7.3 Dynamischer Speicher in Objekten/Zuweisungsoperator

7.3.1 Weitere Probleme mit dynamischem Speicher in Objekten

7.3.1.1 Beispiel 1: keine Probleme

- Beispiel: `c++ mitglied1.* main4.cpp`
- `mitglied1.h`:

```
1  // mitglied1.h
2
3  #ifndef _MITGLIED_H_
4  #define _MITGLIED_H_
5
6  class mitglied
7  {
8  private:
9      char *name;
10     char *motto;
11 public:
12     mitglied(char *nname="", char *nmotto="");
13     ~mitglied(void);
14     mitglied(const mitglied &mg);
15     void setName(char *nname);
16     char *getName(void);
17     void setMotto(char *nmotto);
18     char *getMotto(void);
19     void printit(void);
20 };
21 #endif
```

- `mitglied1.cpp`:

```
1  // mitglied1.cpp
2  #include <iostream.h>
3  #include <string.h>
4  #include "mitglied1.h"
5  //-----------------------------------------------------
6  mitglied::mitglied(char *nname="", char *nmotto="")
7  {
8      cout << "mitglied::ctor(";
9      if(!nname) nname="";
10     if(!nmotto) nmotto="";
11     cout << nname << ")" << endl;
12
13     name = new char[strlen(nname)+1];
14     strcpy(name, nname);
15
16     motto = new char[strlen(nmotto)+1];
17     strcpy(motto, nmotto);
18     cout << "...ok!" << endl;
19 }
20 //-----------------------------------------------------
21 mitglied::~mitglied(void)
22 {
```

```
23      cout << "mitglied::dtor("<< name <<")" <<endl;
24      delete[] motto;
25      delete[] name;
26      cout << "...ok!" << endl;
27  }
28  //-------------------------------------------------------
29  mitglied::mitglied(const mitglied &mg)
30  {
31      cout << "mitglied::cpctor(" << mg.name << ")" << endl;
32      name = new char[strlen(mg.name)+1];
33      strcpy(name, mg.name);
34      motto = new char[strlen(mg.motto)+1];
35      strcpy(motto, mg.motto);
36      cout << "...ok!" << endl;
37  }
38  //-------------------------------------------------------
39  void mitglied::setName(char *nname)
40  {
41      if(!nname) nname="";
42      if(name) delete[] name;
43      name = new char[strlen(nname)+1];
44      strcpy(name, nname);
45  }
46  //-------------------------------------------------------
47  char *mitglied::getName(void){ return name;}
48  //-------------------------------------------------------
49  void mitglied::setMotto(char *nmotto)
50  {
51      if(!nmotto) nmotto="";
52      if(motto) delete[] motto;
53      motto = new char[strlen(nmotto)+1];
54      strcpy(motto, nmotto);
55  }
56  //-------------------------------------------------------
57  char *mitglied::getMotto(void){ return motto;}
58  //-------------------------------------------------------
59  void mitglied::printit(void)
60  {
61      cout << "Ich heisse.....:" << name  << endl;
62      cout << "Mein Motto ist:" << motto << endl;
63  }
64  //-------------------------------------------------------
```

- `main4.cpp`:

```
1  // main4.cpp
2  #include <iostream.h>
3  #include "mitglied1.h"
4  int main(int argc, char *argv[])
5  {
6      mitglied a("heinz", "Laber-Rabaaba");
7      mitglied b=a;
8      mitglied c("willi", "Keine Ahnung-kein Problem");
9      mitglied d("egon",  "Ohne Preis kein Fleiss");
```

```
10
11      b.setName("hans");
12      a.printit();
13      b.printit();
14      c.printit();
15      d.printit();
16      return 0;
17  }
```

– Ergebnis: 4x ctor, 4x dtor

7.3.1.2 Änderung des Programms: Zuweisung

- `c++ mitglied1.cpp main5.cpp`
- `main5.cpp:`

```
1  // main5.cpp
2  #include <iostream.h>
3  #include "mitglied1.h"
4  int main(int argc, char *argv[])
5  {
6      mitglied a("heinz", "Laber-Rabaaba");
7      mitglied b("fritz", "Langeweile kennt keine Grenzen");
8      mitglied c("willi", "Keine Ahnung-kein Problem");
9      mitglied d("egon",  "Ohne Preis kein Fleiss");
10
11     d=c;
12     d.setName("erich");
13     a.printit();
14     b.printit();
15     c.printit();
16     d.printit();
17     return 0;
18 }
```

- Verhalten: Destruktor von c schlägt wieder fehl.
- Das Problem liegt offenbar in der Zeile:

```
1  d=c;
```

7.3.1.3 Woher rührt dieses Problem?

- Antwort: Durch die Zuweisung von c nach d wird jede Komponente von c nach d kopiert, auch die Zeiger (*name und *motto).

