§2 Funktionszeiger und Behälter

Leitideen: *Die Vereinbarung von Funktionen und Funktionszeigern soll ihrem Aufruf ähneln und systematisch lesbar sein („von innen nach außen“).*

Dem Konzept der Funktionszeiger liegt die modellhafte Vorstellung zugrunde, dass für den Aufruf einer Funktion zumindest die Anfangsadresse des Funktionscodes, die Funktionsargumente und der Ergebnistyp erforderlich sind.

Funktionen stehen zu Funktionszeigern ungefähr im gleichen Verhältnis wie C-Vektoren zu Zeigern und es finden automatisch entsprechende Umwandlungen statt.

Funktionsaufrufe können auch direkt mit einem Funktionszeiger erfolgen.

Funktionszeiger sind manchmal zulässig, wo Funktionen nicht erlaubt sind (z.B. als Rückgabewerte, in C-Vektoren und STL-Behältern)

§2 Funktionszeiger - Themenübersicht

- Modellhafte Implementierung des Funktionsaufrufs
- Speicherlayout (Veranschaulichung von Funktionszeigern)
- Vereinbarungssyntax
- Funktion `signal`
- Umwandlung von Funktionsnamen in Funktionszeiger
- Sehnentrapezregel - Erinnerung
- Sehnentrapezregel - Funktionszeigertabellen
- Auswertung erweiterter arithmetischer Ausdrücke I,II

Funktionsaufruf - modellhafte Implementierung

Aufruf:

- Speicherung auf Stack (in neuem Stacksegment):
  - übergebene Argumente
  - lokale Variablen der Funktion
  - Rücksprungadresse
- Sprung zum Codeanfang der aufgerufenen Funktion (Startadresse!)

Beendigung:

- Übergabe des Funktionswerts
- Rücksprung
- Freigabe des Stacksegments

Abstraktion: Funktionszeiger (Startadresse mit Typ)

Beispiel für Funktionsaufruf:

