

## Datentypen

| <code>const ...</code>  | Schreibzugriff auf Variable verbieten |
|---|---------------------------------------|
| <p>Gemeint ist natürlich der Schreibzugriff <i>nach</i> der Initialisierung.</p> <p><code>const</code> gibt es auch für Referenzen, siehe unten.</p>  |                                       |
| <pre>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 to b is                 // forbidden by "const"</pre> |                                       |

| Referenzen  | Alias für bestehende Variable |
|---|-------------------------------|
| <p>Referenzen können <b>nur Variablen</b> ihres zugrundeliegenden Typs referenzieren. Sonst gibt es einen Fehler.</p> <p>Ausserdem können Referenzen <b>nur mit L-Werten initialisiert werden</b> (also Werten mit einer Adresse im Speicher).</p> <p>Funktionen, bei denen die Argumente Referenztyp haben, können ihre Aufrufargumente ändern. <b>Das ist eine sehr mächtige Anwendung von Referenzen.</b> Siehe beispielsweise die Funktion <code>swap</code> aus der Vorlesung.</p> |                               |
| <pre>// Usage int a = 3; int&amp; b = a; // reference to a std::cout &lt;&lt; b &lt;&lt; "\n"; // Output: 3 a = 18; std::cout &lt;&lt; b &lt;&lt; "\n"; // Output: 18 b = 25; std::cout &lt;&lt; a &lt;&lt; "\n"; // Output: 25  // Issues int&amp; c = 3; // Error: 3 is not an lvalue (3 has no address) bool d = false; int&amp; e = d; // Error: d is bool, e wants to reference an int</pre>   |                               |

# Programmier-Befehle - Woche 7

| <code>const</code> Referenzen  | <code>const</code> -Alias für bestehende Variable |
|--|---|
| <p>Im Prinzip funktionieren <code>const</code> Referenzen so wie normale Referenzen, bloss dass der <b>Schreibzugriff</b> auf das Ziel der Referenz <i>via diese Referenz verboten ist</i>.</p> <p>Ein weiterer Unterschied ist, dass <code>const</code> Referenzen <b>R-Werte beinhalten können</b>. Dann wird jeweils ein temporärer Speicher für den R-Wert erstellt, der solange gültig ist, wie die <code>const</code> Referenz selbst. Dies erlaubt beispielsweise, eine Funktion bezüglich Call-by-Reference trotzdem mit R-Werten aufzurufen.</p> <p>Zu beachten ist auch, dass man <b>keine nicht-const Referenz mit einer <code>const</code> Referenz initialisieren darf</b>.</p> |   |
| <pre>double a = 3.0; double&amp; b = a; // non-const reference const double&amp; c = a; // const reference  c = 4.0; // Error: write-access forbidden a = 5.0; // this works, a can be changed through itself b = 6.0; // this works, a can be changed through non-const refs  std::cout &lt;&lt; c &lt;&lt; "\n"; // Output: 6.0, read-access is allowed. double&amp; d = c; // Error: non-const ref from const ref not allowed const double&amp; e = 5.0; // this works for const references.</pre>  |   |

| Vektoren   | “Massenvariable” eines bestimmten Typs |
|--|--|
| <p>Erfordert: <code>#include&lt;vector&gt;</code></p> <p>Wichtige Befehle:</p> <p><b>Definition:</b> <code>std::vector&lt;int&gt; my_vec (length, init_value);</code><br/><b>Zugriff:</b> <code>my_vec[2] = 8 * my_vec[3];</code><br/><b>neues Element hinten:</b> <code>my_vec.push_back(5)</code></p> <p>(Anstatt <code>int</code> gehen natürlich auch andere Typen.)</p> |  |
| <pre>int len; std::cin &gt;&gt; len; // Assume here: len &gt; 2  std::vector&lt;int&gt; my_vec (len, 0); // my_vec: 0, 0, 0, ..., 0 my_vec[1] = 3; // my_vec: 0, 3, 0, ..., 0</pre>  |  |

# Programmier-Befehle - Woche 7

|   |                             |
|---|-----------------------------|
| <code>char</code>   | Datentyp für <b>Zeichen</b> |
| <p>Literal: <code>'a'</code> für Zeichen (<i>einfache</i> Anführungszeichen)<br/>Literal: <code>"Hello World"</code> für Strings (<i>doppelte</i> Anführungszeichen)</p> <p><code>chars</code> können sehr einfach zu <code>int</code> <b>hin und her umgewandelt</b> werden. (Der resultierende <code>int</code>-Wert ist auf den meisten Plattformen eine entsprechende Zahl gemäss ASCII-Code, siehe Vorlesungshandout 7, Slide 45.)</p> |                             |
| <pre>char ch = 'd'; int i = ch; // convert char --&gt; int (here: 'd' --&gt; 100) ++ch;      // increase to 101 which is 'e' ++i; std::cout &lt;&lt; (ch == i) &lt;&lt; "\n"; // compare 101 == 101  // Read single character from user: std::cin &gt;&gt; ch;</pre>  |                             |

## Operatoren

|  |                                     |
|--|-------------------------------------|
| <code>my_vec[...]</code>   | Vektor-Zugriff (Subskript-Operator) |
| Präzedenz: 17 und Assoziativität: links  |                                     |
| Nicht vergessen: Indizes <b>beginnen bei 0</b> und nicht 1   |                                     |
| <pre>int a[] = {8, 9, 10, 11}; std::cout &lt;&lt; a[0]; // outputs 8 a[3] = 5; // a is 8, 9, 10, 5</pre> |                                     |