7.3.1.4 Wie kann man das ändern?

- Der automatisch erzeugte Zuweisungsoperator (das einfache Gleichheitszeichen) muss ersetzt werden.

7.3.2 Wie wird der Zuweisungsoperator aufgerufen?

- Im Gegensatz zu den Konstruktoren und Destruktoren wird der Zuweisungsoperator (wie alle weiteren Operatoren) explizit aufgerufen.

```
d=c;                  // C-Standard; wird in C++
                      // interpretiert als Kurzform
d.operator=(c);       // dieser Form
```

7.3.3 Wie baut man einen eigenen Zuweisungsoperator?

- Der automatisch erzeugte Zuweisungsoperator kopiert nur Element für Element (so genannte flache Kopie).
- Er kann durch einen selbstgebauten Zuweisungsoperator ersetzt werden.
- Sein Name und seine Parameter lauten:

```
void <klasse>::operator=(const <klasse> &<objekt>)
void mitglied::operator=(const mitglied &mg);
```

- "Normale" Zuweisungsoperatoren (d=c) weisen nicht nur c nach d zu, sondern sie liefern auch einen Rückgabewert, der gleich c ist. Das kann man in C und C++ gebrauchen für Operationen wie `d=c=b=a;` oder `if((c=getchar())!=EOF){...}`
- Soll der selbstgebaute Zuweisungsoperator ebenso selbst einen Wert zurückliefern (man kann darauf verzichten), so muss er den Returntyp haben:

```
  <klasse> &<klasse>::operator=(const <klasse> &<objekt>);
```

- Realisierung `mitglied2.h`:

```
// mitglied2.h

#ifndef _MITGLIED_H_
#define _MITGLIED_H_

class mitglied
{
private:
    char *name;
    char *motto;
public:
    mitglied(char *nname="", char *nmotto="");
    ~mitglied(void);
    mitglied(const mitglied &mg);
    void operator=(const mitglied &mg);
    void setName(char *nname);
    char *getName(void);
    void setMotto(char *nmotto);
    char *getMotto(void);
    void printit(void);
};
#endif
```

- Realisierung `mitglied2.cpp`:

```
1  // mitglied2.cpp
2  #include <iostream.h>
3  #include <string.h>
4  #include "mitglied2.h"
5  //-----------------------------------------------------
6  mitglied::mitglied(char *nname="", char *nmotto="")
7  {
8      cout << "mitglied::ctor(";
9      if(!nname) nname="";
10     if(!nmotto) nmotto="";
11     cout << nname << ")" << endl;
12
13     name = new char[strlen(nname)+1];
14     strcpy(name, nname);
15
16     motto = new char[strlen(nmotto)+1];
17     strcpy(motto, nmotto);
18     cout << "...ok!" << endl;
19 }
20 //-----------------------------------------------------
21 mitglied::~mitglied(void)
22 {
23     cout << "mitglied::dtor("<< name <<")" <<endl;
24     delete[] motto;
25     delete[] name;
26     cout << "...ok!" << endl;
27 }
28 //-----------------------------------------------------
29 mitglied::mitglied(const mitglied &mg)
30 {
31     cout << "mitglied::cpctor(" << mg.name << ")" << endl;
32     name = new char[strlen(mg.name)+1];
33     strcpy(name, mg.name);
34     motto = new char[strlen(mg.motto)+1];
35     strcpy(motto, mg.motto);
36     cout << "...ok!" << endl;
37 }
38 //-----------------------------------------------------
39 void mitglied::operator=(const mitglied &mg)
40 {
41     cout << "mitglied::op=(" << mg.name << ")" << endl;
42     if(name) delete[] name;
43     if(motto) delete[] motto;
44
45     name = new char[strlen(mg.name)+1];
46     strcpy(name, mg.name);
47     motto = new char[strlen(mg.motto)+1];
48     strcpy(motto, mg.motto);
49     cout << "...ok!" << endl;
50     return;
51 }
52 //-----------------------------------------------------
```

```
53 void mitglied::setName(char *nname)
54 {
55     if(!nname) nname="";
56     if(name) delete[] name;
57     name = new char[strlen(nname)+1];
58     strcpy(name, nname);
59 }
60 //-------------------------------------------------------
61 char *mitglied::getName(void){ return name;}
62 //-------------------------------------------------------
63 void mitglied::setMotto(char *nmotto)
64 {
65     if(!nmotto) nmotto="";
66     if(motto) delete[] motto;
67     motto = new char[strlen(nmotto)+1];
68     strcpy(motto, nmotto);
69 }
70 //-------------------------------------------------------
71 char *mitglied::getMotto(void){ return motto;}
72 //-------------------------------------------------------
73 void mitglied::printit(void)
74 {
75     cout << "Ich heisse....:" << name  << endl;
76     cout << "Mein Motto ist:" << motto << endl;
77 }
78 //-------------------------------------------------------
```

