

Um die Ein- und Ausgabe nach eigenen Wünschen gestalten zu können, haben Sie in C++ die Möglichkeit, den Datenstrom durch Formatierungen zu gestalten.

In der Vorlesung wurde die Ein- und Ausgabe verschiedener Daten mit Strömen vorgestellt. Dabei waren Sie auf die Standardformatierungen der iostream Library angewiesen. Problematisch kann dies dann werden, wenn Sie bestimmte Wünsche hinsichtlich der Gestaltung der Ausgabe haben. Diese können sie mit Formatierungen verändern.

Width und Fill

Mit `width()` wird die Weite der nächsten Zahlen- oder Textausgabe abgefragt (kein Parameter) bzw. festgelegt (der Parameter gibt die Weite an). Nach jeder Ausgabe wird die Weite automatisch auf 0 zurückgesetzt. Z.B.:

```
cout << 7;
cout.width(6); //Ausgabeweite: 6 Zeichen
cout << 11;
cout << "#" << cout.width();
// Ausgabe: "7    11#0"
```

Vor der zweiten Zahl wurden vier Leerzeichen eingefügt, um mit den zwei Zeichen von "11" auf die Gesamtweite von sechs Zeichen zu kommen.

Mit `fill()` lassen sich auch andere Füllzeichen anstelle des Leerzeichens ausgeben. Auch hier wird ohne Parameter abgefragt und mit Parameter ein neues Zeichen gesetzt. Z.B.:

```
char AltesZeichen = cout.fill(); // Füllzeichen merken
cout.width(6);
cout.fill('0');
cout << 17 << " und " << 4;
cout.fill(AltesZeichen); // altes Füllzeichen setzen
```

erzeugt die Ausgabe "000017 und 4", da die Angabe der Weite ja nur für die nächste Ausgabe gilt.

Vorsicht! Die beiden Methoden `width()` und `fill()` gelten nicht für Ausgaben des Datentyps `char`! Z.B.:

```
cout.width(6);
cout.fill('_');
cout << '(' << 12 << ')';
```

erzeugt die Ausgabe "(____12)", da die erste Ausgabe (die Klammer "(") vom Typ `char` ist und damit die Formatangaben erst für die nächste Ausgabe gelten.

Precision

Mit `precision()` wird die Weite von Fließkommazahlen gesteuert. Dabei gibt der Parameter an, wie viele Ziffern insgesamt vor und nach dem Komma von der Zahl ausgegeben werden sollen (ohne den Dezimalpunkt mitzurechnen!), sofern die Flags `fixed` oder `scientific` nicht gesetzt sind. Mehr über Flags wird Ihnen ihr Kollege erzählen. Andernfalls legt `precision()` die Anzahl der Nachkommastellen fest. Auch hier wird ohne Parameter abgefragt und mit Parameter ein neuer Wert gesetzt. Z.B.:

```
int AlterWert = cout.precision(); // Einstellung merken
cout.precision(8);
cout << 1234.56789 << "\n";      // Ausgabe: 1234.5678
cout.precision(4);
cout << 1234.56789 << "\n";      // Ausgabe: 1235
cout.precision(AlterWert);      // wiederherstellen
```

Die Ausgabe wird automatisch gerundet falls die Anzahl Nachkommastellen den Wert von `precision()` überschreitet. Falls die Anzahl Vorkommastellen den Wert von `precision()` überschreitet wird auf die wissenschaftliche Notation - also Ausgabe mit Exponent - umgeschaltet.

Flush und Eof

Mit `cout` wird eine gepufferte Ausgabe durchgeführt, d.h. die auszugebenden Zeichen werden nicht sofort, sondern erst etwas später ausgegeben. Gerade bei der Fehlersuche, aber auch bei Benutzerhinweisen und -aufforderungen, führt dies zu ungewollten Effekten. Mit `flush()` kann man das Programm zwingen alles im Ausgabepuffer stehende komplett auszugeben. Dies kann ebenfalls mit der Ausgabe von `endl` erreicht werden. Es bewirkt neben dem Zeilenvorschub auch ein `flush`. Z.B.

```
cout << 1234.56789 << "\n";
cout.flush();

cout << 1234.56789 << endl;
```

Dieses Verhalten einer nicht gepufferten Ausgabe kann auch mit dem Flag oder Manipulator `unitbuf` erreicht werden.

`eof()` liefert den Wert 0, solange das Ende des Stroms (der Datei) noch nicht erreicht wurde, danach ein Wert ungleich 0.

```
ifstream examplefile ("example.txt");
char buffer[100];

if (! examplefile.is_open())
{
    cout << "Error opening file";
    exit (1);
}

while (! examplefile.eof() )
{
    examplefile.getline (buffer,100);
    cout << buffer << endl;
}
```