Datentypen

`bool`	Datentyp für **Wahrheitswerte**
Literal: `true`, `false`	

```
bool t = true;
if (!t == false)
    std::cout << "This is output!\n";
if (t)
    std::cout << "This is output as well!\n";
```

`const ...`	***Schreib*zugriff** auf Variable **verbieten**
Gemeint ist natürlich der Schreibzugriff *nach* der Initialisierung.	

```
int a = 3;
const int b = 4;
a = 5;           // valid
b = 3;           // not valid since b is const
int c = -2 * b;  // valid since just WRITE-access is forbidden
                 //     by "const"
```

Operatoren

`&&`	**Logisches UND**
Präzedenz: 6 und Assoziativität: `links`	
Kurzschluss-Auswertung: `&&` wertet ***immer*** den *linken* Operanden zuerst aus. Ist dieser *falsch* (also `false`), so wird der *rechte* Operand ***nicht mehr*** ausgewertet.	

(...)

(...)

```
if (3 > 2 && 10 > 11)    // no short circuit evaluation
    std::cout << "Of course not!\n";

int a = 3;
if (false && ++a < 2)    // short circuit evaluation
    std::cout << "Of course not!\n";

std::cout << a << "\n";  // Output: 3
```

||	**Logisches ODER**

Präzedenz: 5 und Assoziativität: `links`

Kurzschluss-Auswertung: `||` wertet auch ***immer*** den *linken* Operanden zuerst aus. Ist dieser *wahr* (also `true`), so wird der *rechte* Operand ***nicht mehr*** ausgewertet.

```
if (8 < 3 || -2 > -5)    // no short circuit evaluation
    std::cout << "Yes!\n";

int a = 3;
if (3 < 8 || ++a < 2)    // short circuit evaluation
    std::cout << "Yes!\n";

std::cout << a << "\n";  // Output: 3
```

!	**Logisches NICHT**

Präzedenz: 16 und Assoziativität: `rechts`

```
int a,b,c;
std::cin >> a >> b >> c; // read three int values from user

if ( ! (a <= b && c <= b) )
    std::cout << "b is not max!\n";
```

<	**strikt kleiner**

Präzedenz: 11 und Assoziativität: `links`

Sonst gibt es noch:

`>`	**strikt grösser**
`<=`	**kleiner gleich**
`>=`	**grösser gleich**

```
int a,b;
std::cin >> a >> b; // read two int values from user

if (a < b)
    std::cout << "a smaller than b\n";
```

==	**exakt gleich**

Präzedenz: 10 und Assoziativität: `links`

Sonst gibt es noch:

`!=`	**ungleich**

```
int a,b;
std::cin >> a >> b; // read two int values from user

if (a == b)
    std::cout << "a is equal to b\n";
```

Schleifen

for (...) {...}	`for`-Schleife

Wenn man eine leere Condition als Abbruchbedingung angibt, so wird diese als *wahr* interpretiert.

(...)

(...)

```
unsigned int n;
std::cin >> n;

// Compute 1 + 2 + 3 + ... + n
unsigned int sum = 0;
for (unsigned int i = 1; i <= n; ++i)
    sum += i;
std::cout << "1 + 2 + ... + n = " << sum << "\n";
```

Generell

`if-else`	**bedingtes Ausführen** von Code
Der `else`-Teil ist optional.	

```
int a,b;
std::cin >> a >> b; // read two int values from user

// if a < b then output 3; otherwise output 8
if (a < b)
    std::cout << 3 << "\n";
else
    std::cout << 8 << "\n";
```

`assert`	**sofortiges Stoppen** des Programms zu Testzwecken
Erfordert: `#include <cassert>` Wenn das fertige Programm veröffentlicht werden soll, kann man die `assert`-Befehle bequem deaktivieren (siehe `assertion2.cpp`).	

```
int a,b;
std::cin >> a >> b; // read two int values from user
assert(b != 0); // prevent division by 0
std::cout << a / b << "\n";
```