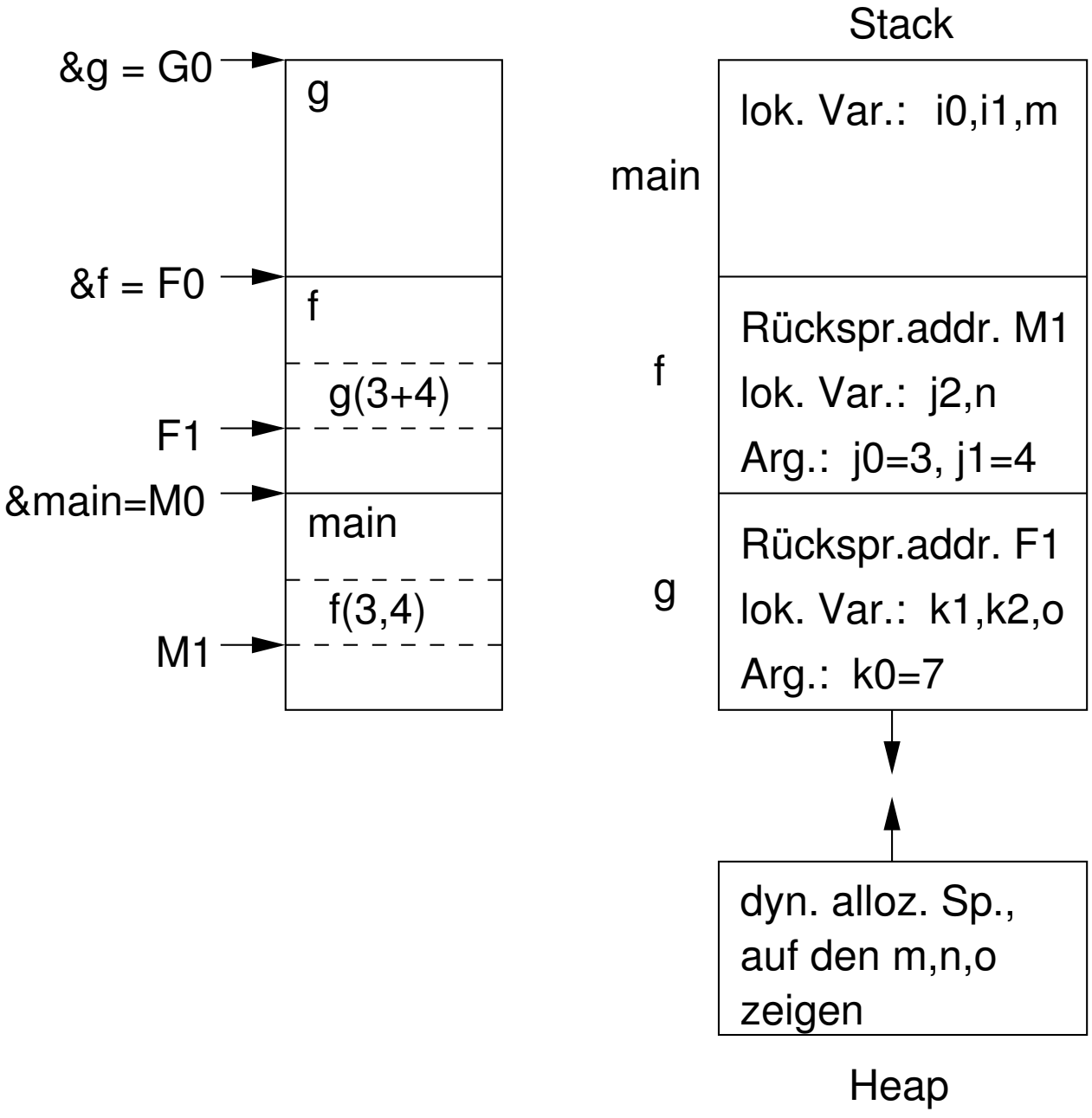
```
g(...)
f(...)        ruft g auf
main(...)     ruft f auf
```

```
void g(int k0)
{
  int k1,k2;
  char *o = new char[256];
  return;
}

void f(int j0,int j1)
{
  int j2;
  char *n = new char[256];
  g(j0+j1);
  return;
}

int main(int argc, char *argv[])
{
  int i0=3,i1=4;
  char *m = new char[256];
  f(i0,i1);
  return 0;
}
```

Speicherlayout - Modell



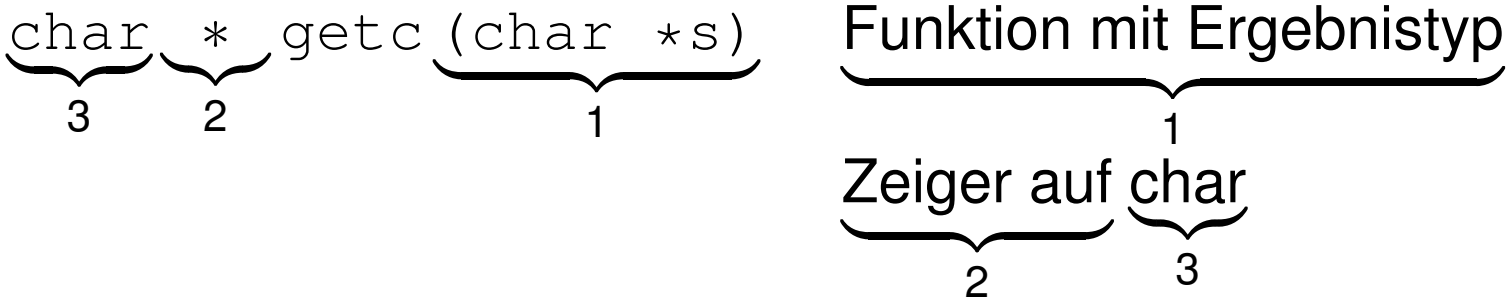
Funktionen - Vereinbarungssyntax

`f(x)`	Funktionswert, z.B. double
`double f(double x)`	Funktion mit Ergebnistyp double

