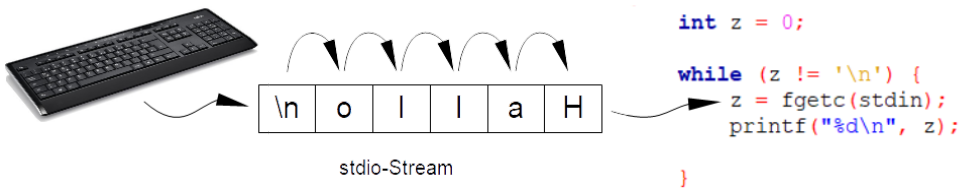
4 Ein-/Ausgabe

Ein-/Ausgabe-Streams: das Lesen und Schreiben von Daten aus oder in ein C-Programm funktioniert über Streams (Datenströme). Werden Daten von der Tastatur gelesen, dann wird dem stdin-Stream ein Zeichen eingefügt, das auf der Tastatur gedrückt wird:



Der Buffer ist als FIFO-Buffer (First-In-First-Out) organisiert, also wird der erste Wert der in den Buffer reinkommt auch als erstes wieder ausgelesen. Die Reihenfolge der Eingabe entspricht also der Reihenfolge der Ausgabe. Der stdio-Stream wird also von der Tastatur befüllt und vom C-Programm wieder geleert.

Für Ausgabe von Daten aus einem C-Programm (Text anzeigen, in Datei schreiben ...) funktioniert das genau umgekehrt. Es gibt hier allerdings zwei Streams stdout (normale Ausgaben) und stderr (Fehlermeldung-Ausgaben). Das C-Programm schreibt die auszugebenden Daten in die Ausgabestreams. Die Daten werden anschließend in gleicher Reihenfolge in der Konsole (Bildschirm) ausgegeben.

Das C-Programm hat also nicht die Möglichkeit ein Zeichen direkt am Bildschirm auszugeben, es kann ein Zeichen nur in den Ausgabestream schreiben. Genauso kann es ein Zeichen nicht direkt von der Tastatur lesen, es kann lediglich ein Zeichen aus dem Eingabestream lesen.

Ausgabe

Zum Ausgeben von Daten am Bildschirm eignet sich die Funktion **int printf(format, parameter ...)**. Mit Platzhaltern können hier Parameter wie Variable mit ausgegeben werden. Der Parameter *format* ist eine Zeichenkette der Form `"...."`. Mit diesem Parameter wird angegeben was in welche Reihenfolge ausgegeben werden soll. Wenn neben Text auch Variablen ausgegeben werden sollen, dann werden diese in *format* als Platzhalter angeführt. Zum Beispiel wird mit `"Hallo %s, wie geht's"` der Text "Hallo xxxxxxx, wie geht's" ausgegeben. Statt des Platzhalters %s wird ein Wert eingefügt, der nach diesem *format*-String als Parameter angeführt wird (%s bedeutet Parameter als String einfügen). Platzhalter beginnen immer mit einem %-Zeichen. Welche Art von Platzhalter es gibt, ist weiter unten unter *Formatbezeichnungen* angeführt. in *printf* müssen immer soviele Parameter (Variablen ...) hinter *format* angeführt werden wie Platzhalterzeichen in *format* verwendet wurden. Umgekehr müssen auch für: `printf("....", x, y, z)` im *format*-String genau 3 Platzhalter angeführt sein um die Variablen x, y und z einfügen zu können. Die Typen der Platzhalter müssen ebenfalls mit den Parametern zusammenpassen. Die Funktion *printf* schreibt immer in den stdout-Stream.

Sehr ähnlich wie die *printf*-Funktion kann auch die **int fprintf(stream, format, parameter)** Funktion verwendet werden. Der einzige Unterschied ist nur, dass als zusätzlicher Parameter der Ziel-Stream in welchem geschrieben werden soll mit angeführt werden kann. Für *stream* kann also *stdout* (Bildschirm), *stderr* (für Fehler, ebenfalls auf dem Bildschirm) oder ein beliebiger Datei-Stream (siehe Dateizugriffe später) verwendet werden. Der Vorteil des Stream-Konzepts ist, dass statt eines Datei-Streams auch der stdout-Stream eingesetzt werden kann, dann wird anstatt in einen Datei-Stroms in den stdout-Stream, also in die Konsole (Bildschirm) geschrieben.

Soll ein einfaches Zeichen in den stdout-Stream ausgegeben werden, dann kann die **int putchar(int c)** Funktion verwendet werden. Da ASCII nur 127 Zeichen kennt, der Parameter *c* aber vom Typ int ist (65536 mögliche Werte) kann auch ein *EOF* (-1) in den Stream geschrieben werden kann.

