Datentypen

Vektoren	**"Massenvariable"** eines bestimmten Typs

Erfordert: `#include <vector>`

Wichtige Befehle:

Definition: `std::vector<int> my_vec (length, init_value);`
Zugriff: `my_vec[2] = 8 * my_vec[3];`
neues Element hinten: `my_vec.push_back(5)`

(Anstatt `int` gehen natürlich auch andere Typen.)
(Die Definition kann auch nur mit Längenangabe erfolgen:
`std::vector<int> my_vec (length)`)

Die Indizes beginnen bei `0`.

Der Programmierer muss **selber** sicherstellen, dass die Indizes nicht über den Array hinausgehen.

Zuweisungen (ausser Initialisierung), Vergleiche, etc. müssen **element-weise** erfolgen.

Die Länge des Vektors muss **NICHT** zum Kompilierzeitpunkt eindeutig bestimmbar sein. Ausserdem besitzen Vektoren "Komfortfunktionen" wie beispielsweise `push_back()`.

```
int len; std::cin >> len; // Assume: len > 2

std::vector<int> my_vec (len, 0); // my_vec: 0, 0, 0, ..., 0
my_vec[1] = 3;                    // my_vec: 0, 3, 0, ..., 0
my_vec.push_back(5);              // my_vec: 0, 3, 0, ..., 0, 5
                                  //   [length increases by 1]
```

`char`	Datentyp für **Zeichen**

Literal: `'a'` für Zeichen (*einfache* Anführungszeichen)
Literal: `"Hello World"` für Strings (*doppelte* Anführungszeichen)

Ein String-Literal wird wie ein Vector des Typs `char` mit passender Länge gespeichert. Es wird immer mit `'\0'` terminiert (hat also ein Zeichen mehr als "Buchstaben").

```
auto a = "my text"; // "my text" generates:   my text\0
a[3] = '5'; // Changes a to: "my 5ext"
// output
for (int i = 0; a[i] != '\0'; ++i)
    std::cout << a[i] << "\n";
// Note: '\0' marks the end of the string.
```

`std::string`	**komfortablerer** Datentyp für **Zeichen**

Erfordert: `#include <string>`

Vorteile gegenüber `char`-Arrays:

variable Länge:	`std::string my_str (n, 'a');` (`n` kann variabel sein)
Länge abfragen:	`my_str.length()`
vergleichbar:	`text1 == text2`
hintereinander hängen:	`text1 += text2`
bequemer Output:	`std::cout << my_str;`

```
std::string my_word (5, 'a'); // initialize my_word as aaaaa
std::string ref (5, 'z');
my_word += ref; // append ref to my_word.
                // Afterwards my_word:   aaaaazzzzz
                // Afterwards ref:       zzzzz
for (int i = 0; i < my_word.length(); ++i)
    std::cin >> my_word[i]; // read user input into our word
if (my_word == ref) // impossible since lengths differ (5 VS 10)
    std::cout << "never output\n";
std::cout << my_word << "\n"; // output whole string at once
```

Operatoren

`my_array[...]`	**Vektor-Zugriff** (Subskript-Operator)
Präzedenz: 17 und Assoziativität: `links`	

```
std::vector<int> a (2, 0);
a[1] = 5; // a is 0, 5
```