Ziel der Vereinbarungssyntax: Aussehen wie Ausdrücke

`char *getc(char *s)`	äquivalent zu
`char *(getc(char *s))`	Inhalt des Funktionswerts von getc ist ein char, d.h. getc liefert Zeiger auf char

Mechanismus: Lesen von innen nach außen



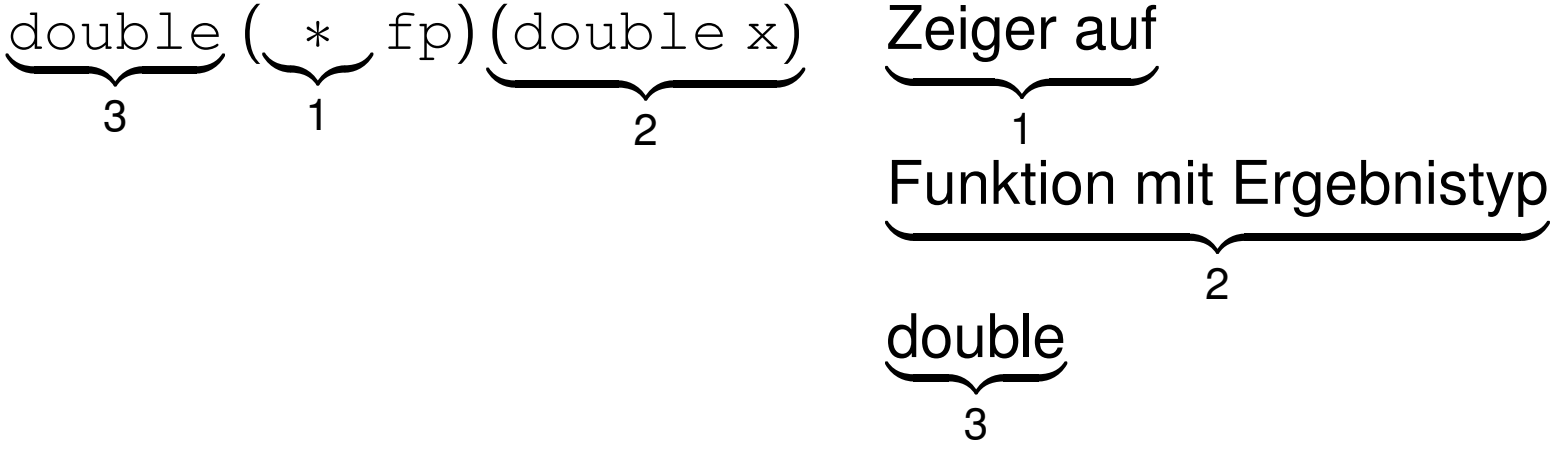
Funktionen - Vereinbarungssyntax II

Ziel der Vereinbarungssyntax: Aussehen wie Ausdrücke

```
double (*fp)(double x)
```

Inhalt von fp ist eine Funktion mit Ergebnistyp double, d.h.
fp ist Zeiger auf eine Funktion mit Ergebnistyp double

Mechanismus: Lesen von innen nach außen



[∗] Funktion signal als Bsp. für Vereinbarungssyntax

Zweck: Registriert Funktion, die beim Eintreffen eines Signals ausgeführt wird

```
void (*signal(int sig, void (*func)(int)))(int)
```

Signalbehandlungsfunktion:

```
void ( * func)(int)
```

3 1 2

Zeiger auf (1) Funktion mit Ergebnistyp (2) void (3)

Funktion signal:

```
void ( * signal (int sig, void (*func)(int))) (int)
```

4 2 1 3

Funktion mit Ergebnistyp (1)

Zeiger auf (2) Funktion mit Ergebnistyp (3) void (4)

Fkt.wert: Zeiger auf die zuvor für das Signal registrierte Fkt.

