

Programmier-Befehle - Woche 8

Datentypen

<code>char</code>	Datentyp für Zeichen
<p>Literal: <code>'a'</code> für Zeichen (<i>einfache</i> Anführungszeichen) Literal: <code>"Hello World"</code> für Strings (<i>doppelte</i> Anführungszeichen)</p> <p><code>chars</code> können sehr einfach zu <code>int</code> hin und her umgewandelt 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 --> 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>	

<code>std::string</code>	komfortablerer Datentyp für Zeichen										
<p>Erfordert: <code>#include<string></code></p> <p>Vorteile:</p> <table><tr><td>variable Länge:</td><td><code>std::string my_str (n, 'a');</code> (n kann variabel sein)</td></tr><tr><td>Länge abfragen:</td><td><code>my_str.length()</code></td></tr><tr><td>vergleichbar:</td><td><code>text1 == text2</code></td></tr><tr><td>hintereinander hängen:</td><td><code>text1 += text2</code></td></tr><tr><td>bequemer Output:</td><td><code>std::cout << my_str;</code></td></tr></table>		variable Länge:	<code>std::string my_str (n, 'a');</code> (n kann variabel sein)	Länge abfragen:	<code>my_str.length()</code>	vergleichbar:	<code>text1 == text2</code>	hintereinander hängen:	<code>text1 += text2</code>	bequemer Output:	<code>std::cout << my_str;</code>
variable Länge:	<code>std::string my_str (n, 'a');</code> (n kann variabel sein)										
Länge abfragen:	<code>my_str.length()</code>										
vergleichbar:	<code>text1 == text2</code>										
hintereinander hängen:	<code>text1 += text2</code>										
bequemer Output:	<code>std::cout << my_str;</code>										

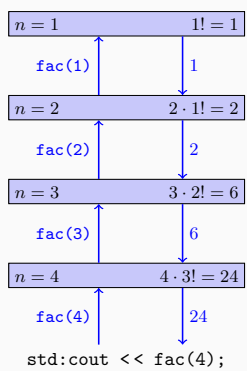
(...)

Programmier-Befehle - Woche 8

(...)

```
std::string my_word (5, 'a'); // initialize my_word as aaaaa
std::string ref (5, 'z');
my_word += ref; // append ref to my_word.
                // Afterwards my_word:  aaaaazzzzz
                // Afterwards ref:      zzzzz
std::cout << my_word.length() << "\n"; // output: 10
my_word[3] = 'b'; // change my_word to aaabzzzzz
if (my_word == ref) { // false
    std::cout << "not output\n";
}
std::cout << my_word << "\n"; // output whole string at once
```

Funktionen

Rekursion	Selbstaufruf einer Funktion
<p>Jeder rekursive Funktionsaufruf hat seine eigenen, unabhängigen Variablen und Argumente. Dies kann man sich sehr gut anhand des in der Vorlesung gezeigten Stacks vorstellen (<code>fac</code> ist im Beispiel unten definiert):</p>  <pre>graph TD n1["n = 1"] -- "1!" --> n2["n = 2"] n2 -- "2 · 1!" --> n3["n = 3"] n3 -- "3 · 2!" --> n4["n = 4"] n4 -- "4 · 3!" --> output["std::cout << fac(4);"] output -- "24" --> n4 n4 -- "6" --> n3 n3 -- "2" --> n2 n2 -- "1" --> n1</pre>	
<pre>// POST: return value is n! int fac (const int n) { if (n <= 1) return 1; return n * fac(n-1); // n > 1 }</pre>	