Es kann genauso die File-Put-Character-Funktion **fputc(zeichen, stream)** verwendet werden. Wie der Name sagt, ist *fputc* eine Funktion zum Schreiben in eine Datei. Für Dateien muss immer zuerst ein Datenstrom in eine Datei geöffnet werden (siehe später Dateizugriff).

Zusammengefasst

- Ausgabe in Standard-Ausgabe-Stream: **printf**
- Ausgabe in beliebigen Stream: **fprintf**
- Ausgabe in Zeichenkette: **sprintf**

Eingabe

Beim Lesen von Zeichen von der Tastatur ist es wichtig zu verstehen, dass in C vom Eingabe-Stream stdin gelesen wird. Es wird nicht direkt von der Tastatur gelesen sondern das nächste/die nächsten Zeichen aus dem stdin-Stream geholt. Die können durchaus schon länger darin liegen. Bei der Eingabe über die Tastatur wird die Eingabe in den Eingabe-Stream erst durch Eingabe von <Enter> abgeschlossen.

Mit den gleichen Formatbezeichnern wie beim printf() kann mit **int scanf(format, &parameter ...)** Eingabestream eingelesen werden. Eine scanf-Funktion wird solange ausgeführt (die C-Code-Zeile wird erst dann verlassen!) bis

- entweder der komplette Formatierungsstring erfüllt ist
- sich beim Drücken der Eingabetaste während eines Teils der Eingabe herausstellt, dass der Formatierungsstring bislang nicht erfüllt ist

Im folgenden Beispiel `scanf("%d\n%d\nabc", &num1, &num2);` wird etwa erwartet, dass eingegeben wird:

- eine Zahl gefolgt von einem <Enter> (\n)
- eine Zahl gefolgt von einem <Enter> (\n)
- die Zeichen abc gefolgt von einem <Enter> (Abschluss der Eingabe)

Bei Eingabe von: `1<Enter>2<Enter>` ist scanf noch nicht beendet. Da das Programm noch auf die Eingabe von "abc" wartet, muss noch einmal eine Eingabe getätigt werden und mit <Enter> abgeschlossen werden. Diese letzte Eingabe wird mit "abc" verglichen, wenn "abc" eingegeben wurde, wird dieses "abc" aus dem Buffer entnommen. Wurde etwas anderes entnommen, dann wird num1 und num2 zwar korrekt übernommen, allerdings verbleibt der Teil der nicht abc entspricht im stdin-Stream, da ja eine andere Zeichen-Folge nicht als Teil dieser scanf-Ausführung gewünscht war. Das macht sich bemerkbar wenn später wieder lesend auf den Stream zugegriffen wird (z.B. erneutes Einlesen von der Tastatur), dann werden zuerst diese verbliebenen Zeichen übernommen.

Die Funktion scanf() hat in der Klammer die Aufrufparameter und Referenzen auf Variablen in welche die Eingabe eingelesen werden soll. Auch in C kann nicht in Aufrufparameter einer Funktion geschrieben werden. Durch den &-Operator wird der Methode allerdings die Adresse/Referenz auf eine Variable mitgegeben. Die Methode kann somit direkt auf die Adresse der Variable schreiben und somit auch die Variable mit der Eingabe beschreiben.

```
char zeichen;
scanf("%c", &zeichen);
```

Hier wird ein Zeichen (%c) in die char-Variable zeichen eingelesen. Auch wenn hier nur ein einzelnes Zeichen von der Tastatur eingelesen werden soll, wird die Eingabe in den stdin-Stream erst durch Eingabe von <Enter> abgeschlossen. Der Aufruf von

```
scanf("%c", zeichen);
```

führt zu einem Fehler.

Mit der Funktion **int fgetc(stream)** wird ein einzelnes Zeichen aus einem mit angegebenen Input-Stream gelesen. Sie eignet sich daher auch für Einlesen aus einer Datei (f steht für File). Für das Einlesen von der Tastatur wird statt eines Datei-Streams einfach der stdin-Stream mit angegeben: **int fgetc(stdin)**.

fgetc() nimmt nur ein Zeichen aus dem Stream. Die Eingabe in dem Stream mit der Tastatur wird allerdings erst durch Eingabe von <Enter> abgeschlossen. Werden davor noch weitere Zeichen in den Stream eingegeben, verbleiben diese im Stream. Bei einem weiteren Aufruf dieser Funktion werden daher diese verbleibenden Zeichen eingelesen. Da fgetc() auch für das Einlesen aus einer Datei verwendet wird, kann auch ein *End-Of-File* Zeichen eingelesen werden. Dieses Zeichen entspricht der Nummer -1, daher gibt die Funktion einen int-Wert und keinen char-Wert zurück. Die Tastatur kann in Windows ein EOF mit <Strg>+Z emulieren.

