8.1 Sonstiges/Variable Argumentlisten

8.1.1 Situation

Gesucht ist eine Funktion `strmulticat()`, die beliebig viele Strings an den ersten String anhängen kann. Beispiel:

```
1  char gericht[800]="Kartoffelsalat";
2  strmulticat(gericht, " mit Ketchup", " und gut durchgebraten");
```

Das Problem liegt darin, dass die Anzahl und die Typen der Parameter in C festliegen: Der Compiler vergleicht, ob bei einem Aufruf die aktuellen Parameter des Aufrufs mit den formalen Parametern der Parameterliste im Funktionskopf übereinstimmen. Falls nicht, gibt es eine Warnung oder eine Fehlermeldung.

Andererseits gibt es Funktionen wie `printf()` und `scanf()`, bei denen nach dem Formatstring beliebig viele Parameter und beliebige Typen erlaubt sind. Ein Aufruf der Manual-Page zu `printf()` gibt Auskunft:

Terminal

```
schueler@debian964:~$ man 3 printf
#include <stdio.h>
       int printf(const char *format, ...);
```

Mit den drei Punkten wird also angegeben, dass hier beliebige Parameter folgen können. Gut, nur wie kommt man in der Funktion an Parameter heran? Es sind ja keine Namen vorhanden!

8.1.2 Die Makros von stdarg.h

In `src/multisum.c` ist ein Beispiel:

```
1  #include <stdarg.h>
2  #include <stdio.h>
3  double multisum(int, ...);
4
5  int main(void)
6  {
7      double x;
8      x=multisum(4, 228.3, 240.4, 232.7, 227.3);
9      printf("%f\n", x);
10     return 0;
11 }
12
13 double multisum(int anzahl, ...)
14 {
15     va_list liste;
16     int lauf;
17     double summand, summe;
18
19     va_start(liste, anzahl);
20     summe=0.0;
21     for(lauf=0; lauf<anzahl; ++lauf)
22     {
23         summand = va_arg(liste, double);
24         summe += summand;
25     }
26     va_end(liste);
27     return summe;
```

```
28 }
```

Zeile 1 Es wird `<stdarg.h>` eingebunden. Darin sind ausnahmsweise keine Funktionen vereinbart, sondern Makros (das sind `#define`-Anweisungen mit Parametern).

Zeile 3 Hier ist der Prototyp. Sein zweites Argument besteht aus drei Punkten, das bedeutet: Hier sind beliebig viele weitere Argumente möglich.

Zeile 8 Aufruf von `multisum()` mit mehreren Argumenten.

Zeile 13 Hier beginnt die Definition von `multisum()`.

Zeile 15 Mit `liste` erhält man einen Merker für die Parameterliste. `va_list` ist eine Makrodefinition, möglicherweise die Definition eines Zeigers.

Zeile 19 Auch `va_start()` ist ein Makro. Es initialisiert `va_list` mit Hilfe des Namens des *letzten festen* Parameters.

Zeile 23 Das Makro `va_arg()` holt aus der Parameterliste den nächsten Parameter, und zwar in der Annahme, dass es sich um eine `double`-Variable handelt. Gleichzeitig wird die Liste so verändert, dass beim nächsten Aufruf von `va_arg()` die nächstfolgende Variable geholt werden kann.

Zeile 26 Das Makro `va_end()` schließt die Liste.

Wären `va_start()`, `va_arg()` und `va_end()` Funktionen und keine Makros, dann müssten sie jeweils die Adresse von `liste` erhalten.

8.1.3 Wozu ist `vprintf()` da?

In der Manual-Page zu `printf()` findet man auch eine Funktion mit dem Namen `vprintf()`:

Terminal

```
schueler@debian964:~$ man 3 printf
#include <stdio.h>
#include <stdarg.h>
       int vprintf(const char *format, va_list ap);
```

Hier sind keine drei Punkte in der Parameterliste angegeben, aber eine Variable vom Typ `va_list`. Wie wird `vprintf()` benutzt? In `src/myprintf.c` sieht man ein Beispiel:

```
1  #include <stdarg.h>
2  #include <stdio.h>
3  void myprintf(const char *format, ...)
4  {
5      va_list liste;
6      va_start(liste, format);
7      printf("Jetzt kommt's:\n");
8      vprintf(format, liste);
9      printf("Das war's.\n");
10     va_end(liste);
11 }
12 int main(void)
13 {
14     myprintf("Zahlen: %f, %i\n", 3.14, 12345);
15     return 0;
16 }
```

Man kann also mit `vprintf()` eigene Funktionen schreiben, die die Funktionalität von `printf()` nutzen und erweitern und die ebenso wie `printf()` beliebig viele Parameter erlauben.