C

Zusammenfassung für nützliche Sachen beim Programmieren mit C.

Input seperated with space or /

https://stackoverflow.com/questions/15330047/understanding-scanf-dealing-with-formatted-input
To match both the space-separated and the slash-separated inputs, you'll need a modestly complex format string:

```
if (scanf("%[^ /]%*1[ /]%d%*1[ /]%f", name, &age, &wage) == 3)
    ...data was read properly...
else
    ...something went wrong...
```

The first conversion specification is a scan set that accepts a sequence of non-blanks, non-slashes (so it will stop at the first blank or slash). It would be best to specify an upper bound on how many characters will be accepted so as to avoid stack overflow; for example, if char name[32];, then %31[^ /] (note the off-by-one). The second conversion specification *%1[/] accepts a single character (1) that is either a blank or slash [/], and does not assign it to any variable (*). The third conversion specification is a standard numeric input, skipping leading blanks, allowing for negative numbers to be entered, etc. The fourth conversion specification is the same as the second, and the fifth is a standard format for a float (which means that 34000.25 with 7 significant digits is at the outer end of the range of representable values).

Note that the 'something went wrong' part has a difficult time reporting the error coherently to the user. This is why many people, myself included, recommend against using scanf() or fscanf() and prefer to use fgets() or perhaps POSIX getline to read a line from the user and then use sscanf() to analyze it. You can report the problems much more easily. Also note that the return value from scanf() is the number of successful assignments; it does not count the conversion specifications that include *.

Datentypen

Type	**Keyword**	Bytes	Range
character	char	1	-128 .. 127
unsigned character	unsigned char	1	0 .. 255
integer	int	2	-32 768 .. 32 767
short integer	short	2	-32 768 .. 32 767
long integer	long	4	-2 147 483 648 .. 2 147 483 647
unsigned integer	unsigned int	2	0 .. 65 535
unsigned short integer	unsigned short	2	0 .. 65 535
unsigned long integer	unsigned long	4	0 .. 4 294 967 295
single-precision floating-point (7 Stellen)	float	4	1.17E-38 .. 3.4E38
double-precision floating-point (19 Stellen)	double	8	2.2E-308 .. 1.8E308

Zusammenfassung Tutorial

Bit, Byte

1 Byte = 8 Bit
100 MB (MegaBit) = 12,5 Megabyte (Mb)

Datentypen bei Deklaration

- char, int, float, double
- char = einzelne Zeichen, keine Zahlen
- int = Ganzzahlen
- float, double = Kommazahlen Variablen am besten immer gleich mit Werten deklarieren, damit keine Zufallswerte verwendet werden. int iZahl=0;

Konstanten

```
const int raeder = 4;
```

Der Compiler bringt eine Warnung, wenn die Variable noch einmal zugewiesen wird.

Variablenbenennung

1. Der Name darf nur aus Buchstaben, Ziffern und dem Unterstrich _ bestehen.
2. Das erste Zeichen muss ein Buchstabe oder der Unterstrich _ sein.
3. Groß- und Kleinschreibung ist relevant, d.h. „zahl“ und „Zahl“ ist nicht dasselbe.
4. Der Name darf kein Schlüsselwort sein.

Üblicherweise mit Kleinschreibung beginnen und Worttrennung über Großschreibung realisieren.

Inkrement

```
int a,  b=0;
  // Erst Zuweisung, dann Inkrement, a ist 0, b ist 1
  a = b++;
  // Erst Inkrement, dann Zuweisung, a ist 2, b ist 2
  a = ++b;
```

Ein- Ausgabe

Was rein geht, geht auch raus.

Variablen einlesen

```
float variable=32.5;
```

Ausgabe

```
printf("Laenge: %5.2f", variable);
```

Integer kann mit double ausgegeben werden. Kommazahlen sind auch möglich.

Zeichen einlesen

Einzelnes Zeichen

```
c = getchar();
```

Zahlen einlesen

```
scanf("%d",&alter);
```

Double in, double out.

Mit %d werden Ganzzahlen eingelesen, mit %f Kommazahlen. Alle Eingaben inklusive dem Enter kommen in den Puffer. Um das Enter abzufangen, könnte das enter in eine Variable &temp abgefangen werden.

Verzweigungen

If Else

```
int zahl=6;
if(zahl==5) {
        printf("fuenf\n");
}else {
    if(zahl==6) {
        printf("sechs\n");
    }else {
        printf("nicht fuenf und nicht sechs\n");
        }
}
```

Kurzfassung

Kommt nur eine Anweisung in den if Block, so könnte man auch die geschweiften Klammern weglassen.

```
  int zahl=6;
  if(zahl==5) printf("fuenf\n");
  else if(zahl==6) printf("sechs\n");
  else printf("nicht fuenf und nicht sechs\n");
```

Vergleichoperatoren

```
== Ist gleich \\
!= Ist nicht gleich \\
> Größer \\
>= Größer gleich
< Kleiner \\
<= Kleiner gleich \\
```

Logische Operatoren

```
    ! Negation
    && UND
    !! ODER
```

Switch case

```
  switch(a){
    case 1: printf("Bla");break;
    case 2: printf("bla2");break;
  }
```

