

Datentypen

| struct | Container für Datentypen |
|--|--------------------------|
| <p>Wichtige Befehle:</p> <p>Definition: <pre>struct str_name { int mem1; bool mem2; int mem3; };</pre></p> <p>Objekt erstellen: <pre>str_name obj1;</pre></p> <p>mit Startwerten: <pre>str_name obj2 = {3, true, 4};</pre></p> <p>aus anderem Objekt: <pre>str_name obj3 = obj2;</pre></p> <p>Zugriff auf Member: <pre>obj1.mem1</pre></p> <p>Die <i>Definition</i> eines Structs hat ein <code>;</code> am Schluss.</p> <p>Nur der Zuweisungsoperator (=) wird automatisch erstellt (und kopiert dann die Member einzeln). Die anderen Operatoren (z.B. ==, !=, ...) muss man selbst passend überladen (siehe Eintrag operator...).</p> <p>Bei der Default-Initialisierung eines Objekts des Typs <code>str_name</code> werden alle Member einzeln default-initialisiert. Für fundamentale Typen (<code>int</code>, <code>float</code>, usw.) bedeutet das, dass sie <i>uninitialisiert</i> sind, bis man ihnen nachträglich einen Wert zuweist. Das führt zu Problemen, falls man ihren Wert vorher schon ausliest.</p> | |
| <pre>struct candidate { std::string name; // Name of the participant int age; // Her/his age }; int main () { // Initialization candidate mary; // default-initialisation std::cout << mary.age; // Undefined behavior mary.name = "Mary"; mary.age = 43; std::cout << mary.age; // Problem gone: mary.age is 43 candidate bob = {"Bob", 28}; // using starting values candidate fred = bob; // using other object fred.name = "Fred"; return 0; }</pre> | |

Operatoren

| <code>operator...</code> | Einen Operator überladen. |
|---|---------------------------|
| <p>Operator-Überladung wird zum Beispiel verwendet, um Operatoren (+, -, *, etc.) auf eigenen Structs zu definieren.</p> <p>Mittels dem <code>operator...</code> Keyword ist es ebenfalls möglich, den Operator auszuführen. Das sollte man aber vermeiden, da damit der Code unlesbar wird.</p> | |
| <pre>struct rational { int n; int d; // INV: d != 0 }; // POST: return value is the sum of a and b rational operator+ (const rational a, const rational b) { rational result; result.n = a.n * b.d + a.d * b.n; result.d = a.d * b.d; return result; } // POST: return value is the sum of a and b rational operator+ (const rational a, const int b) { rational b_rat; b_rat.n = b; b_rat.d = 1; // b_rat is b/1 return a + b_rat; // Use operator+ for two rationals (above) } int main () { rational r = {1, 2}; rational s = {3, 4}; rational t = r + s; // first overload std::cout << t.n << "/" << t.d << "\n"; // Output: 10/8 rational u = r + 3; // second overload std::cout << u.n << "/" << u.d << "\n"; // Output: 7/2 return 0; }</pre> | |