1 Typumwandlung

- **Implicit Casting**: Automatische Typumwandlung durch den Compiler.
  - `int a = 5.4;` $\Longrightarrow$ a wird zu einem int (5)
  - `float b = 7/2;` $\Longrightarrow$ Ganzzahlige Division, Ergebnis 3 wird zu double (3.0)
  - `float c = 7/2.0;` $\Longrightarrow$ Einer der Werte ist float, Ergebnis 3.5
  - `double d = 'A'- 12;` $\Longrightarrow$ char wird zu int (65), dann -12 (53), dann zu double (53.0)
  - `int e = true + 3;` $\Longrightarrow$ bool wird zu int (1) + 3 (4), dann zu int (4)
  - Allgemein: *Der kleinere Typ wird in den größeren umgewandelt*
- **Explicit Casting**: Manuelle Typumwandlung durch den Programmierer.
  - `int x = (int)3.7;` $\Longrightarrow$ Klassischer Cast: Ergebnis ist 3
  - `int y = static_cast<int>(3.7);` $\Longrightarrow$ Moderner Cast mit `static_cast`: Ergebnis ist ebenfalls 3

2 Hierarchie von Operatoren

Priorität	Operator	Beschreibung
Hoch	`! * &`	**Unär:** Log. NICHT, Deref., Adresse
↓	`* /`	**Binär:** Multiplikation, Division
↓	`+ -`	**Binär:** Addition, Subtraktion
↓	`<< >>`	**Binär:** Bit-Shift Links/Rechts
↓	`&`	**Binär:** Bitweises UND
↓	`|`	**Binär:** Bitweises ODER
↓	`&&`	**Binär:** Logisches UND
Niedrig	`||`	**Binär:** Logisches ODER

3 Wertebereiche von Datentypen

Datentyp	Bytes	Wertebereich
`bool`	1	`true` oder `false`
`char`	1	-128 bis 127
`unsigned char`	1	0 bis 255
`short`	2	-32.768 bis 32.767
`unsigned short`	2	0 bis 65.535
`int`	4	-2.147.483.648 bis 2.147.483.647
`unsigned int`	4	0 bis 4.294.967.295
`long long`	8	ca. -9,2 x 10^{18} bis 9,2 x 10^{18}
`float`	4	ca. ±3.4 x 10^{38} (7 Dezimalstellen)
`double`	8	ca. ±1.8 x 10^{308} (15 Dezimalstellen)

4 Overflow von Zahlen

Overflow = Zugewiesene oder berechnete Zahl liegt außerhalb des darstellbaren Bereichs eines Datentyps.

- **Ganzzahlen**: Undefiniertes Verhalten. z.B. zu hohe Bits werden abgeschnitten oder es wird auf den Minimalwert zurückgesetzt.
- **Gleitkommazahlen**: Im IEEE 754 Standard wird bei Overflow der Wert `inf` (unendlich) zugewiesen.

5 Definition und Deklaration

- **Definition**: Reserviert Speicherplatz für eine Variable oder Funktion und kann optional initialisiert werden.
  - Beispiel Variable: `int x = 5;`
- **Deklaration**: Informiert den Compiler über den Typ und Namen einer Variable oder Funktion, reserviert aber keinen Speicherplatz.
  - Beispiel Variable: `int x;`
  - Beispiel Funktion: `void foo();`
- **Prototyp**: Funktionsdeklaration ohne Funktionskörper.
  - Beispiel: `int map(double[], int, int (*)(double));`
- **Wichtig**: *Jede Definition ist auch eine Deklaration!*

6 String API

Methode	Beschreibung
`.length()`	Gibt die Länge des Strings zurück
`.empty()`	Prüft, ob der String leer ist
`.clear()`	Löscht den Inhalt des Strings
`.append(str)`	Fügt den String `str` an das Ende an (+=)
`.at(idx)`	Gibt das Zeichen an der Position `idx` zurück ([index])
`.substr(start, len)`	Gibt einen Teilstring ab `start` mit Länge `len` zurück
`.find(str)`	Sucht nach `str` und gibt die Startposition zurück

7 Vector API

Methode	Beschreibung
`.size()`	Gibt die Anzahl der Elemente im Vektor zurück
`.empty()`	Prüft, ob der Vektor leer ist
`.clear()`	Entfernt alle Elemente aus dem Vektor
`.push_back(val)`	Fügt das Element `val` am Ende des Vektors hinzu
`.pop_back()`	Entfernt das letzte Element des Vektors
`.at(idx)`	Gibt das Element an der Position `idx` zurück
`.front()/ .back()`	Gibt das erste/letzte Element des Vektors zurück
`.begin()/ .end()`	Gibt Iteratoren auf den Anfang/Ende des Vektors zurück

8 Nützliche std:: Funktionen

Methode	Beschreibung
`std::sort(b, e)`	`void` Sortiert einen Bereich
`std::find(b, e, v)`	`Iterator` Sucht einen Wert im Bereich
`std::reverse(b, e)`	`void` Dreht die Reihenfolge im Bereich um
`std::max(a, b)`	`T` Gibt das größere von zwei Werten zurück
`std::find_if(b, e, p)`	`Iterator` Sucht das erste Element, das das Prädikat erfüllt
`std::count_if(b, e, p)`	`int` Zählt Elemente, die das Prädikat erfüllen
`std::all_of(b, e, p)`	`bool` Prüft, ob alle Elemente das Prädikat erfüllen
`std::any_of(b, e, p)`	`bool` Prüft, ob mindestens ein Element das Prädikat erfüllt
`std::transform(b, e, d, f)`	`void` Wendet Funktion `f` auf alle Elemente an und speichert sie in `dest`

- `b` = `begin()`, `e` = `end()`
- `p` = Prädikat (Funktion, die bool zurückgibt) z.B. `[](int x){return x>5;}`
- `v` = Wert, der gesucht wird
- `d` = Zieliterator (z.B. Anfang eines anderen Containers)
- `f` = Funktion, die auf jedes Element angewendet wird (z.B. `[](int x){return x*2;}`)