7.3.4 Die letzte Falle: Selbstzuweisung

- Beispiel: (`main8.cpp`)

```
1  // main8.cpp
2  #include <iostream.h>
3  #include "mitglied3.h"
4  int main(int argc, char *argv[])
5  {
6      mitglied a("heinz", "Laber-Rabaaba");
7      mitglied b("fritz", "Langeweile kennt keine Grenzen");
8      mitglied c("willi", "Keine Ahnung-kein Problem");
9      mitglied d("egon",  "Ohne Preis kein Fleiss");
10
11     a=a;
12
13     a.printit();
14     b.printit();
15     c.printit();
16     d.printit();
17     return 0;
18 }
```

- Das Problem liegt hier in der Zeile:

```
1      a=a;
```

7.3.4.1 Was ist passiert?

- Zuerst wurde delete[] auf Objekt (ist hier gleichzeitig Operand a bzw. mg) angewandt, dann wurde mg in den neuen Speicherbereich kopiert.

7.3.4.2 Wie kann man das Problem lösen?

- Vermeidung der Selbstzuweisung (`mitglied4.cpp`) im Zuweisungsoperator selbst:

```
1  // mitglied4.cpp
2  #include <iostream.h>
3  #include <string.h>
4  #include "mitglied3.h"
5  //-----------------------------------------------------
6  mitglied::mitglied(char *nname="", char *nmotto="")
7  {
8      cout << "mitglied::ctor(";
9      if(!nname) nname="";
10     if(!nmotto) nmotto="";
11     cout << nname << ")" << endl;
12
13     name = new char[strlen(nname)+1];
14     strcpy(name, nname);
15
16     motto = new char[strlen(nmotto)+1];
17     strcpy(motto, nmotto);
18     cout << "...ok!" << endl;
19 }
20 //-----------------------------------------------------
21 mitglied::~mitglied(void)
22 {
23     cout << "mitglied::dtor("<< name <<")" <<endl;
24     delete[] motto;
25     delete[] name;
26     cout << "...ok!" << endl;
27 }
28 //-----------------------------------------------------
29 mitglied::mitglied(const mitglied &mg)
30 {
31     cout << "mitglied::cpctor(" << mg.name << ")" << endl;
32     name = new char[strlen(mg.name)+1];
33     strcpy(name, mg.name);
34     motto = new char[strlen(mg.motto)+1];
35     strcpy(motto, mg.motto);
36     cout << "...ok!" << endl;
37 }
38 //-----------------------------------------------------
39 const mitglied &mitglied::operator=(const mitglied &mg)
40 {
41     cout << "mitglied::op=(" << mg.name << ")" << endl;
42     if(this == &mg)
43     {
44         cout << "...Selbstzuweisung...ok!" << endl;
45         return *this;
46     }
```

```
47
48      if(name) delete[] name;
49      if(motto) delete[] motto;
50      name = new char[strlen(mg.name)+1];
51      strcpy(name, mg.name);
52      motto = new char[strlen(mg.motto)+1];
53      strcpy(motto, mg.motto);
54
55      cout << "...ok!" << endl;
56      return *this;
57  }
58  //-------------------------------------------------------
59  void mitglied::setName(char *nname)
60  {
61      if(!nname) nname="";
62      if(name) delete[] name;
63      name = new char[strlen(nname)+1];
64      strcpy(name, nname);
65  }
66  //-------------------------------------------------------
67  char *mitglied::getName(void){ return name;}
68  //-------------------------------------------------------
69  void mitglied::setMotto(char *nmotto)
70  {
71      if(!nmotto) nmotto="";
72      if(motto) delete[] motto;
73      motto = new char[strlen(nmotto)+1];
74      strcpy(motto, nmotto);
75  }
76  //-------------------------------------------------------
77  char *mitglied::getMotto(void){ return motto;}
78  //-------------------------------------------------------
79  void mitglied::printit(void)
80  {
81      cout << "Ich heisse....:" << name  << endl;
82      cout << "Mein Motto ist:" << motto << endl;
83  }
84  //-------------------------------------------------------
```