Datentypen

Array	**"Massenvariable"** eines bestimmten Typs

Wichtige Befehle:

Definition: `int my_arr[5] = {2, 3, 8, -1, 3};`
Zugriff: `my_arr[2] = 8 * my_arr[3];`

(Anstatt `int` gehen natürlich auch andere Typen.)
(Die Definition kann auch ohne Initialisierung erfolgen: `int my_arr[5];`)

Wie bei Vektoren beginnen die Indizes bei 0. Allerdings muss der Programmierer bei Arrays **selber** sicherstellen, dass die Indizes nicht über den Array hinausgehen, weshalb wir von der Verwendung von Arrays abraten und stattdessen Vektoren empfehlen.

Zuweisungen (ausser Initialisierung), Vergleiche, etc. müssen **elementweise** erfolgen. Sie können nicht direkt gemacht werden.

Die Länge des Arrays **muss zum Kompilierzeitpunkt eindeutig bestimmbar sein.** (z.B. Literal oder `const`-Variable, die mittels Literal eingelesen wurde, etc.)

```
int a[10];

// Accessing an array:
for (int i = 0; i < 10; ++i)
    a[i] = i; // a becomes {0 1 2 ... 9}
a[10] = 2; // NOT allowed, index 10 outside
a[-4] = 2; // NOT allowed, index -4 outside

// Copying an array:
int b[10] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
for (int i = 0; i < 10; ++i)
    a[i] = b[i]; // Have to do it element-wise
a = b; // NOT valid: direct array-copying is forbidden
```

Operatoren

`my_array[...]`	**Array-** und **Vektor-Zugriff** (Subskript-Operator)
Präzedenz: 17 und Assoziativität: `links` Nicht vergessen: Indizes beginnen bei `0` und nicht `1`	

```
int a[] = {8, 9, 10, 11};
std::cout << a[0]; // outputs 8
a[3] = 5; // a is 8, 9, 10, 5
```

Dynamische Datentypen

`new ...[]`, `delete[]`	**Ranges mit dynamischer Lebensdauer und Länge** erstellen.

```
int n; std::cin >> n;
int* range = new int[n];
// Read in values to the range
for (int* i = range; i < range + n; ++i) std::cin >> *i;
delete range; // ERROR: must say: delete[]
delete[] range; // This works
```

Zeiger-Arithmetik

Zeiger	**Iterieren**

Wichtige Befehle:

Zeiger:	`int* ptr = new int[6];`	
temporärer Shift:	`ptr + 3`	
	`ptr - 3`	
permanenter Shift:	`++ptr`	
	`ptr++`	
	`--ptr`	
	`ptr--`	
	`ptr += 3`	
	`ptr -= 3`	
Distanz bestimmen:	`ptr1 - ptr2`	
Position vergleichen:	`ptr1 < ptr2`	(Sonst: `<=`, `>`, `>=`, `==`, `!=`)

Achtung: Die grünen Shifts erzeugen einen neuen (temporären) Zeiger und verschieben `ptr` **nicht**. Die violetten Shifts verschieben aber `ptr`.

```
// Read 6 values into an array
std::cout << "Enter 6 numbers:\n";
int* a = new int[6];
int* pTE = a+6;
for (int* i = a; i < pTE; ++i)
    std::cin >> *i; // read into array element

// Output:   a[0]+a[3],  a[1]+a[4],  a[2]+a[5]
for (int* i = a; i < a+3; ++i) {
    assert(i+3 < pTE); // Assert that i+3 stays inside.
    std::cout << (*i + *(i+3)) << ",  ";
}
```