Speicher

Aus Sicht von C++ zerfällt der Hauptspeicher in vier Teile:

- Code: Hier steht das Programm.
- Static: Hier stehen die globalen Variablen (von deren Benutzung wir abraten).
- Stack (Stapel): Enthält lokale Variablen, Funktionsparameter, Rücksprungadressen für Funktionen und statische Arrays. Dieser wird automatisch freigegeben, wenn er nicht mehr gebraucht wird.
- Heap (Haufen): Enthält den dynamischen Speicher. Dieser wird *nicht* automatisch freigegeben (keine Garbage Collection).

Wir werden hier aber Datenstrukturen zeigen, die Heap-Speicher liefern, ohne dass Sie sich um Speicherfreigabe kümmern müssen.

Mittel der Wahl zur Speicherbeschaffung: `vector`

Die Klasse `vector` aus der Standardbibliothek liefert einen komfortablen Weg dynamischen Speicher zu verwalten.

```
1 #include <vector>
2 ....
3 int n = 0;
4 std::cin >> n;
5 std::vector<int> zahlen(n); //Speicher fuer n integers
6 // nun kann man mit zahlen[i] (fuer 0 <= i < n) auf
7 // Eintrage von zahlen zugegriffen werden
8 // Speicher muss nicht selbst wieder freigegeben werden.
```

Zahl der Einträge kann mit `size` abgefragt werden:

```
zahlen.size();
```

Damit kann man auf einfache Art über Vektor-Einträge laufen:

```
for(unsigned int i = 0; i < zahlen.size(); i++){
    zahlen[i] = 42;
}
```

Danach sind alle Einträge auf dem Wert `42`.

Vektoren

Vektoren sind erweiterbar mit `push_back`:

```
int k = 37;
zahlen.push_back(k);
```

Mögliche Initialisierungen:

- Man kann einen Vektor mit 0 Elementen anlegen:
  `std::vector<int> mein_vektor(0);`
- Denselben Effekt erreicht man mit:
  `std::vector<int> mein_vektor;`
- Anschließend kann man mit `mein_vektor.push_back(...)` Elemente einfügen.
- Man kann auch gleich alle Elemente eines Vektors initialisieren (hier drei Einträge, die all den Wert `2.7` haben:
  `std::vector<double> eintraege(3, 2.7);`

Bemerkung: Ein Vektor kann nicht als ganzes durch `std:cout` ausgegeben werden (die einzelnen Einträge aber vielleicht schon).

Schleifen über Vektoren

```
1  #include <iostream>
2  #include <vector>
3  int main()
4  {
5     /*Legt einen double-Vektor der Laenge 3 an, dessen Eintraege
6       alle mit 2.7 initialisiert sind: */
7     std::vector<double> eintraege(3, 2.7);
8
9     for(unsigned int i = 0; i < eintraege.size(); i++){
10       std::cout << eintraege[i] << std::endl;
11    }
12    return 0;
13 }
```

Zeilen 9-11 können ersetzt werden durch:

```
for(double entry : eintraege) {
    std::cout « entry « std::endl;
}
```

oder durch:

```
for(auto entry : eintraege) {
    std::cout « entry « std::endl;
}
```

Vektoren und Funktionen

Weiteres zu `std::vector`:

- `std::vector` kann auch Rückgabetyp einer Funktion sein.
- Vektoren können einer Funktion übergeben werden.
- Wenn Vektoren nicht per Referenz übergeben werden, werden sie kopiert (eventuell sehr aufwendig)
- Meist Mittel der Wahl: Übergabe per (`const`) Referenz