Wiederholung: Zeiger in C

In C (und C++):

```
int n;       // n enthält ganze Zahl.
int* pn;     // pn enthält Zeiger auf ganze Zahl.
int** ppn;   // ppn enthält Zeiger auf Zeiger auf
             // ganze Zahl.
```

(gilt so für *alle* Datentypen).

Referenzierungs- und Dereferenzierungsoperatoren:

`*px`	liefert das, worauf die Zeigervariable `px` zeigt.
`&x`	liefert die Adresse der Variablen `x`.
`px->abc`	Abkürzung für `(*px).abc`

Wiederholung: Zeiger in C

```
int a = 2;
int b = 5;

int* pa = &a;
int* pb = &b;

// Der Wert von pa ist jetzt ein Zeiger auf a,
// der Wert von pb ist jetzt ein Zeiger auf b.
// Folgende Zuweisungen sind darum gleichwertig:

a = b;

a = *pb;

*pa = b;

*pa = *pb;
```


Wiederholung: Zeiger in C

```
void swap(int* px, int* py) {
    int hilf;
    hilf = *px;
    *px = *py;
    *py = hilf;
}

void main() {
    int a = 1;
    int b = 2;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
}
```


Was ist in C++ anders als in Java?

Call by Reference:

In Java werden bei Funktionsaufrufen *Werte* übergeben.

```
void f(int x) { ... }

int a = 5;
f(a);    // die Variable x in f erhält den Wert 5.
```

Was ist in C++ anders als in Java?

In C werden ebenfalls Werte übergeben.
Diese Werte können auch Zeiger auf Variable sein:

```
void f(int* px) { ... }

int a = 5;
f(&a);   // die Variable px in f erhält als Wert
         // einen Zeiger auf a.
```


Was ist in C++ anders als in Java?

In C++ können außerdem auch *die Variablen selbst* übergeben werden:

```
void f(int& x) { ... }

int a = 5;
f(a);   // die Variable x in f ist ein anderer
        // Name für die Variable a.
```

Man bezeichnet diesen Mechanismus auch als *Call by Reference*
im Unterschied zum üblichen *Call by Value*
(ähnlich in Pascal: *var*-Parameter).

Was ist in C++ anders als in Java?

```
void swap(int* px, int* py) {
    int hilf;
    hilf = *px;
    *px = *py;
    *py = hilf;
}
void main() {
    int a = 1;
    int b = 2;
    swap(&a, &b);
}
```

```
void swap(int& x, int& y) {
    int hilf;
    hilf = x;
    x = y;
    y = hilf;
}
void main() {
    int a = 1;
    int b = 2;
    swap(a, b);
}
```


Was ist in C++ anders als in Java?

Klassendeklarationen:

```
class Unter : public Ober {
    // Unter ist Unterklasse von Ober; die Zugriffsrechte
    // der geerbten Daten werden nicht weiter eingeschränkt.

    // private Variable und/oder Methoden:
    const int MAX = 100;
    int n;
    Unter* next;

public:
    // öffentliche Variable und/oder Methoden:
    bool test(int);
};
```

Was ist in C++ anders als in Java?

Klassenschnittstelle und Implementierung stehen gewöhnlich in verschiedenen Dateien:

```
bool Unter::test(int i) {
  return next->n == i;
}
```


Was ist in C++ anders als in Java?

Packages gibt es in C++ nicht; man kann aber mit dem Schlüsselwort `friend` ausgewählten fremden Funktionen Zugriff auf private Daten geben.

Eine oberste Klasse `Object` existiert nicht.
Auch Interfaces gibt es nicht; dafür ist aber mehrfache Vererbung erlaubt.

Wie in C (und anders als in Java) können Funktionen und Variable auch außerhalb von Klassen definiert werden.

Was ist in C++ anders als in Java?

Infix-Operatoren:

Man kann in C++ Funktionen so deklarieren, daß sie in infix-Notation aufgerufen werden können.
Wenn man zum Beispiel eine Klasse `complex` hat, dann kann man definieren

```
complex operator+ (complex, complex);
complex operator- (complex, complex); // binär
complex operator- (complex);          // unär
complex operator* (complex, complex);
...
```

und anwenden:

```
complex a, b, c;
...
a = a + b * c;
```


Was ist in C++ anders als in Java?

Sogar Zuweisungs- und Arrayindex-Operatoren kann man definieren, beispielsweise in einer Klasse `Vector`:

```
char& operator[] (int);
```

`char&` bedeutet, daß hier statt eines Wertes eine Referenz zurückgegeben wird.
Das heißt, der *Methodenaufruf* `v[5]` kann links in einer Zuweisung stehen:

```
v[5] = 'a';
```

Was ist in C++ anders als in Java?

Speicherverwaltung:

In Java wird der Speicherplatz von Objekten, die im Programm nicht mehr benötigt werden, vom Garbage Collector automatisch wieder freigegeben.
In C++ muß man das manchmal von Hand erledigen (`delete`).
Man benötigt außer Konstruktoren

```
Array(int)
```

auch Destruktoren

```
~Array()
```


Was ist in C++ anders als in Java?

```
class FloatArray {
    int size;
    float* elems;
    void check(int)
public:
    FloatArray(int);
    ~FloatArray()
        { delete[] elems; }
    float& operator[] (int index)
        { check(index); return elems[index]; }
    int length()
        { return size; }
    friend ostream& operator<< (ostream&, FloatArray&);
};
```


Was ist in C++ anders als in Java?

```
FloatArray::FloatArray(int newSize) {
    if (newSize > 0) {
        size = newSize;
        elems = new float[newSize];
    } else {
        size = 0;
        elems = NULL;
    }
}

void FloatArray::check(int index) {
    // wirf exception, falls index zu groß
    ...
}
```


Was ist in C++ anders als in Java?

```
ostream& operator<< (ostream& os, FloatArray& a) {
    int i;
    for (i = 0;  i < a.size;  i++) {
        os << a.elems[i] << ' ';
    }
    os << endl;  // end of line
    return os;
}
```

(Die Funktion gehört *nicht* zur Klasse `FloatArray`, aber da sie als `friend` deklariert ist, darf sie dennoch auf die privaten Variablen `size` und `elems` zugreifen.)

Was ist in C++ anders als in Java?

Templates:

> In der Praxis kommt es häufig vor, daß man einen komplexen Datentyp (z. B. Vektoren oder Bäume) für verschiedene Basistypen benötigt (also `int`-Vektoren, `char`-Vektoren, usw.). Mittels „Templates“ (Schablonen) kann man hier Schreibarbeit einsparen.

Was ist in C++ anders als in Java?

```
template<class T> class Array {
    int size;
    T* elems;
    void check(int)
public:
    Array(int);
    ~Array()
        { delete[] elems; }
    T& operator[] (int index)
        { check(index); return elems[index]; }
    int length()
        { return size; }
};
```


Was ist in C++ anders als in Java?

```
template<class T> Array<T>::Array(int newSize) {
    if (newSize > 0) {
        size = newSize;
        elems = new T[newSize];
    } else {
        size = 0;
        elems = NULL;
    }
}

template<class T> void Array<T>::check(int index) {
    // wirf exception, falls index zu groß
    ...
}
```


Was ist in C++ anders als in Java?

Wenn Objekte dieser Familie von Klassen so deklariert werden:

```
Array<char> c(1024);
Array<int> point(3);
Array<float> values(1024);
```

dann erzeugt der Compiler aus dem Template neue Klassen

```
Array<char>
Array<int>
Array<float>
```