4.5 Von C nach C++/new und delete

4.5.1 Idee

In C wird die Speicherverwaltung mit den Betriebssystemaufrufen `malloc()` und `free()` sowie `realloc()` vorgenommen. Für die Berechnung der Speichergrößen sorgten der Programmierer selbst sowie der `sizeof()`-Operator von C.

Die Speicherverwaltung kann leistungsfähiger sein, wenn sie quasi in die Programmiersprache, in diesem Fall in das Laufzeitsystem integriert ist. Dafür werden in C++ die neuen Operatoren `new` und `delete` vorgesehen. `new` und `delete` sind weiterhin mit der C++-eigenen Fehlerbehandlung verbunden.

4.5.2 new statt malloc()

Es soll Speicherplatz für ein (!) `int` geholt werden:

```
1 /* C */
2   int* pn;
3   pn = malloc(1*sizeof(int));
4   if (!pn)
5   {
6     printf("malloc()-Problem!\n"); exit(1);
7   }
```

Und jetzt des Gleiche mit `new`:

```
1 // C++
2   int* pn;
3   pn = new int;
```

Hier entfällt also der `sizeof()`-Operator. Seine Funktionalität ist bereits im `new`-Operator enthalten.

Es gibt eine weitere Möglichkeit von `new`:

```
1   pn = new int(2002);
```

`new` ruft, wenn es für ein Objekt aufgerufen wird, (anders als die Betriebssystem-Funktion `malloc()`) den so genannten *Konstruktor* des Objekts auf. Damit kann man mit `new` eine Variable bereits initialisieren.

4.5.3 delete statt free()

Nun soll der Speicherplatz wieder freigegeben werden.

```
1 /* C */
2   free(pn);
```

Und jetzt das Gleiche mit `delete`:

```
1 // C++
2   delete pn;
```

`delete` ruft, wenn es für ein Objekt aufgerufen wird, (anders als die Betriebssystem-Funktion `free()`) den so genannten *Destruktor* des Objekts auf.

4.5.4 new für Arrays

In C muss man bei Arrays für `malloc()` einfach mehr Speicher anfordern:

```
1 /* C */
2   int* pn;
3   pn = malloc(100*sizeof(int));
4   if(pn==NULL)
5   {
6       ...
7   }
```

In C++ kann man stattdessen quasi den Datentyp angeben:

```
1 // C++
2   int* pn;
3   pn = new int[100];
```

`new` erlaubt mehrdimensionale Arrays und berechnet die Größe selbst. Allerdings müssen hier alle Dimensionen außer der äußersten konstant sein.

```
1   int x=98; // variabel, äußerste Dimension
2   ...
3   int (*px)[10][8][12] = new int[x][10][8][12];
```

4.5.5 delete für Arrays

In C ist nichts zu beachten.

```
1 /* C */
2   free(pn);
```

In C++ dagegen muss man angeben, dass `pn` ein Array war¹:

```
1 // C++
2   delete []pn;
```

¹Warum ist das Klammerpaar in der Sprache unbedingt erforderlich? Da die C-basierten C++-Präcompiler aus `delete` intern einen `free()`-Aufruf machten, liegt nahe, dass hier ein `sizeof()`-Aufruf eingefügt wird. Der Grund könnte aber auch in einer unterschiedlichen internen Organisation von `new int` und `new int[100]` liegen.