Manchmal soll ohne Umweg über den Eingabe-Stream direkt von der Tastatur gelesen werden. Beim Lesen über einen Stream wird immer auf eine Eingabe von <Enter> gewartet. Mit der Nicht-Standard-Funktion **int getch()** steht eine entsprechende Funktion zur Verfügung. Es wird nicht auf die Eingabetaste zum Abschluss gewartet. Die Funktion liesst nicht aus dem stdin-Stream. Mit dieser Funktion können auch Sondertasten (F1, Cursor-Tasten ...) erfragt werden. Im Fall von Sondertasten werden allerdings 2 Nummern übergeben. Die erste Nummer zeigt an ob es sich eine Sondertaste handelt (0 oder 244) oder nicht (wenn ein ASCII-Code zwischen 1 und 127). Wenn es sich um eine Sondertaste handelt dann kann die zweite Nummer direkt mit einem zweiten Aufruf von getch() geholt werden:

```
int key = getch();
if ((key == 244) && (getch() == 72)) printf("Pfeil rauf gedrueckt!\n");
```

Wird die Pfeil-Hinauf-Taste gedrückt, dann wird augenblicklich die Nummer 244 aus dem ersten getch() Aufruf zurückgegeben. Mit einem zweiten Aufruf von getch() wird ohne weiteren Tastendruck sofort die Nummer 72 zurückgegeben.

Zum Einlesen von Zeichenketten (~Strings) eignet sich die Funktion **char *fgets(str, n, stream)**. Mit dieser Funktion wird aus dem *stream* (kann für Tastatur auch stdin sein) bis zu *n* Zeichen in *str* gelesen werden. Es werden dabei n Zeichen oder bis zum Zeilenumbruch oder zum EOF (Ende der Datei) gelesen, was immer davon früher kommt. Dadurch kann sichergestellt werden, dass keine zu lange Zeichenkette in die Zielzeichenkette zugewiesen wird. Als Rückgabewert wird **NULL** zurückgeliefert, wenn kein einziges Zeichen geliefert wird (zum Beispiel EOF). Im Kapitel Strings wird auf das Arbeiten mit Zeichenketten eingegangen.

Das Einlesen von Werten von der Tastatur führt häufig zu Problemen. Für Mehrfach-Eingaben vieleicht am einfachsten zu bedienen sind die Funktionen **fgetc** und **fgets**.

Zusammengefasst

- Einlesen zusammengesetzter Größen (Datum...) mit sehr eingeschränkter Fehlerbehandlung **scanf**
- Einlesen Zeichen für Zeichen aus dem Eingabe-Stream **fgetc**
- Einlesen einer Zeichenkette aus dem Eingabe-Stream **fgets**
- Einlesen direkt von der Tastatur (ohne Stream) nur in Windows **getch**

Stream Leeren

Sollen von einem Stream nur neu eingefügte Zeichen gelesen werden, dann kann der Stream geleert werden mit **fflush(stream)**:

```
fflush(stdin);
```

Allerdings kommt es vor, dass die Funktion nicht wie gewünscht funktioniert (das exakte Verhalten ist nicht standardisiert).

Escape-Sequenzen

Um sich auf spezielle Zeichen in einem C-Programm beziehen zu können gibt es sogenannte Escape-Sequenzen. Soll etwa ein Tabulator als Zeichen angegeben werden, dann kann das mit dem Zeichen '\t' erfolgen. Das ist eine einzelnes Zeichen (und nicht Backslash und t). Escape-Sequenzen werden immer mit einem Back-Slash eingeleitet und von einem Buchstaben gefolgt. Beispiele:

- \n Zeilenumbruch (ASCII 10)
  \r Carrige Return (ASCII 13)
  \t Tabulator (ASCII 9)

Diese Sonderzeichen können überall verwendet werden wo auch "normale" Zeichen angeführt werden können. Sie können natürlich auch durch ihren ASCII Code angegeben werden.

Anhang: Formatbezeichnungen

Formatbezeichnungen werden verwendet um Werte formatiert auszugeben oder einzulesen.

