Datentypen

Vektoren (mehrdim.)	eines bestimmten Typs

Erfordert: `#include <vector>`

Wichtige Befehle:

Definition:
```
std::vector<std::vector<int> >
  my_vec (n_rows, std::vector<int>(n_cols,
                                   init_value))
```

Zugriff:
```
my_vec[1][1] = 8 * my_vec[0][2];
```

(Anstatt `int` gehen natürlich auch andere Typen.)
(Die Definition kann auch ohne Initialisierung erfolgen:
`std::vector<std::vector<int> > my_vec (n_rows, std::vector<int>(n_cols)) )`

Beachte: `> >` muss mit Abstand geschrieben werden. Sonst gibt es Doppeldeutigkeiten mit dem Eingabe-Operator `>>` (siehe `std::cin`).

```
std::vector<std::vector<int> > my_vec (2, std::vector<int>(4, 0));
my_vec[1][2] = 3;
    // my_vec becomes
    //     0,  0,  0,  0
    //     0,  0,  3,  0
```

`std::string`	**komfortablerer** Datentyp für **Zeichen**

Erfordert: `#include <string>`

Vorteile gegenüber `char`-Arrays:

variable Länge:	`std::string my_str (n, 'a');` (`n` kann variabel sein)
Länge abfragen:	`my_str.length()`
vergleichbar:	`text1 == text2`
hintereinander hängen:	`text1 += text2`
bequemer Output:	`std::cout << my_str;`

```
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
for (int i = 0; i < my_word.length(); ++i)
    std::cin >> my_word[i]; // read user input into our word
if (my_word == ref) // impossible since lengths differ (5 VS 10)
    std::cout << "never output\n";
std::cout << my_word << "\n"; // output whole string at once
```

`std::stringstream`	Datentyp für **String-Streams**

Erfordert: `#include <sstream>`

Man kann Objekte dieses Typs beispielsweise verwenden, um dem Programm eine Eingabe via `std::cin` vorzutäuschen.

Beachte: "*String*-Stream" heisst, dass im Objekt ein String *enthalten* ist. Es heisst aber **nicht**, dass nur Strings (oder `chars`) daraus *extrahiert* werden können. Darin können beispielsweise auch Zahlen als String vorliegen. Diese kann man als String (oder `char`) oder als Zahl extrahieren.

Objekte des Typs `std::stringstream` können nicht direkt kopiert werden. Deshalb sollte man sie immer via Call-by-Reference an Funktionen übergeben.

```
char c;
std::cin >> c; // e.g. value: 'b'
std::stringstream s ("b345e"); // stringstream with value "b345e"
s >> c; // c gets value 'b' and s is now "345e"

s >> c; // c gets value '3' (!as char! since type of c is char)
int n;
s >> n; // n gets value 45 (!as int! since type of n is int)
        // (This works since the computer sees that the next
        // 2 characters in the string "45e", namely '4' and '5',
        // can be used as the int 45. So after this operation
        // s is "e".)
```

`std::istream`	Datentyp für **Input-Streams**

Erfordert: `#include <istream>` oder `#include <iostream>`

Beispielsweise `std::cin` hat den Typ `std::istream`. Objekte des Typs `std::stringstream` können auch als `std::istream` verwendet werden (siehe `calculator.cpp`).

Objekte des Typs `std::istream` können nicht direkt kopiert werden. Deshalb sollte man sie immer via Call-by-Reference an Funktionen übergeben.

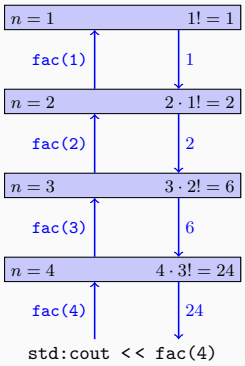
```
// POST: Two characters are removed from is. If is contains less
//       characters it is emptied.
void remove_two (std::istream& is) {
    char a;
    is >> a >> a; // remove two chars
}

int main () {
    // Assume that the user enters "Informatics".
    remove_two(std::cin);
    char out;
    while (std::cin >> out)
        std::cout << out; // Output: formatics
    return 0;
}
```

Funktionen

Rekursion	**Selbstaufruf** einer Funktion

Jeder rekursive Funktionsaufruf hat seine eigenen, unabhängigen Variablen und Argumente. Dies kann man sich sehr gut anhand des in der Vorlesung gezeigten Stacks vorstellen (`fac` ist im Beispiel unten definiert):




```
// POST: return value is n!
unsigned int fac (const unsigned int n)
{
    if (n <= 1) return 1;
    return n * fac(n-1); // n > 1
}
```

Input/Output

`leerer Eingabestrom`	Prüfe, ob **mehr Eingaben vorhanden** sind.

Dahinter steckt eine Konvertierung von `std::cin` zu `bool`:

`true:`	weitere Eingaben vorhanden
`false:`	keine Eingaben mehr vorhanden

Wir brauchen diese Abfrage meistens, um eine Schleife solange laufen zu lassen, wie weitere Eingaben vorhanden sind. (siehe Beispiel unten)

Erfolgt die Eingabe per Tastatur, so kann die Eingabe durch drücken von `[Ctrl]+[D]` beendet werden.

```
char input;
int length_of_text = 0;
while(std::cin >> input)
    ++length_of_text;
std::cout << length_of_text;
```