

## Dynamische Datentypen

<code>new, delete</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 <code>Konstruktor</code> aufgerufen wird. Bei <code>delete</code> wird zuerst ein Destruktor aufgerufen, bevor der Speicherplatz freigegeben wird.</p> <p>Der Rückgabewert von <code>new</code> ist ein <code>Pointer</code> auf das neu erstellte Objekt. Wird mit <code>delete</code> ein Objekt gelöscht, so sollte man immer <code>alle Pointer</code>, die auf das Objekt zeigen, auf <code>nullptr</code> setzen.</p> <p>Jedes <code>new</code> braucht ein <code>delete</code>. Sonst existieren die erstellten Objekte bis zum Ende des Programms, was je nach Laufdauer eine grosse Speicherverwendung ist.</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 delete ptr; ptr = nullptr; ptr2 = nullptr; // has to be done !separately! ...</pre>	

# Programmier-Befehle - Woche 12

<code>new ...[], delete[]</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; delete range; // ERROR: must say: delete[] delete[] range; // This works</pre>	

Copy-Konstruktor	Kopier-Initialisierung
Der Copy-Konstruktor ist der Konstruktor, dessen Argumenttyp <code>const My_Class&amp;</code> ist.	
<pre>struct Customer {     std::string name;     int duration;     int amount_insured; };  class Insurance { public:     Insurance (const Insurance&amp; rhs)         : length (rhs.length), ... // copy remaining data mbrs     {         cust = new Customer [length];         for (int i = 0; i &lt; length; ++i)             cust[i] = rhs.cust[i];     }     ... // other public members private:     Customer* cust; // pointer to an array containing customers     int length;    // length of cust     ... // other private members };</pre>	

# Programmier-Befehle - Woche 12

operator=	Kopier-Zuweisung
<p>Eng verwandt mit <code>operator=</code> ist der Copy-Konstruktor. Der Unterschied ist, dass der Copy-Konstruktor nur bei der <b>Initialisierung</b> aufgerufen wird, <code>operator=</code> hingegen nur <i>nach</i> der Initialisierung. z.B.</p> <pre>my_class a (5, 6), c (4, 4); // Call a general constructor my_class b = a;           // Call copy-constructor c = b;                   // Call operator=</pre> <p><code>operator=</code> kann anders als der Copy-Konstruktor implementiert werden müssen. Ein Beispiel sind Klassen, welche Pointer auf dynamisch generierte Objekte als Member haben. Dann muss bei <code>operator=</code> meistens zuerst das aktuell vorhandene Objekt gelöscht werden, bevor die Kopie erstellt werden kann. Dies ist beispielsweise beim Stack aus der Vorlesung relevant.</p> <p><code>operator=</code> gibt im Normalfall eine <b>Referenz</b> auf seinen linken Operanden zurück.</p> <p><b>Faustregel:</b> Meistens führt <code>operator=</code> zuerst die Aufgaben des <b>Destructors</b>, und dann die Aufgaben des <b>Copy-Konstruktors</b> aus.</p>	
<pre>// for Customer-struct see example on Copy-Constructor  class Insurance { public:     Insurance&amp; operator= (const Insurance&amp; rhs) {         // Cleanup of current customers         delete[] cust;          // Copy over the customers from rhs         cust = new Customer [length];         for (int i = 0; i &lt; length; ++i)             cust[i] = rhs.cust[i];         length = rhs.length;         ... // copy other data members         return *this; // return a reference to left operand     }     ... // other members };</pre>	

Destruktor	Class abbauen
<pre data-bbox="347 495 1173 719">// for Customer-struct see example on Copy-Constructor  class Insurance { public:     ~Insurance () { delete[] cust; } // free dynamic space     ... };</pre>	