Umwandlung von Funktionsnamen in Funktionszeiger

- Automatische Typumw. bei C-Vektoren in Ausdrücken:
  C-Vektor $\rightarrow$ Zeiger (bereits behandelt)
  Ausnahmen: `&a`, `sizeof a`, `a=`
- Automatische Typumw. bei Funktionen in Ausdrücken:
  Funktion $\rightarrow$ Funktionszeiger
  Ausnahmen: `&f`, `f()`, `f=`
- Ähnlich auch bei Parameterübergabe von Funktionen!
- Vereinfachung seit ANSI C: Funktionsaufruf direkt mit Funktionszeiger möglich

  Bsp.: `fp=&f; fp(x); // korrekter Fktaufruf.`

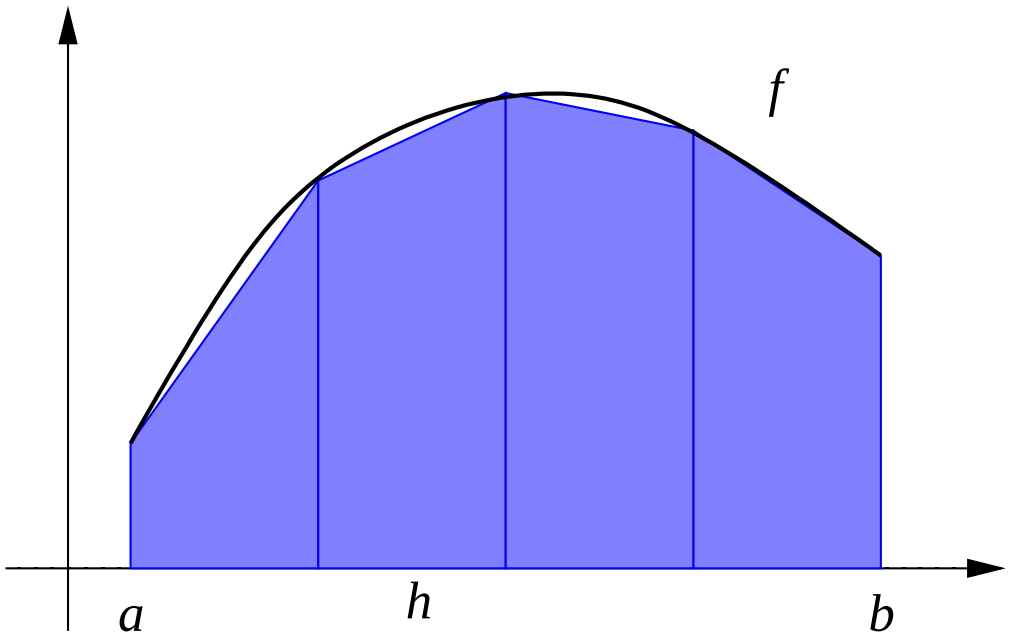
Caveat

- `f();` Funktionsaufruf bei leerer Argumentliste
  `f;` kein Funktionsaufruf (Umw. in Funktionszeiger)

Sehnentrapezregel I - Erinnerung

Näherung für $\int_a^b f(x)dx$: $\quad h\sum_{i=0}^{n} w_i f(x_i)$

mit $h = \frac{b-a}{n}$, $x_i = a + ih$ $(i=0,\ldots,n)$, $w_i = \begin{cases} \frac{1}{2} & i=0,n \\ 1 & i=1,\ldots,n-1 \end{cases}$



