# Funktionen mit variabler Argumentliste

- ellipse, Ursprung im Griechischen,  Satz mit Auslassungen
- In der Programmierung: Funktionen wie printf, scanf ….

```
int printf(const char * format, ...);
```

- Benötigtes Include:

```
#include <stdarg.h>
```

- Stellt nachfolgende Macros bereit:

```
va_start
va_arg
va_end
```

- Alle drei Macros verwenden einen Argumentpointer

```
va_list ap;
```

Frei wählbarer Bezeichner,
üblicher Weise ap

# Wie man damit hantiert

- Mit va_start wird ap so eingestellt, dass er auf den angegebenen Parameter im Stack zeigt.

```
va_start(ap,fmt);
```

Parameter vor ...

- Mit va_arg wird der Argumentpointer ap auf den jeweils nächsten Parameter eingestellt und dieser zurückgegeben. Der angegebene Typ (hier int) muss dabei unbedingt mit dem Typ des tatsächlich angegebenen Parameters übereinstimmen.

```
ival=va_arg(ap,int);
```

Typ des Parameters, hier int

- Mit aufeinanderfolgenden Aufrufen von va_arg kann man sich nun durch die Parameterliste im Stack „hangeln“.

- Funktionsweise in vereinfachter Form:

```
(typ) (((char*) ap)+=sizeof(typ)),
*((typ*) (((char*)ap)-sizeof(typ)))
```

# Wie's funktioniert

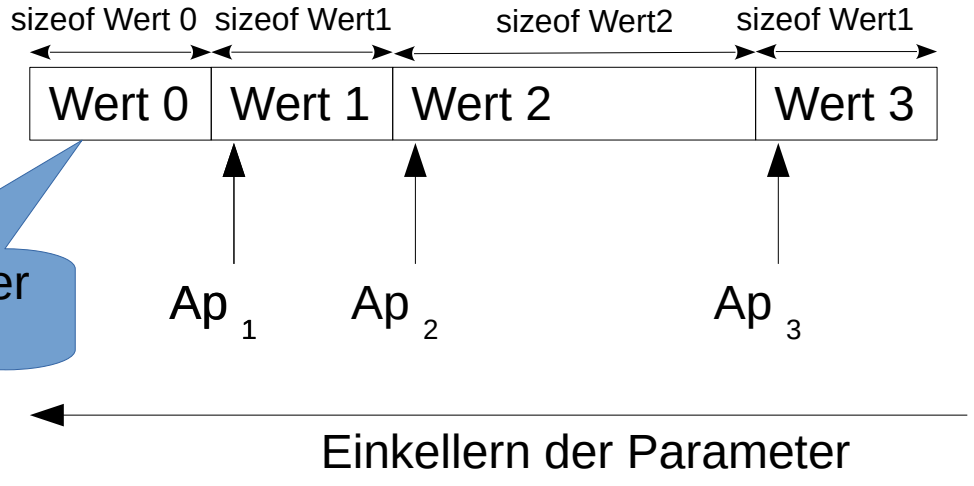
Teil 1:
Erhöhen des Argumentpointers

```
#define va_arg(AP, TYPE)                                        \
 (AP = (__gnuc_va_list) ((char *) (AP) + __va_rounded_size (TYPE)),  \
  *((TYPE *) (void *) ((char *) (AP)                            \
       - ((sizeof (TYPE) < __va_rounded_size (char)  \
   ? sizeof (TYPE) : __va_rounded_size (TYPE))))))
```

Kommaoperator

Teil 2:
Bereitstellung des Wertes von der vorhergehenden Position des ap

Fester Prameter vor …



# Ein paar Ausgabefunktionen

```
6   void iputdec(int i)
7   {
8     if (i>=10) iputdec(i/10);
9     putchar(i%10+'0');
10  }
11
12  void iputhex(unsigned long d)
13  {
14    if (d>=0x10) iputhex(d/0x10);
15    putchar(d%0x10>9?d%16+'A'-10:d%16+'0');
16  }
17
18  void putint(int i)
19  {
20    if (i<0)putchar('-'),iputdec(-i);
21    else                 iputdec( i);
22  }
```

```
24  void putdouble(double d)
25  {
26    int i;
27    if(d<0)
28      {putchar('-');d=-d;}
29    i=d;
30    putint(i);putchar('.');
31    d-=i;
32    d*=1000;
33    i=d;
34    putint(i);
35  }
```

```
37  void myPrintf(char*fmt,...)
38  {
39    va_list ap;
40    char   *p,
41           *sval,
42            cval;
43    int     ival;
44    double  dval;
45
46    va_start(ap,fmt);
47    for (p=fmt;*p;p++)
48    {
49      if(*p !='%')
50      {
51        putchar(*p);
52        continue;
53      }
54      switch (*++p)
55      {
56        case 'd':ival=va_arg(ap,int);
57                 putint(ival);          break;
58        case 'x':ival=va_arg(ap,int);
59                 iputhex(ival);         break;
60        case 'f': dval=va_arg(ap,double);
61                 putdouble(dval);       break;
62        case 's': for(sval=va_arg(ap,char*); *sval!='\0'; sval++)
63                 putchar(*sval);                 break;
64        case 'c': cval=va_arg(ap,int);
65                          putchar(cval);         break;
66        default : putchar(*p);                   break;
67      }
68    }
69    va_end (ap);
70  }
```

# Die Funktion myPrintf

# Main-Funktion dazu

```
72  int main()
73  {
74    myPrintf("INT %d,%x Double %f, Zeichenkette %s, Zeichen% c\n"
75  »       »       »       ,-123456,-123456,(double)4.12345,"TEST",'A');
76
77    return 0;
78  }
79
```