Beispiel für Ausgabe eines Characters (%**c**):

```
printf("%c", charVal);
```

Der Bezeichner setzt sich aus optionalen und verpflichtenden Anteilen zusammen:

%fm.G**T**

Die Einleitung **%** und der Formatbezeichner **T** muss immer angeführt werden und beginnt bzw. beendet den Bezeichner. Es bedeuten:

%

Muss. leitet Formatierungssymbol ein.

f

Optional. Flag/Kennzeichnung:

Kennzeichnung	Bedeutung
-	linksbündig ausrichten
0	führende Felder mit 0 ausfüllen
+	Vorzeichen immer ausgeben
leer	positive Zahl mit Leerzeichen statt +
#	unterschiedliche Bedeutung

ein # an dieser Stelle hat unterschiedliche Bedeutung:

#	Bedeutung
%#o	Zahl wird mit Oktalen-Prefix 0 angezeigt
%#x	Zahl wird mit Hex-Prefix 0x angezeigt
%#X	Zahl wird mit Hex-Prefix 0X angezeigt
%#e	Dezimalpunkt wird immer angezeigt
%#E	Dezimalpunkt wird immer angezeigt
%#f	Dezimalpunkt wird immer angezeigt
%#g	Dezimalpunkt wird immer angezeigt. Nullen nach Dezimalpunkt werden nicht gelöscht
%#G	Dezimalpunkt wird immer angezeigt. Nullen nach Dezimalpunkt werden nicht gelöscht

m

Optional. Nummer gibt die minimale Anzahl an Feldern an. Werte werden standardmäßig rechtsbündig eingefügt, ausser es wurde für f ein "-" angegeben.

.G

Optional. Möchte man die Genauigkeit (Stellen hinter dem Komma) angeben, dann muss ein Punkt mit der Anzahl der Stellen hinter dem Komma folgen.

T

Muss. Der Typ des Werts:

Formatbezeichner	Bedeutung
c	Character
d	int-Ganzzahl mit Vorzeichen (oder %i)
i	int-Ganzzahl mit Vorzeichen (oder %d)
f	double-Fließkommazahl ([-]mmm.ddd) - Genauigkeit legt Anzahl d fest, mit 0 kein Dezimalpunkt
e	double-Fließkommazahl (oder %E) in Exponentendarstellung
g	double-Fließkommazahl (oder %G), wie **e** wenn Exponent kleiner **-4** sonst wie **f**
p	Einheitliche Zeigerdarstellung (Adresse)
%n	Anzahl der bisherigen Argumente - dieser Formatierer benötigt keinen Parameter
g	Fließkommazahl (oder %G)
o	Oktalzahl ohne Vorzeichen und führende Null
x	Hex-Zahl mit Kleinbuchstaben (abcdef) - alternativ %X für Großbuchstaben (ABCDEF)
u	Vorzeichenlose int-Ganzzahl
s	Zeichenkette (String / char-Array) bis \0-Abschluss (oder so viele Zeichen wie in Genauigkeit angegeben)

d kann maximal den signed Integer Bereich darstellen. Zahlen die darüber hinausgehen benötigen mehr Breite. In vielen Kompilern gibt's dann:

lu (unsigned long), lld (long long dezimal), llu (long long unsigned), I64u (64 Bit Integer unsigned) und andere mehr.

Für das Einlesen (scan ...) gibt es ausserdem noch %[...] um eine Gruppe erlaubter Zeichen anzugeben und %[^...] um eine Gruppe nicht erlaubter Zeichen anzugeben (ähnlich regex). So bedeutet "%10[0-9]" bis zu 10 Zeichen der Gruppe 0-9. Oder "%[^0-9]" bis zu 10 Zeichen der Gruppe alles ausser 0-9.

Beispiele:

#1:

```
int number=5;
char *pointer="little";

printf("Here is a number-%4d-and a-%10s-word.\n", number, pointer);
// Here is a number-   5-and a-    little-word.
```

#2:

```
int number = 3;

printf(" %-4d_\n", number);
printf(" %04d_\n", number);
printf(" %-#4x_\n", number);
printf(" %#x_\n", number);

// 3   _
// 0003_
// 0x3 _
// 0x3_
```

#3: Lesen einer kompletten Zeichenkette inklusive Whitespaces:

```
sscanf("ab abab\tcdef", "%[^\0]", strWert);
```

```
sscanf("abcdef", "%s%n", strWert, &numChars);   // numChars erhält 6
sscanf("123cdef\n", "%1i%2is%n", &num1, &num2, strWert, &numChars);     // numChars erhält 8 (\n kommt auch dazu)
```

```
sscanf("abababcdef", "%[abc]", strWert);        // strWert->"abababc"  def sind nicht in der Gruppe abc enthalten
```

```
sscanf("abababcdef", "%[^e]", strWert);     // strWert->"abababcd"  e darf nicht in der Gruppe sein
```

Referenz

siehe Formatspezifizierer, z.B. https://www.tutorialspoint.com/c_standard_library/c_function_printf.htm

http://w3adda.com/c-tutorial/c-escape-sequence-or-backslash-character

http://www.cplusplus.com/reference/cstdio/fscanf/

Fragen

- Geben Sie eine int-Zahl in hexadezimaler Form aus.
- Lesen Sie mehrere numerische Werte hintereinander von der Konsole ein.