Sehnentrapezregel II - Funktionszeigertabellen

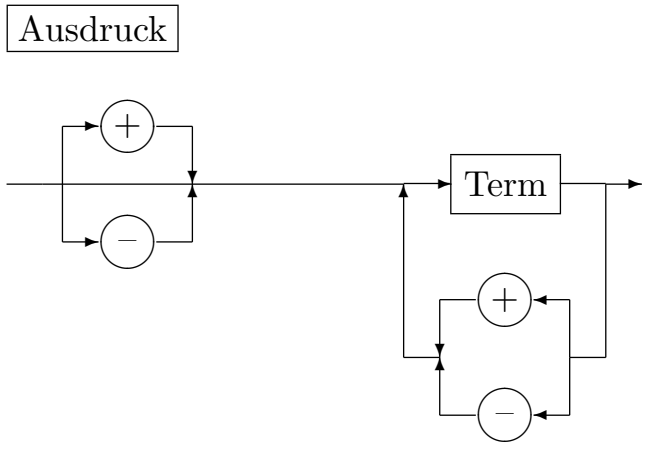
- Funktionszeiger als Komponenten von C-Vektoren und STL-Behältern zulässig. (Funktionen nicht!)
- Funktionszeigertyp als Template-Parameter zulässig.
  Verwendung hier: Funktionen, Quadraturformeln
- Direkte Angabe möglich, mit typedef wohl verständlicher
- Statt STL-Behälter ($\rightarrow$ *Inf.bl. 12*) auch C-Vektoren möglich:
- ```
  double (*funcptrvec[])(double)
    4      2            1   3
  ```

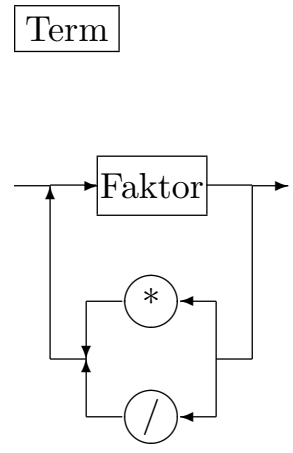
  Vektor (1) von Zeigern (2) auf Funktionen (3) mit Erg.typ double (4)
- ```
  double (*quadptrvec[])(double,double,int,
    4      2            1   3        double(double))
  ```

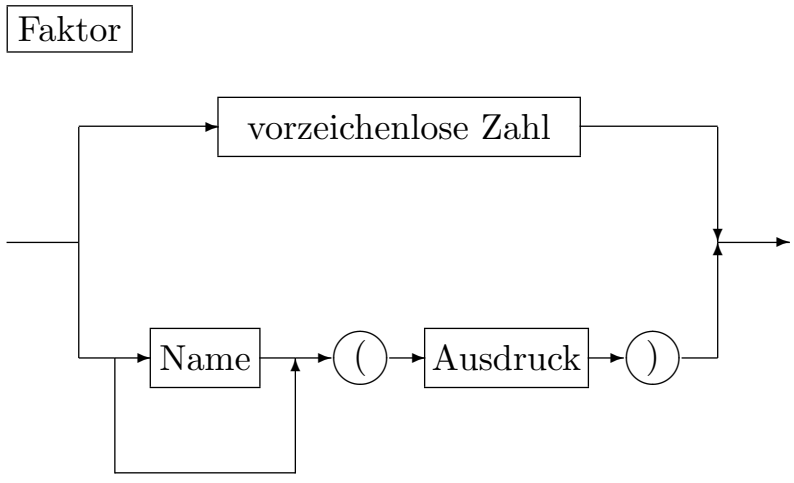
  Vektor (1) von Zeigern (2) auf Funktionen (3) mit Erg.typ double (4)

Auswertung erweiterter arithmetischer Ausdrücke I

Syntaxdiagramme







Auswertung erweiterter arithmetischer Ausdrücke II

- Kapselung in der Klasse `Ausdruck`: Nur die Konstruktoren und die Funktion `auswerten` sind außerhalb der Klasse zugreifbar
- Keine Forward-Deklarationen erforderlich, weil Komp.fkt. und befreundete Funktionen sich gegenseitig benutzen dürfen
- `WertAusdruck` und `WertTerm` gegenüber Prog.I kaum verändert (`cin`→`ein`).
- `WertTerm` benutzt die Map `fmap` zur Zuordnung *Funktionsname*→*Funktionszeiger*.
- String-Parameter in Funktion `fehler` zur besseren Fehlerlokalisierung.
- Konstruktor speichert den auszuwertenden String und initialisiert den Stringstream *ein*.
- Beim Erstaufruf des Konstruktors wird mit `mapinit` die *statische* Map `fmap` initialisiert.