Beherrschen Sie C++ ?

Robert Hoschek

Selbst Software-Entwickler mit langjähriger Erfahrung müssen zugeben, daß sie manchmal auf Aufgabenstellungen stoßen, die sich als harte Nuß erweisen. So mancher C++ Guru greift dann zum Buch oder stellt seine Frage in eine einschlägige Newsgroup im Internet.

Beherrschen Sie C++ ? Dann überlegen Sie sich einmal folgende Aufgabe:

Sie möchten eine Klasse `MyClass` definieren und einem anderen Programmierer zur Verfügung stellen, allerdings mit der Einschränkung, daß sie nur am Heap erzeugt werden darf. Dadurch wird der Stack des Programms nicht belastet, was für die gestellte Aufgabe wichtig ist. Der Programmierer soll Ihre Klasse also folgenderweise verwenden:

```
MyClass *pmclass = new MyClass();   //
Instanz am Heap anlegen
…
delete pmclass;           // Instanz vom
Heap entfernen
Dagegen soll folgende Verwendung nicht
erlaubt sein:
MyClass mclass; // Instanz der Klasse am
lokalen Stack erzeugen
```

Natürlich können Sie nicht verhindern, daß jemand diese Zeile eingibt. Sie müssen den Compiler dazu bringen, daß er die Zeile als falsch zurückweist, weil er sie nicht kompilieren kann. Wie Sie sicher wissen, haben Objekte, die am lokalen Stack erzeugt werden, ein angenehmes - zumindest klar definiertes - Verhalten: Sie werden beim Verlassen des Gültigkeitsbereichs, also spätestens am Programmende, gelöscht. Das Löschen eines Objekts ist aber immer mit dem Aufruf des Destruktors verbunden. Was nun aber, wenn der Compiler den Destruktor nicht aufrufen kann? Sehen Sie sich die Definition der Klasse (bzw. das für uns interessante Fragment) an:

```
class MyClass {
public:
     MyClass();
…
private:
    ~MyClass();
};
```

Während der Konstruktor `public`, d.h. von außen zugänglich ist, ist der Destruktor `private` deklariert. Er kann von außen, in unserem Fall vom Compiler, nicht aufgerufen werden. Da der Compiler also die Instanz der Klasse beim Verlassen des Gültigkeitsbereichs nicht richtig behandeln kann, verweigert er die Kompilierung. Warum - können Sie jetzt fragen - kann man dann das Objekt am Heap erzeugen und löschen? Dazu muß man nur überlegen, was `delete` eigentlich bedeutet. `delete` ist ein Operator, also eine Memberfunktion unsere Klasse. Und diese hat Zugriff auf private Memberfunktionen.

Jetzt nehmen wir einmal den umgekehrten Fall an; Sie wollen den Heap verbieten und den Anwender Ihrer Klasse zwingen, den lokalen Stack zu verwenden. Damit vermeiden Sie z.B. Probleme mit Speicherbereichen, die nicht mehr freigegeben werden. Kann man das (mit Unterstützung des Compilers) erzwingen?

Man kann. Unsere Klasse sieht jetzt folgendermaßen aus:

```
class MyClass {
private:
static void * operator new (size_t size);
static void operator delete (void *ptr);
…
};
```

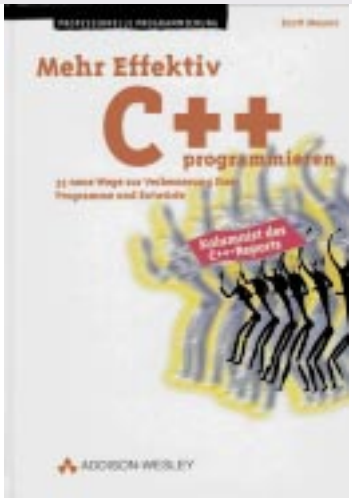
Wir haben den `new` und `delete` Operator als `private` deklariert. Sie können also von außen nicht mehr aufgerufen werden. Folglich muß der Versuch

```
MyClass *pmclass = new MyClass();   //
Instanz am Heap anlegen
```

scheitern.

War das ein Thema, das Sie interessiert? Dann empfehle ich Ihnen die Lektüre von Scott Meyers „Mehr Effektiv C++ programmieren", in diesem Buch finden Sie alles, was Ihre Programmierung effektiver macht.

Scott Meyers: „Mehr Effektiv C++ programmieren: 35 neue Wege zur Verbesserung Ihrer Programme und Entwürfe", Addison-Wesley-Longman, 1997, ISBN 3-8273-1275-2



Interaktive Neuronen

Norbert Bartos

Multimediale Lernsoftware ist nun endlich auch für das an neuronalen Netzen interessierte Publikum verfügbar. Die vorliegende CDROM von Autoren der Universität Dortmund enthält ein gut aufgebautes und klares Trainingsprogramm für das Gebiet der neuronalen Netze, welches keinerlei zusätzliche Kenntnisse in Elektronik, Medizin oder Computertechnik voraussetzt. Auch das mathematische Wissen der Anwender wird nur sparsamst beansprucht. Die Zielgruppe sind Studierende und Berufstätige der Fachrichtung Wirtschaftsinformatik. Als Beilage ist ein Begleitheft mit etwa 50 Seiten vorhanden, welches die Fähigkeiten des Systems übersichtsartig in kurzer und einfacher Form erläutert. Die im Kurs verwendeten Anwendungsbeispiele spiegeln allgemein verständlichen Problemstellungen wider, wie zum Beispiel die Kreditwürdigkeit von Bankkunden oder die Materialbedarfsprognose für Fertigungsbetriebe. Ein Balken am unteren Bildschirmrand gibt an, wieviel Prozent des aktuellen Kapitels bereits abgearbeitet sind. Mathematisch tiefergehende Formeln und Erklärungen können nach Bedarf explizit angefordert werden und belasten das Erscheinungsbild des Kurses für nur oberflächlich Interessierte daher nicht. Jedes Kapitel enthält am Ende eine Anzahl von Prüfungsfragen, die beantwortet werden können. Bei falschen Antworten wird auf das entsprechende Kapitel verwiesen, welches dann wiederholt werden sollte.

Es werden in den fünf Kapiteln die folgenden Aspekte behandelt:

1 Geschichtliche Aspekte der neuronalen Netze

2 Biologische Zusammenhänge und formales Neuronenmodell (Funktion eines künstlichen Neurons)

3 Arbeitsweise von neuronalen Netzen (einschließlich Datencodierung, Lernen und Testen)

4 Netztopologien (Perzeptron, Backpropagation Netzwerk, Hopfield-Netz, Simulated Annealing)

5 Anwendungsbeispiele (Personalbeurteilung, Materialbedarfsprognose, Kreditwürdigkeit)

Insgesamt kann gesagt werden, dass diese CDROM als Einführungskurs für Einsteiger auf dem Gebiet der neuronalen Netze durchaus empfehlenswert ist, unabhängig davon, ob sie eine technische Vorbildung besitzen oder nicht. Bei der Bedienung des Systems sollten wegen seiner Einfach-

heit keine gröberen Probleme auftauchen, allerdings waren beim Test durch einige Studierende des Transferlehrganges am TGM trotzdem auch Systemabstürze zu verzeichnen.

Die Daten der CDROM (in einer buchartigen Kartonhülle, gemeinsam mit einem Begleitheft):

Richard Lackes, Dagmar Mack, Neuronale Netze - Prinzipien und Anwendungen, (Interaktive Wirtschaftsinformatik), Addison-Wesley, 1997, ISBN 38273-1081-4, ATS 605,-