Funktionen: Kleine Unterprogramme, die Teilprobleme lösen.

Deklaration:

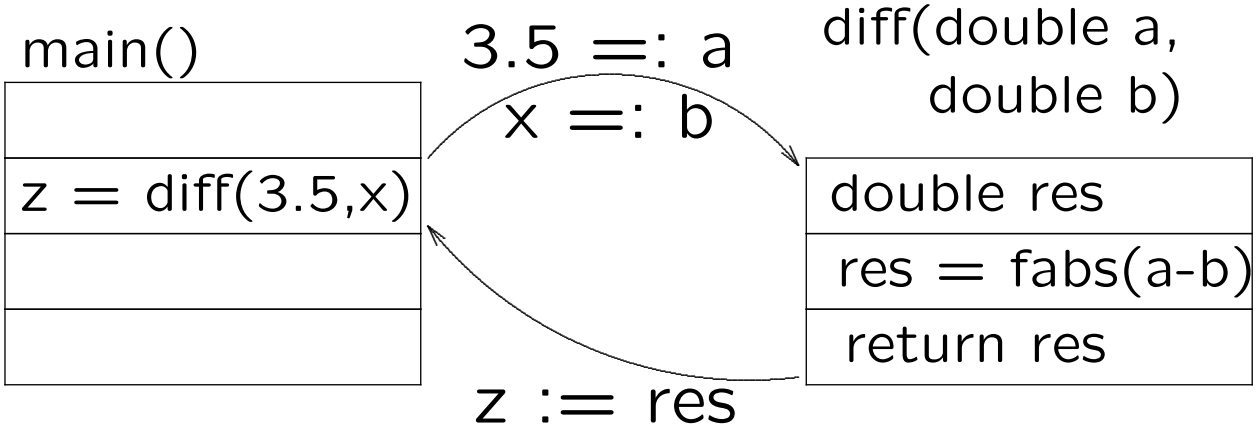
```
typ funk_name(PARAMETERLISTE) {
/*  Anweisungen  */
}
```

`typ` Typ des Rückgabewertes (Ergebnis)
`funk_name` Bezeichner der Funktion

`PARAMETERLISTE`
Liste der übergebenen Parameter mit Typ und Bezeichner (kann leer sein)

`Anweisungen`
Befehlsfolge, die Ergebnis berechnet und mittels `return`-Anweisung zurück gibt.

Parameter: Schnittstelle zw. Funktionen



Argumente: 3.5 und x
.

Parameter a und b
Rückgabewert res

Kopie der Argumente wird an Parameter übergeben (*call by value*).

Rückgabewert repräsentiert das Ergebnis der Funktion.

Parameter verändern:

```
int main() { ...
    x = 5.0; y = 6.0;
    z = diff(x,y); ...
}

double diff(double a, double b) {
    a = fabs(a-b);
    return a;
}
```

`x` vor Funktionsaufruf: 5.0
`a` beim Funktionsaufruf: 5.0
`a` nach Funktionsberechnung: 1.0
`x` nach Funktionsaufruf: 5.0

Argument `x` bleibt unverändert, da nur Kopie überschrieben wird.

Argumente verändern:

Verwendung bei mehreren Ergebnisvariablen und großen Datenstrukturen.

Zeiger: `double *p_x;` (Speicheradresse)

Wert hinter Zeiger: `*p_x` `x = *p_x`
(dereferenzieren mit $*$-Operator)

Adresse einer Variable: `&x` `p_x = &x`

Übergabe einer **Kopie der Adresse** einer Variable als Parameter an eine Funktion ermöglicht Manipulation (*call by reference*).

