

## Zeiger

Zeiger (generell)	Adresse eines Objekts im Speicher								
<p>Wichtige Befehle:</p> <p><b>Definition:</b> <code>int* ptr = address_of_type_int;</code> (ohne Startwert: <code>int* ptr = nullptr;</code>)</p> <p><b>Zugriff auf Zeiger:</b> <code>ptr = otr_ptr // Pointer gets new target.</code></p> <p><b>Zugriff auf Target:</b> <code>*ptr = 5 // Target gets new value 5.</code></p> <p><b>Adresse auslesen:</b> <code>int* ptr_to_a = &amp;a; // (a is int-variable)</code></p> <p><b>Vergleich:</b> <code>ptr == otr_ptr // Same target?</code> <code>ptr != otr_ptr // Different targets?</code></p> <p>(Anstatt <code>int</code> gehen natürlich auch andere Typen.) (Eine <code>address_of_type_int</code> kann man durch einen anderen Zeiger oder auch mittels dem Adressoperator <code>&amp;</code> erzeugen (siehe Beispiel unten).)</p> <p>Der <b>Wert</b> des Zeigers ist die Speicheradresse des <b>Targets</b>. Will man also das Target via diesen Zeiger verändern, muss man zuerst "zu der Adresse gehen". Genau das macht der Dereferenz-Operator <code>*</code>.</p> <p><b>Beispiel:</b> (Gelte <code>int a = 5;</code>)</p> <table><tr><td>Wert von <code>a</code>:</td><td>5</td></tr><tr><td>Speicheradresse von <code>a</code>:</td><td>0x28fef8</td></tr><tr><td>Wert von <code>a_ptr</code>:</td><td>0x28fef8</td></tr><tr><td>Wert von <code>*a_ptr</code>:</td><td>5</td></tr></table> <p>Ein Zeiger kann immer nur auf den entsprechenden Typ zeigen. (z.B. <code>int* ptr = &amp;a;</code> Hier muss <code>a</code> Typ <code>int</code> haben.)</p>		Wert von <code>a</code> :	5	Speicheradresse von <code>a</code> :	0x28fef8	Wert von <code>a_ptr</code> :	0x28fef8	Wert von <code>*a_ptr</code> :	5
Wert von <code>a</code> :	5								
Speicheradresse von <code>a</code> :	0x28fef8								
Wert von <code>a_ptr</code> :	0x28fef8								
Wert von <code>*a_ptr</code> :	5								
<pre>int a = 5; int* a_ptr = &amp;a; // a_ptr points to a a_ptr = a; // NOT valid (same as: a_ptr = 5; ) //      5 is NOT an address. a_ptr = &amp;a; // valid *a_ptr = 9; // a obtains value 9 std::cout &lt;&lt; "a == " &lt;&lt; a &lt;&lt; "\n"; // Output: a == 9 std::cout &lt;&lt; "a == " &lt;&lt; *a_ptr &lt;&lt; "\n"; // Output: a == 9</pre>									

# Programmier-Befehle - Woche 9

<code>const</code> (Zeiger)	Zeiger Konstantheit
Es gibt zwei Arten von Konstantheit:	
kein Schreibzugriff auf Target:	<code>const int* a_ptr = &amp;a;</code>
kein Schreibzugriff auf Zeiger:	<code>int* const a_ptr = &amp;a;</code>
<pre>int a = 5; int b = 8;  const int* ptr_1 = &amp;a; *ptr_1 = 3; // NOT valid (change target) ptr_1 = &amp;b; // valid (change pointer)  int* const ptr_2 = &amp;a; *ptr_2 = 3; // valid (change target) ptr_2 = &amp;b; // NOT valid (change pointer)  const int* const ptr_3 = &amp;a; *ptr_3 = 3; // NOT valid (change target) ptr_3 = &amp;b; // NOT valid (change pointer)</pre>	

