

Datentypen

Vektoren (mehrdim.)	mehrdimensionale "Massenvariable" eines bestimmten Typs
<p>Erfordert: <code>#include<vector></code></p> <p>Wichtige Befehle:</p> <p>Definition: <code>std::vector<std::vector<int> ></code> <code>my_vec (n_rows, std::vector<int>(n_cols, init_value))</code></p> <p>Zugriff: <code>my_arr[1][1] = 8 * my_arr[0][2];</code> (Anstatt int gehen natürlich auch andere Typen.)</p>	
<pre>std::vector<std::vector<int> > my_vec (2, std::vector<int>(4, 0)); my_vec[1][2] = 3; // my_vec becomes // 0, 0, 0, 0 // 0, 0, 3, 0</pre>	

<code>std::string</code>	komfortablerer Datentyp für Zeichen
<p>Erfordert: <code>#include<string></code></p> <p>Vorteile:</p> <p>variable Länge: <code>std::string my_str (n, 'a');</code> (n kann variabel sein)</p> <p>Länge abfragen: <code>my_str.length()</code></p> <p>vergleichbar: <code>text1 == text2</code></p> <p>hintereinander hängen: <code>text1 += text2</code></p> <p>bequemer Output: <code>std::cout << my_str;</code></p>	

(...)

Programmier-Befehle - Woche 8

(...)

```
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
std::cout << my_word.length() << "\n"; // output: 10
my_word[3] = 'b'; // change my_word to aaabzzzzz
if (my_word == ref) // false
    std::cout << "not output\n";
std::cout << my_word << "\n"; // output whole string at once
```

Input/Output

<code>std::noskipws</code>	Whitespaces einlesen
Erfordert: <code>#include<ios></code> oder <code>#include <iostream></code>	
<pre>char c; // Version 1: Assume the user enters: // a b std::cin >> c; // read 'a' std::cin >> c; // read 'b' // Version 2: Assume the user enters again: // a b std::cin >> std::noskipws; std::cin >> c; // read 'a' std::cin >> c; // read ' ' std::cin >> c; // read 'b'</pre>	

leerer Eingabestrom	Prüfe, ob mehr Eingaben vorhanden sind.
---------------------	--

(...)

(...)



Dahinter steckt eine Konvertierung von `std::cin` zu `bool`:

`true:` weitere Eingaben vorhanden
`false:` keine Eingaben mehr vorhanden

Wir brauchen diese Abfrage meistens, um eine Schleife solange laufen zu lassen, wie weitere Eingaben vorhanden sind. (siehe Beispiel unten)

```
char input;  
int length_of_text = 0;  
while(std::cin >> input)  
    ++length_of_text;  
std::cout << length_of_text;
```

Turtle

Turtle Plots	Zeichnen von Geraden																
<p>Erfordert: <code>#include "turtle.h"</code></p> <p>Die Turtle kennt 8 Befehle:</p> <table><tr><td><code>turtle::forward():</code></td><td>gezeichneter Schritt vorwärts</td></tr><tr><td><code>turtle::jump():</code></td><td>nicht gezeichneter Schritt vorwärts</td></tr><tr><td><code>turtle::left(my_angle):</code></td><td>Drehung nach links</td></tr><tr><td><code>turtle::right(my_angle):</code></td><td>Drehung nach rechts</td></tr><tr><td><code>turtle::save():</code></td><td>Position <i>und Blickrichtung</i> merken</td></tr><tr><td><code>turtle::restore():</code></td><td>Position <i>und Blickrichtung</i> laden</td></tr><tr><td><code>turtle::colorcycle():</code></td><td>Farbe wechseln</td></tr><tr><td><code>turtle::colorcycle2(d):</code></td><td>Farbe wechseln (eigene Abstufung d)</td></tr></table> <p>Die Turtle kann mehrere Positionen speichern (mittels <code>turtle::save()</code>). <code>turtle::restore()</code> lädt dann die Neueste und entfernt diese Position aus der Merkliste (somit ist dann die vorher zweitneuste Position neu die neuste).</p>		<code>turtle::forward():</code>	gezeichneter Schritt vorwärts	<code>turtle::jump():</code>	nicht gezeichneter Schritt vorwärts	<code>turtle::left(my_angle):</code>	Drehung nach links	<code>turtle::right(my_angle):</code>	Drehung nach rechts	<code>turtle::save():</code>	Position <i>und Blickrichtung</i> merken	<code>turtle::restore():</code>	Position <i>und Blickrichtung</i> laden	<code>turtle::colorcycle():</code>	Farbe wechseln	<code>turtle::colorcycle2(d):</code>	Farbe wechseln (eigene Abstufung d)
<code>turtle::forward():</code>	gezeichneter Schritt vorwärts																
<code>turtle::jump():</code>	nicht gezeichneter Schritt vorwärts																
<code>turtle::left(my_angle):</code>	Drehung nach links																
<code>turtle::right(my_angle):</code>	Drehung nach rechts																
<code>turtle::save():</code>	Position <i>und Blickrichtung</i> merken																
<code>turtle::restore():</code>	Position <i>und Blickrichtung</i> laden																
<code>turtle::colorcycle():</code>	Farbe wechseln																
<code>turtle::colorcycle2(d):</code>	Farbe wechseln (eigene Abstufung d)																
<pre>// Draw a triangle (see below) turtle::forward(); turtle::left(120); turtle::forward(); turtle::left(120); turtle::forward(); // Move to neutral position turtle::left(120); // horizontal viewing direction turtle::jump(10); // move away from triangle (without drawing) // Draw a letter T (see below) turtle::forward(); turtle::save(); // memorize middle of letter T turtle::forward(); turtle::restore(); // go back to middle of letter T turtle::right(90); turtle::forward(2); // The argument means: 2 steps forward</pre> <hr/> <div style="display: flex; justify-content: space-around; align-items: center;"></div>																	