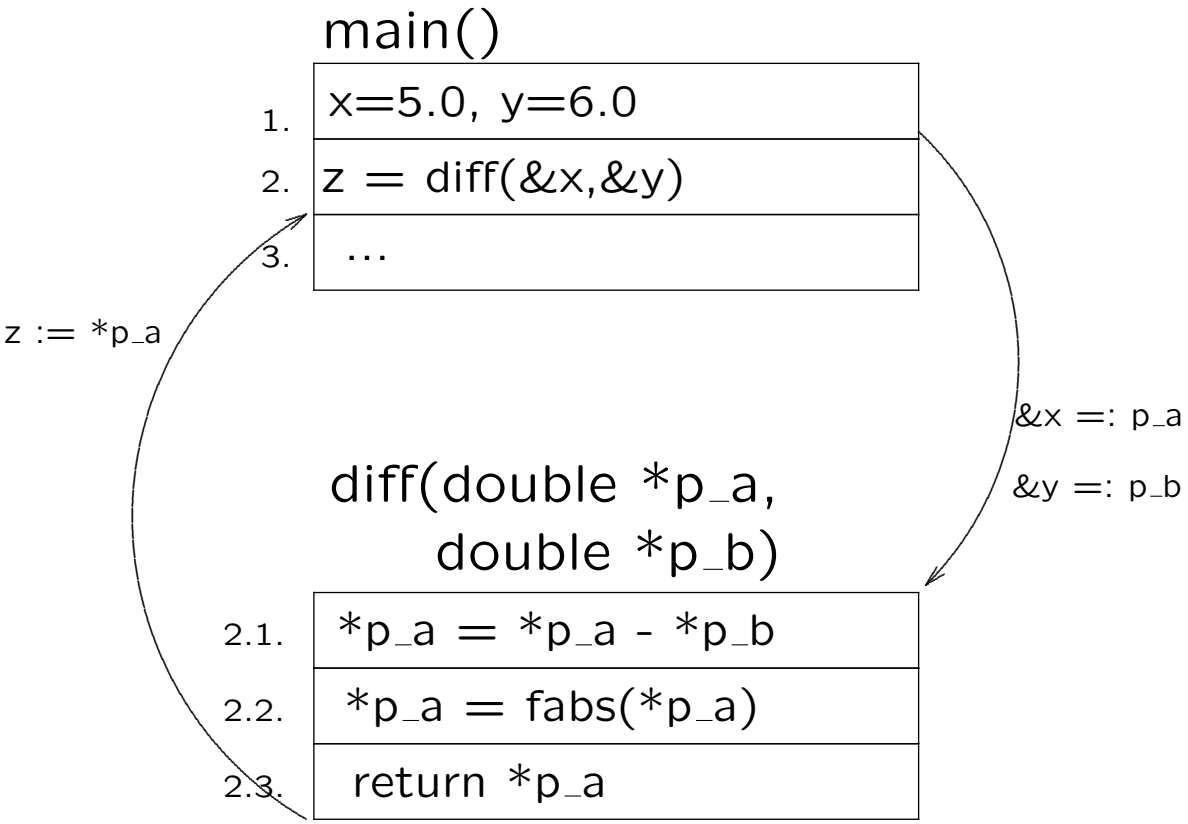
Referenzen übergeben:

```
int main() { ...
    x = 5.0; y = 6.0;
    z = diff(&x,&y); ...
}

double diff(double *p_a, double *p_b) {
    *p_a = *p_a - *p_b;
    *p_a = fabs(*p_a);
    return *p_a;
}
```

x vor Funktionsaufruf: 5.0
*p_a beim Funktionsaufruf: 5.0
*p_a nach Funktionsberechnung: 1.0
x nach Funktionsaufruf: 1.0

x wird verändert, da der Speicherbereich von x durch *p_a überschrieben wird.



	2.	2.1.	2.2.	2.3.	3.	
						0x1F20
p_b		0x1F08	0x1F08	0x1F08		0x1F1C
p_a		0x1F00	0x1F00	0x1F00		0x1F18
z					1.0	0x1F10
y	6.0	6.0	6.0	6.0	6.0	0x1F08
x	5.0	5.0	-1.0	1.0	1.0	0x1F00