While Schleife

```
while(bedingung i < 1000){
        printf("Bla");
        i++;
    }
```

For Schleife

```
int i;
```

```
  for(i=0; i<5; i++) {
          printf("Zahl %d\n", i+1);
  }
```

Do while

```
int alter;

  do {
      printf("\nBitte geben sie ihr Alter ein: ");
      scanf("%d", &alter);
  } while(alter <  5 || alter > 100);

  printf("Danke.\n");
```

Funktionen

Die Funktion muss vor dem Main positioniert werden, damit sie dann in der Main aufgerufen werden kann.

```
#include<stdio.h>

int addiere(int summand1, int summand2) {
    return (summand1 + summand2);
}

int main() {
    int summe = addiere(3, 7);
    printf("Summe von 3 und 7 ist %d\n", summe);
    return 0;
}
```

Funktionsprototypen

Diese kommen vor das Hauptprogramm, wobei der Funktionskörper dann an einer beliebigen Stelle im Script sein darf.

```
#include<stdio.h>

// Funktions-Prototypen
float eingabeZahl();
float multipliziere(float zahl1, float zahl2);
void ausgabeErgebnis(float ergebnis);

// Hauptprogramm
int main() {
```

Arrays

Arrays sind einzeilige Matrixen um viele Daten speichern zu können. Zugegriffen wird auf die einzelnen Werte mit dem Index.

Schleifen

Setzen von Werten mit Benutzereingabe

```
int punkte[5], i;
// Werte einlesen
for(i=0; i<5; i++) {
    printf("\nBitte geben sie eine Punktzahl ein (ganze Zahl): ");
    scanf("%d", &punkte[i]);
}

// Werte auslesen
for(i=0; i<5; i++) {
    printf("(Index %d) Punktzahl Aufgabe %d: %d\n", i, i+1, punkte[i]);
}
```

Initialisierung

Dazu die Werte eines Feldes einfach in geschweifte Klammern schreiben. Ist die Anzahl der Werte kleiner als die Feldgröße, werden die restlichen Werte auf 0 gesetzt. Ohne Größenangabe wird das Array die Größe durch die Anzahl der Initialisierungswerte bestimmt.

```
int i, punkte[5] = { 1, 3, 5, 7, 9 };

// Werte ausgeben
for(i=0; i<5; i++) {
    printf("Wert Index %d: %d\n", i, punkte[i]);
}
```

Zweidimensionale Felder

```
 int brett[8][8];
```

Für eine bessere Vorstellung kann man von

```
int brett[y][y];
```

annehmen, nicht wie gewohnt XY!

Initialisierung

Die Initialisierung zweidimensionaler Felder erfolgt nach unserer Definition der Achsen spaltenweise.

Mehrdimensionale Felder

Der Dimension sind keine Grenzen gesetzt, ein 3D Feld könnte wie folgt initialisiert werden

```
int cube[2][2][2];
```

Zeigerarithmetik

Der Zugriff erfolgt mittels Zeigern. Der Zugriff mit [] ist nur eine Möglichkeit, durch die im Hintergrund Zeiger verwendet werden.

Positionszeiger

Ein Positionszeiger kann immer auf einen bestimmten Wert im Array zeigen. Die Adresse des ersten Elements erhalten wir durch den bloßen Feldnamen.

```
int punkte[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Positionszeiger
int *pos;

// Position auf erstes Element setzen (punkte[0])
pos = punkte;
printf("(punkte[0]) Wert pos: %d\n", *pos);

// Position auf naechstes Element setzen (punkte[1])
pos++;
printf("(punkte[1]) Wert pos: %d\n", *pos);

// Position auf 5. Element setzen (punkte[4])
pos = punkte + 4;
printf("(punkte[4]) Wert pos: %d\n", *pos);

// Position auf vorheriges Element setzen (punkte[3])
pos--;
printf("(punkte[3]) Wert pos: %d\n", *pos);
```

ermittlung der Index-Nummer

```
int punkte[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```c
// Positionszeiger, Setzen auf 6.Element
int *pos = punkte + 5;

// Index berechnen
int index = pos - punkte;

printf("Index: %d\n", index);
printf("Wert pos / punkte[index]: %d\n", punkte[index]);
```

Mehrdimensional

```c
// Array und Spaltengrenze setzen
int brett[8][8]={ 0 }, Y_Max=8;

// Startposition
int *posStart = &brett[0][0];

// Testwert setzen
brett[2][4] = 7;

// Beschaffung Testwert mit Zeigerarithmetik
int *pos;
pos = posStart + (y * Y_Max) + x;

printf("%d\n", *pos );
```

Die entscheidende Programmzeile ist die, in der der Positionszeiger pos gesetzt wird. Seine Basis ist die Startadresse des Arrays. Benötigt wird der Wert in Zeile 2 und Spalte 4, Index [2][4]. Um dorthin zu kommen, müssen wir uns erst zeilenweise und dann spaltenweise vorarbeiten. Also setzen wir den Zeiger auf das erste Element in Zeile 2. Eine Zeile hat 8 Spalten-Elemente, also müssen wir den Zeiger 2 * 8 Elemente weiterrutschen. Zum Schluss addieren wir noch die 4, um in die vierte Spalte zu springen.