7.1 Dynamischer Speicher in Objekten/Konstruktor und Destruktor

7.1.1 Situation

In einer Vereinsmitglieder-Verwaltung soll die Klasse `mitglied` erstellt werden. Gegeben ist die C++-Beschreibung der Komponenten; erstellt werden sollen die Methoden.

```
#ifndef _MITGLIED_H_
#define _MITGLIED_H_
class mitglied
{
private:
    char *name;
    char *motto;
public:
    mitglied(void);
    mitglied(const char *nname, const char *nmotto);
    ~mitglied(void);
    void setName(const char *nname);
    const char *getName(void);
    void setMotto(const char *nmotto);
    const char *getMotto(void);
    void printit(void);
};
#endif
```

7.1.2 Dynamischer Speicher im Objekt

Was ist hier anders als bei den bisherigen Klassen? Hier ist neu, dass zwei Attribute dieser Klasse dynamischen Speicher benutzen sollen. Indiz dafür ist der Datentyp *Zeiger* bei diesen Attributen.

7.1.3 Konstruktor und Destruktor

Es ist sinnvoll, den dynamischen Speicher falls nötig bereits im Konstruktor zu allozieren (zu holen). Im Destruktor muss er dann wieder freigegeben werden. Daher braucht man für solche Klassen einen Destruktor.

Zur Verwaltung des dynamischen Speichers kann man in C und C++ die Funktionen `malloc()` und `free()` benutzen, hier ein Beispiel:

```
    int *px;
    px = malloc(40*sizeof(int));
    if(px==NULL) { exit(1); }
    ...
    free(px);
```

Speziell für C++ wurden die neuen Operatoren `new` und `delete` geschaffen; sie ermitteln selbständig die benötigte Speichermenge (und rufen sogar den Konstruktor bzw. Destruktor für die Klasse der angelegten **Attribute** auf). Im ersten Beispiel soll Platz für *ein* Objekt geholt werden:

```
    int *px;
    px = new int; // oder mit Init.: px = new int(3);
    ...
    delete px;
```

Im zweiten Beispiel soll Platz für mehrere Objekte geholt werden:

```
    int *px;
    px = new int[40];
    ...
    delete[] px; // Achtung!!!
```

Die fertig implementierten Methoden zeigt dann das folgende Listing:

```
#include <iostream>
#include <string.h>
using namespace std;
#include "mitglied.h"
//-----------------------------------------------------
mitglied::mitglied(void)
{
    cout << "mitglied::std-ctor" << endl;
    name = new char[1];
    name[0]='\0';
    motto = new char[1];
    motto[0]='\0';
}
//-----------------------------------------------------
mitglied::mitglied(const char *nname, const char *nmotto)
{
    cout << "mitglied::ctor" << endl;
    if(!nname) nname="";
    if(!nmotto) nmotto="";

    name = new char[strlen(nname)+1];
    strcpy(name, nname);
    motto = new char[strlen(nmotto)+1];
    strcpy(motto, nmotto);
}
//-----------------------------------------------------
mitglied::~mitglied(void)
{
    cout << "mitglied::dtor" << endl;
    delete[] motto;
    delete[] name;
}
//-----------------------------------------------------
void mitglied::setName(const char *nname)
{
    if(!nname) nname="";
    if(name) delete[] name;
    name = new char[strlen(nname)+1];
    strcpy(name, nname);
}
//-----------------------------------------------------
const char *mitglied::getName(void){ return name;}
//-----------------------------------------------------
void mitglied::setMotto(const char *nmotto)
{
    if(!nmotto) nmotto="";
    if(motto) delete[] motto;
```

```
48      motto = new char[strlen(nmotto)+1];
49      strcpy(motto, nmotto);
50  }
51  //-------------------------------------------------------
52  const char *mitglied::getMotto(void){ return motto;}
53  //-------------------------------------------------------
54  void mitglied::printit(void)
55  {
56      cout << "Ich heisse " << name
57           << " und mein Motto ist: " << motto << endl;
58  }
```

Getestet werden kann das Programm mit einem kurzen Hauptprogramm:

```
1   #include <iostream>
2   using namespace std;
3   #include "mitglied.h"
4
5   int main(void)
6   {
7       mitglied a("Heinz", "Laber - Rhabarber");
8       mitglied b;
9       mitglied c("Willi", "Keine Ahnung - kein Problem");
10      mitglied d("Egon",  "Ohne Preis kein Fleiss");
11      a.printit();
12      b.printit();
13      c.printit();
14      d.printit();
15      return 0;
16  }
```

7.1.4 Zusammenfassung

Klassen mit dynamischen Komponenten sind etwas aufwendiger (man braucht einen Destruktor), durch das automatische Aufrufen von Konstruktor und Destruktor aber bis hier gut beherrschbar.