer, wenn man ausdrücken möchte, dass eventuell der Nullpointer in der Variable ist
- Man nimmt Referenzen, wenn man sich absolut sicher ist, dass das Objekt vorhanden ist
- Wenn man `new` schreibt, sollte man sich angewöhnen auch gleich das `delete` dazu zu schreiben
- Man sollte sich angewöhnen, alle Pointer mit 0 zu initialisieren und vor `delete` auf 0 zu überprüfen
- Zu `new MyObject[42]` gehört `delete []`
- Destruktoren sollten immer virtuell sein (`virtual ~MyObject` im Header)

Wie sage ich es meinem Kollegen? Mit `const` deklarieren

- Ich biete eine Funktion an, bei der die Übergabeobjekte nicht verändert werden

```
Eintrag& meineFunktion(const Telefonbuch& t, const string& s);
```

- Ich biete eine Funktion an, bei dem keine Membervariablen (auch nicht indirekt) geändert werden

```
Eintrag& meineFunktion(meinObject& mo1, meinObject& mo2) const;
```

- Ich biete eine Funktion an, bei dem der Übergabe nicht geändert werden darf, damit nicht schlimmeres passiert

```
const Eintrag& meineFunktion(meinObject& mo1, meinObject& mo2);
```

Tipps zu const

- Der Compiler prüft, ob der Code zu den `const` Deklarationen korrekt ist und wirft einen Fehler (keine Warnung) falls nicht
- Man muss nicht mit `const` programmieren, aber es hilft
- Mit einem Cast kann man dem Compiler sagen, dass man doch auf eine konstante Variable zugreifen möchte
- `const` immer von rechts nach links lesen!
 - `const MyObject*` ist ein Pointer auf ein konstantes Objekt
 - `MyObject const *` ist ein konstanter Pointer auf ein Objekt
 - `const MyObject const *` ist eine konstanter Pointer auf ein konstantes Objekt