

```

#define SUM(a,b) a+b
#define MUL(a,b) a*b
.
.
.
int i, j, k, l;
i=5;
j=2;
k=3;

int sum(int a, int b)
{return a+b;}

int mul(int a, int b)
{return a*b;}

...
i=5;
j=2;
k=3;
l=mul(sum(i,j),k);          (21)

l=MUL(SUM(i,j),k);
↳ l=i+(j*k);

```

das Makro muss geklammert werden, damit der Ausdruck richtig berechnet wird:

```

#define SUM(a,b) ((a)+(b))
#define MUL(a,b) ((a)*(b))

```

der Ausdruck wird dann wie folgt vom Präprozessor ausgelegt:

```
l=(((i)+(j)))*(k);
```

die folgende Abfrage (a<b ?) ist ein ternärer Operand:

```

#define MIN(a,b) ((a)<(b) ? (a) : (b))
...
int i=5;
int j=4;
int k=MIN(i,j);
↳ int k=((i)<(j) ? (i) : (j));

int min(int a, int b)
{return a<b ? a : b;}

...
int i=5;
int j=4;
int k=min(i,j);

```

werden die letzten Zeilen geändert in:

```
int k=MIN(i++,j++)          int k=min(i++,j++);
```

so erkennt der Präprozessor dies als:

```
int k=((i++)<(j++)?(i++):(j++));
int k=min(i++,j++);
```

Die Ergebnisse daraus lauten:

```

i= 6          i= 6
j= 6          j= 5
k= 5          k= 4

```

Der Fehler lässt sich umgehen mit:

```

int k=MIN(i,j);
i++;
j++;

inline (vor der Funktion, nur unter C++)

```

Es gibt funktionsähnliche Makros, man kann sie auch verwenden, aber man sollte aufpassen auf "Seiteneffekte" und auf die Klammerung. Unter C++ sollte man keine funktionsähnlichen Makros verwenden.

```
__LINE__           Zeilennummer
__FILE__           Quelltextdatei
printf("ich bin in %d\ in %sn", __LINE__, __FILE__);

#line 36           __LINE__ wird auf Zeile 36 gesetzt.
```

```
printf("ich bin in %d\n", __LINE__
```

gibt ich bin in 37 **aus**.

```
int i;

int main()
{
    while(...)
    {
        for (i=0; i<10;i++)
        { ... }
    }
}
```

Die Variable `i` sollte im obigen Beispiel innerhalb der `main`-Methode definiert werden, da `i` sonst als globale Variable behandelt wird. Allerdings wird `i` ja nur als Iterationsvariable für die Schleife benötigt.

Bedingtes Kompilieren

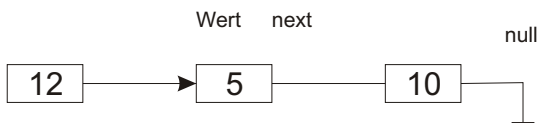
```
#define KUNDE MEIER

#if defined KUNDE
    ...
#else
#error "Kunde???"
#endif
```

mit `#if defined` kann man überprüfen, ob ein bestimmtes Makro definiert wurde.

```
#ifdef __unix           // __linux
    ...
#elif defined _WIN32
    ...
#else
#error "Wie soll ich...?"
#endif
```

Strukturen



```
struct LE
{
    int wert;
    struct LE * next;
};

void up(int * wert)
{
    (* wert)++;
}

int i=5;
up(&i);
```

```
struct LE * lanf = (...) malloc (...)
(* lanf).wert = 12;
(* lanf).next = (struct LE*) malloc (1 * sizeof(struct LE));
>(* lanf).next.wert = 5;
>(* lanf).next.next = (...) malloc(...);
>(* lanf).next.next.wert = 10;
>(* lanf).next.next.next = NULL;
```

analoge Schreibweise:

```
lanf->next->next->next = NULL;
```

```
int wert1, wert2;
char name[100];
```

```
printf ("Bitte 2 Werte: \n");
scanf ("%d%d", &wert1, &wert2);
scanf ("%s%d", &name[0], &wert1); => scanf ("%s%d", name, &wert1);
```

Man kann auch überprüfen, ob die Anzahl der Direktiven (hier %s%d, also zwei) vom Eingabe richtig erfüllt wurden, also ein eingegebener String gefolgt von einer Ganzzahl:

```
if (scanf ("%s%d", name, &wert1) == 2)
{...}
```