# Programmier-Befehle - Woche 9

Zeiger	Iterieren										
<p>Wichtige Befehle:</p> <table><tr><td><b>Zeiger:</b></td><td><code>int* ptr = new int[6];</code></td></tr><tr><td><b>temporärer Shift:</b></td><td><code>ptr + 3</code> <code>ptr - 3</code></td></tr><tr><td><b>permanenter Shift:</b></td><td><code>++ptr</code> <code>ptr++</code> <code>--ptr</code> <code>ptr--</code> <code>ptr += 3</code> <code>ptr -= 3</code></td></tr><tr><td><b>Distanz bestimmen:</b></td><td><code>ptr1 - ptr2</code></td></tr><tr><td><b>Position vergleichen:</b></td><td><code>ptr1 &lt; ptr2</code> (Sonst: <code>&lt;=</code>, <code>&gt;</code>, <code>&gt;=</code>, <code>==</code>, <code>!=</code>)</td></tr></table> <p><b>Achtung:</b> Die <b>grünen Shifts</b> erzeugen einen neuen (temporären) Zeiger und verschieben <code>ptr</code> <b>nicht</b>. Die <b>violetten Shifts</b> verschieben aber <code>ptr</code>.</p>		<b>Zeiger:</b>	<code>int* ptr = new int[6];</code>	<b>temporärer Shift:</b>	<code>ptr + 3</code> <code>ptr - 3</code>	<b>permanenter Shift:</b>	<code>++ptr</code> <code>ptr++</code> <code>--ptr</code> <code>ptr--</code> <code>ptr += 3</code> <code>ptr -= 3</code>	<b>Distanz bestimmen:</b>	<code>ptr1 - ptr2</code>	<b>Position vergleichen:</b>	<code>ptr1 &lt; ptr2</code> (Sonst: <code>&lt;=</code> , <code>&gt;</code> , <code>&gt;=</code> , <code>==</code> , <code>!=</code> )
<b>Zeiger:</b>	<code>int* ptr = new int[6];</code>										
<b>temporärer Shift:</b>	<code>ptr + 3</code> <code>ptr - 3</code>										
<b>permanenter Shift:</b>	<code>++ptr</code> <code>ptr++</code> <code>--ptr</code> <code>ptr--</code> <code>ptr += 3</code> <code>ptr -= 3</code>										
<b>Distanz bestimmen:</b>	<code>ptr1 - ptr2</code>										
<b>Position vergleichen:</b>	<code>ptr1 &lt; ptr2</code> (Sonst: <code>&lt;=</code> , <code>&gt;</code> , <code>&gt;=</code> , <code>==</code> , <code>!=</code> )										
<pre>// Read 6 values into an array std::cout &lt;&lt; "Enter 6 numbers:\n"; int* a = new int[6]; int* pTE = a+6; for (int* i = a; i &lt; pTE; ++i)     std::cin &gt;&gt; *i; // read into array element  // Output: a[0]+a[3], a[1]+a[4], a[2]+a[5] for (int* i = a; i &lt; a+3; ++i) {     assert(i+3 &lt; pTE); // Assert that i+3 stays inside.     std::cout &lt;&lt; (*i + *(i+3)) &lt;&lt; ", "; } }</pre>											

# Programmier-Befehle - Woche 9

<code>*this</code>	Zugriff auf <b>implizites Argument</b>
<p>Memberfunktionen einer Klasse haben ein implizites Argument, nämlich das <b>aufrufende Objekt</b>. Und <code>this</code> ist ein Zeiger darauf. Via <code>*this</code> kann man darauf zugreifen.</p> <p>Bei Zugriffen von innerhalb einer Klasse aus auf Daten-Member oder Member-Funktionen wird das implizite Argument automatisch verwendet. Man muss es dann also <b>nicht unbedingt explizit angeben</b> (siehe Eintrag <b>Memberfunktion</b>). Man muss <code>*this</code> aber mindestens explizit verwenden, falls z.B. eine <b>Referenz auf das implizite Argument</b> zurückgegeben werden soll.</p>	
<pre>// General example class Human { public:     void set (const int a) { age = a; } // or (*this).age = a;     void print1 () const { std::cout &lt;&lt; (*this).age; }     void print2 () const { std::cout &lt;&lt; age; } // equivalent private:     int age; }; ... Human me; me.set(175); me.print1(); // 175 me.print2(); // 175 ----- // Another example class Complex { public:     // Note: In most applications     // a reference should be returned.     Complex&amp; operator+= (const Complex&amp; b) {         real += b.real;         imag += b.imag;         return *this;     }     ... // other members private:     float real;     float imag; };</pre>	

## Dynamische Datentypen

# Programmier-Befehle - Woche 9

<code>new</code>	Objekt mit dynamischer Lebensdauer erstellen.
<p>Mit <code>new</code> wird ein Objekt erstellt, indem der nötige Speicherplatz reserviert wird, und dann ein gegebener <b>Konstruktor</b> aufgerufen wird.</p> <p>Der Rückgabewert von <code>new</code> ist ein <i>Pointer</i> auf das neu erstellte Objekt.</p>	
<pre>Class My_Class { public:     My_Class (const int i) : y (i) { std::cout &lt;&lt; "Hello"; }     int get_y () { return y; } private:     int y; };  ... My_Class* ptr = new My_Class (3); // outputs Hello My_Class* ptr2 = ptr; // another pointer to the new object std::cout &lt;&lt; (*ptr).get_y(); // Output: 3 ...</pre>	

<code>new ... []</code>	Ranges mit dynamischer Lebensdauer und Länge erstellen.
<pre>int n; std::cin &gt;&gt; n; int* range = new int[n]; // Read in values to the range for (int* i = range; i &lt; range + n; ++i) std::cin &gt;&gt; *i;</pre>	

## Operatoren

<code>&amp;</code>	<b>Adressoperator</b> (siehe: <i>Adresse auslesen</i> unter Zeiger (generell), Summary 8)
Präcedenz: 16 und Assoziativität: <i>rechts</i>	

# Programmier-Befehle - Woche 9

<code>*</code>	<b>Dereferenz-Operator</b> (siehe: <i>Zugriff auf Objekt</i> unter <i>Zeiger</i> (generell))
Präzedenz: 16 und Assoziativität: rechts	

<code>...-&gt;...</code>	<b>Auf einen Member eines Objekts zugreifen</b> , auf das ein Pointer gegeben ist.
<code>ptr-&gt;mem</code> macht das Selbe wie <code>(*ptr).mem</code> .	
<pre>struct my_class {     // POST: "Hi! " is written to std::cout     void say_hi () const { std::cout &lt;&lt; "Hi! "; } }; ... my_class obj; my_class* ptr = &amp;obj; // just a pointer to obj obj.say_hi();        // direct access ptr-&gt;say_hi();        // using -&gt; (*ptr).say_hi();     // equivalent</pre>	