

Datentypen

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

Programmier-Befehle - Woche 7

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

Programmier-Befehle - Woche 7

Vektoren	“Massenvariable” eines bestimmten Typs
<p>Erfordert: <code>#include<vector></code></p> <p>Wichtige Befehle:</p> <p>Definition: <code>std::vector<int> my_vec = std::vector<int>(length, init_value);</code></p> <p>Zugriff: <code>my_vec.at(2) = 8 * my_vec.at(3);</code></p> <p>neues Element hinten: <code>my_vec.push_back(5)</code></p> <p>(Anstatt <code>int</code> gehen natürlich auch andere Typen.)</p> <p>Statt der Syntax</p> <pre>std::vector<int> my_vec = std::vector<int>(...)</pre> <p>zur Deklaration und Initialisierung eines Vektors, kann alternativ auch eine der folgenden verwendet werden:</p> <pre>auto my_vec = std::vector<int>(...) // 1st alternative std::vector<int> my_vec(...) // 2nd alternative</pre> <p>In Alternative 1 kann die Typdeklaration der Variable durch das Schlüsselwort <code>auto</code> ersetzt werden, da der Typ durch die rechte Seite eindeutig bestimmt werden kann. In Alternative 2 ist es sozusagen andersherum: Da der Typ der neuen Variablen bekannt ist, muss er bei der Initialisierung nicht zwingend wiederholt werden. Alle drei Formen sind auch für andere Typen nutzbar, d.h. nicht speziell für Vektoren. Mehr zu Objektinitialisierung (und Konstruktoraufrufen) erfahren Sie im Kapitel zu Klassen.</p>	
<pre>int len; std::cin >> len; // Assume here: len > 2 // my_vec: 0, 0, 0, ..., 0 std::vector<int> my_vec = std::vector<int>(len, 0); my_vec.at(1) = 3; // my_vec: 0, 3, 0, ..., 0</pre>	

Programmier-Befehle - Woche 7

<code>char</code>	Datentyp für Zeichen
<p>Literal: <code>'a'</code> für Zeichen (<i>einfache</i> Anführungszeichen)</p> <p>Literal: <code>"Hello World"</code> für Strings (<i>doppelte</i> Anführungszeichen)</p> <p><code>chars</code> können sehr einfach zu <code>int</code> <i>hin und her umgewandelt werden</i>. (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 --> int (here: 'd' --> 100) ++ch; // increase to 101 which is 'e' ++i; std::cout << (ch == i) << "\n"; // compare 101 == 101 // Read single character from user: std::cin >> ch;</pre>	

Operatoren

<code>my_vec.at(...)</code>	Vektor-Zugriff (at-Methode)
<p>Präzedenz: 17 und Assoziativität: links</p> <p>Nicht vergessen: Indizes <i>beginnen bei 0</i> und nicht 1</p>	
<pre>// Directly Initialise vector with given values std::vector<int> a = {8, 9, 10, 11}; std::cout << a.at(0); // outputs 8 a.at(3) = 5; // a is 8, 9, 10, 5</pre>	