Informatik Übungsstunde - Woche 11

Eric Ceglie

Pointer

Ein *Pointer* oder auch *Zeiger* ist ein C++ Objekt, das auf eine Speicheradresse zeigt. Die Adresse eines Objekts `a` können wir immer mit `&a` ausgeben (nicht verwechseln mit Referenz).

Initialisieren

Ein Pointer wird im Allgemeinen mit

```
Typ* name;
```

initialisiert, wobei der Typ dem Typen der Adresse entsprechen muss.

Dereferenzieren

Ein Pointer `p` kann mit dem Operator `*` *dereferenziert* werden. Dies sieht zum Beispiel wie folgt aus:

```
std::cout << *p;
```

Nullpointer

Mit `Typ* p = nullptr` lässt sich ein Pointer erstellen, der "ins Nichts" zeigt, das heisst er zeigt auf keine Speicheradresse.

Der Befehl "new"

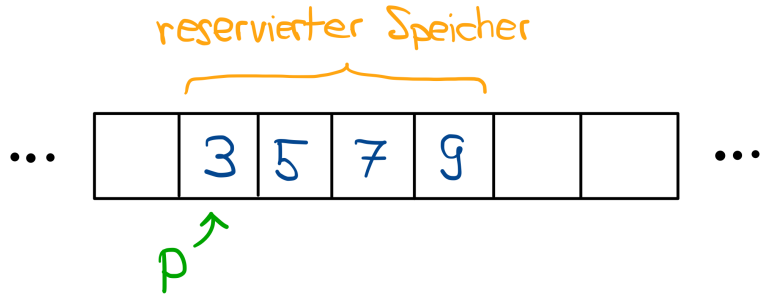
Mit `new` können wir einen Speicher reservieren. Dies tun wir zum Beispiel mit

```
new Typ[n];
```

um n Speicherplätze für Objekte eines gegebenen Typen zu reservieren. Der Befehl `new` gibt dabei direkt einen Pointer auf die erste reservierte Speicherstelle zurück. Das Beispiel

```
int* p = new int[4]{3, 5, 7, 9};
```

lässt sich wie folgt visualisieren:



Pointerarithmetik

Wir benutzen weiterhin den Pointer `p` aus obigem Beispiel. Wir können nun mit Pointern rechnen, um uns sehr einfach im Speicher zu bewegen. Zum Beispiel liefert das folgende Programm

```
int* p = new int[4]{3, 5, 7, 9};
p += 2;
std::cout << *p;
```

den Output `7`.

Wir können Pointer subtrahieren. Zum Beispiel liefert das Programm

```
  std::vector<int> v = {3,5,7,9,11};
  int* p = &v.at(0); // dereferenziert
  int* q = &v.at(3); // dereferenziert
  std::cout << q-p;
```

den Output `3`.

Merke dir folgende Präzedenzregeln:

```
int* p = new int[4]{3, 5, 7, 9};
*p++ = 21; // <=> *p = 21; p++;
*++p = 21; // <=> ++p; *p = 21;
```

Constness

Wie bei den üblichen Datentypen, gibt es auch die Möglichkeit konstante Pointer zu definieren. Dies geschieht mit `Typ* const p;`.

Im Allgemeinen gilt dabei die Faustregel: Bezieht sich das `const` auf das Objekt, so steht es *vor* dem `*`, bezieht es sich auf den Pointer, so steht es *nach* dem `*`.

Dies ergibt also folgende Regeln:

```
int* p;             // => dynamischer Pointer auf dynamischen Wert
const int* p;       // => dynamischer Pointer pointer auf konstanten Wert
int const* p;       // => dynamischer Pointer auf konstanten Wert
int* const p;       // => konstanter Pointer auf dynamischen Wert
const int* const p; // => konstanter Pointer auf konstanten Wert
```

Die Symbole `&` und `*` in C++

In C++ hat das Symbol `&` drei verschiedene Bedeutungen, abhängig davon, wo es steht:

1. Der logische *UND-Operator*.
   Beispiel: `bool valid = a && b;`
2. *Deklaration* einer Variable als *Referenz*.
   Beispiel: `int& x = a;`
3. Zugriff auf *Adresse* einer Variablen.
   Beispiel: `int* p = &a;`

Das Symbol `*` hat ebenso drei verschiedene Bedeutungen, abhängig davon, wo es steht:

1. *Arithmetischer Multiplikationsoperator*.
   Beispiel: `int x = 3 * a;`
2. *Deklaration* einer *Pointer* Variable.
   Beispiel: `int* p = &a;`
3. *Dereferenzierungsoperator* eines Pointers, um auf die Speicherstelle zuzugreifen.
   Beispiel: `int a = *p;`