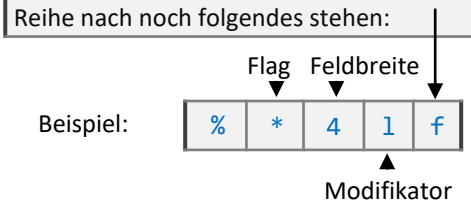
Merkblatt zur Programmierung in C

Formatierte Eingabefunktionen deklariert in <stdio.h>

Formatierte Eingabefunktionen ...scanf()	Beispiel
`int fscanf(FILE *fp, char *format, &arg1, &arg2, …);`	`n = fscanf(fp,"%f", &x);`
`fscanf()` liest Zeichen vom Datenstrom `fp` und interpretiert sie unter der Kontrolle der Zeichenkette `format` und legt die Resultate in den restl. Argumenten ab (**müssen alle Zeiger sein!**). Liefert die Anzahl erfolgreicher Zuweisungen zurück.	
`int scanf(char *format, &arg1, &arg2, …);`	`n = scanf("%10s", str);`
`scanf(format, …)` ist definiert als `fscanf(stdin, format, …)` und liest von der Standardeingabe (stdin). Die Eingabe wird **erst** durch die Eingabetaste (`LF` = `'\n'`) abgeschlossen. Nicht gewandelte Zeichen bleiben im Eingabestrom (-puffer) stehen und werden beim nächsten Aufruf wieder bereitgestellt!	
`int sscanf(char *string, char *format, &arg1, &arg2, …);`	`n = sscanf(str, "%d", &i);`
`sprintf()`, liest Zeichen von der Zeichenkette `string`.	
Die `format`-Zeichenkette kann folgendes enthalten, bzw. es gilt folgendes:	
• Leerzeichen oder TAB bzw. Zwischenraumzeichen (whitespaces). Diese führen dazu, dass auch Zwischenraumzeichen in der Eingabe ignoriert werden.	
• Gewöhnliche Zeichen (nicht aber `%`), die dem nächsten Zeichen nach dem Zwischenraum im Eingabestrom entsprechen müssen. Sind diese nicht vorhanden, erfolgt keine Umwandlung und `scanf()` bricht ab, ohne die weiteren Eingaben zu wandeln. Die Werte der Variablen, deren Adressen übergeben wurde, bleibt dann **unverändert**.	
• Nicht zugewiesene Zeichen bleiben im Eingabepuffer stehen! Rückgabewert von `scanf()` überprüfen!	
• Umwandlungsangaben, bestehend aus `%`, optionale weitere Formatangaben und einem **Umwandlungszeichen**:	

Zeichen	Arg. Typ	passende Eingabedaten
i	`int*`	Ganze Zahl (Basis (0) oder (0x) wird erkannt).
d	`int*`	**d**ezimale ganze Zahl (0 und 0x auch möglich)
o	`int*`	**o**ktale ganz Zahl.
x, X	`int*`	He**x**adezimale ganze Zahl.
u	`unsigned int*`	dezimale Zahl ohne Vorzeichen (**u**nsigned).
c	`char*`	Ein oder mehrere Zeichen (**c**har); ohne `'\0'`
s	`char*`	Zeichenkette (**s**tring) Alle Zeichen bis zum nächsten Zwischenraum; Abschluss mit `'\0'`
e, f, g	`float*`	Gleitkommazahl (**f**loat); Optional mit Vorzeichen, Dezimalpunkt und Exponenten.
[…]	`char*`	Suchzeichen: erlaubt nur Zeichen aus der Menge in […]. (z.B. [a-z] nur Kleinbuchstaben)
[^…]	`char*`	erlaubt nur Zeichen , die NICHT in der Menge […] sind. (z.B. [^abc] kein a, b oder c).
n	`int*`	Anzahl benutzter Zeichen aus Eingabe wird dem nächstem (int) Argument zugewiesen.
%		Ein **%** wird erkannt; keine Zuweisung.

Zwischen **%** und den **Umwandlungszeichen** kann der Reihe nach noch folgendes stehen:



Typ	Arg.	Funktion
Flag	`*`	Eingabe wird überlesen und eine Zuweisung verhindert
Feldbreite	`zahl`	Maximale Feldbreite
Modifikator	`h`	Als **h**ort (z.B. `short int*`)
Modifikator	`l`	**l**ong (z.B. `long int*` , `double*`) – anders als printf!
Modifikator	`ll`	**l**ong **l**ong
Modifikator	`L`	**L**ong double

Beispiele

```
int n, i, j, k;
char s[128];
```

~~Durchgestrichene~~ Zeichen in der Eingabe sind Zeichen, die "aufgebraucht", d.h. aus dem Eingabepuffer entnommen wurden. ␣ ist das Leerzeichen, ↵ die Eingabetaste

scanf() Aufruf	Eingabe	Variablenwerte nach Aufruf
`n = scanf("%d%d", &i, &j);`	~~12~~␣,␣34↵	`n:1; i:12, j:unverändert`
`n = scanf("%d,%d", &i, &j);`	~~12~~␣,␣34↵	`n:1; i:12, j:unverändert`
`n = scanf("%d ,%d", &i, &j);`	~~12~~␣~~,~~␣~~34~~↵	`n:2; i:12, j:34`
`n = scanf("%d, %d", &i, &j);`	~~12~~␣,␣34↵	`n:1; i:12, j:unverändert`
`n = scanf("%*d%d", &i);`	12␣34↵	`n:1; i:34`
`n = scanf("%*s%s", s);`	~~Liebe␣Frau~~␣Maier↵	`n:1; s:"Frau"`
`n = scanf("%1d%2d%3d", &i, &j, &k);`	~~12345~~↵	`n:3; i:1, j:23, k:45`
`n = scanf("%[0123456789]", s);`	~~123~~abc↵	`n:1; s:"123"`
`n = scanf("%[^0123456789]", s);`	~~abc~~123↵	`n:1; s:"abc"`