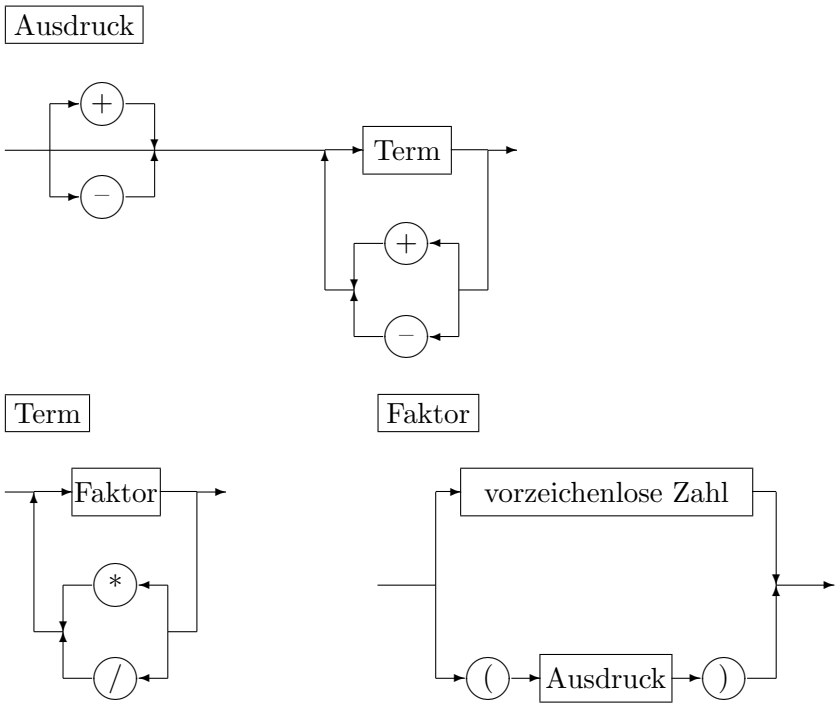
Beispiel zu wechselseitig rekursiven Funktionen

- **Auswertung arithmetischer Ausdrücke**




```
/* Auswertung einfacher arithmetischer Ausdruecke
   Zwischenraum ist unzulaessig
   Fehlerbehandlung erfolgt nur sehr rudimentaer */

#include <iostream>
#include <iomanip>
#include <string>
#include <cctype>
#include <cstdlib>

using namespace std;

double WertAusdruck();
double WertTerm();
double WertFaktor();
void fehler();

int main()
{
  double ergebnis;
  cout << "Arithm. Ausdruck: ";
  ergebnis = WertAusdruck();
  if (cin.peek()=='\n')    // Aufhoeren am Zeilenende
    cout << "Ergebnis: " << ergebnis << endl;
  else
    fehler();
  return 0;
}
```

```
double WertAusdruck()
{
  double summe; char z;
  z = cin.peek();
  switch(z) {
    case '+': cin >> z; summe =  WertTerm(); break;
    case '-': cin >> z; summe = -WertTerm(); break;
    default : summe =  WertTerm(); break;
  }
  z = cin.peek();
  while( z=='+' || z=='-' ) {
    switch(z) {
      case '+': cin >> z; summe += WertTerm(); break;
      case '-': cin >> z; summe -= WertTerm(); break;
    }
    z = cin.peek();
  }
  return summe;
}

double WertTerm()
{
  double produkt = WertFaktor(); char z;
  z = cin.peek();
  while( z=='*' || z=='/' ) {
    switch(z) {
      case '*': cin >> z; produkt *= WertFaktor(); break;
      case '/': cin >> z; produkt /= WertFaktor(); break;
    }
    z = cin.peek();
  }
  return produkt;
}

double WertFaktor()
{
  double wert; char z;
  z = cin.peek();
  if ( z=='(' ) {
    cin >> z; wert = WertAusdruck();
    z = cin.peek(); if ( z!=')' ) fehler(); cin >> z;
  }
  else if (isdigit(z)) {
    cin >> wert;
  } else {
    fehler();
  }
  return wert;
}

void fehler()
{
  cout << "Fehler aufgetreten."  << endl; exit(1);
}
```

Zeiger und Referenzen

Zeiger

Zeigertypen sind Datentypen, deren Werte Adressen von Objekten eines festen Datentyps sind. Durch den Zeigertyp wird somit festgelegt, welcher Datentyp mit einer Adresse verbunden ist.

Der Adreßoperator `&` liefert die Adresse (mit Typinformation) eines Datenobjekts (z.B. einer Variablen); umgekehrt liefert der Inhaltsoperator `*` das Datenobjekt (z.B. eine Variable), auf das die Adresse zeigt.

0 bedeutet Verweis auf *kein* Datenobjekt.

Zeigerwerte können mit 0 verglichen werden.

Unzulässig ist die Anwendung des Inhaltsoperators auf eine Zeigervariable mit Wert 0 oder auf eine undefinierte Zeigervariable.

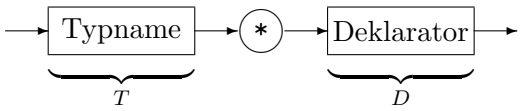
Vorsicht: Die Syntax für die Initialisierung und für die Zuweisung von Zeigern ist unterschiedlich.

Beispiel:

```
int i=5,j,*ip,*jp=0; /* bewirkt jp=0 ! */
ip=&i;
j=*ip;
jp=ip;
*jp=6;
```

Vorsicht: `j=5/*ip` [Kommentaranfang] im Unterschied zu `j=5/ *ip` .

Vereinbarungssyntax



Falls $T\ D$ einen Namen als "Datenstruktur aus T" vereinbart, dann vereinbart $T\ *D$ denselben Namen als "Datenstruktur aus Zeiger auf T".

Diese Definition bewirkt, dass der Datentyp aus dem Deklarator durch Lesen von innen nach außen bestimmt werden kann.

Beispiel:

`double *a[N];`	N-Vektor aus Zeigern auf double
`double (*b)[N];`	Zeiger auf N-Vektor von double

Mathematische Standardfunktionen mit Zeigerparametern (`cmath`)

In C gibt es *keine* Referenzparameter, an ihrer Stelle kommen Zeiger zum Einsatz. Nicht für alle aus C übernommenen Standardfunktionen gibt es Varianten mit Referenzparametern.

Funktionskopf	*Bedeutung*
`double frexp(double x, int *np)`	Liefert als Funktionswert y aus der Zerlegung $x = y \cdot 2^n$ mit $\frac{1}{2} \leq \|y\| < 1$ und n ganzzahlig, falls $x \neq 0$, und speichert n in dem Speicherplatz, auf den *np* zeigt.
`double modf(double x, double *np)`	Zerlegt $x = n + r$ mit n ganzzahlig, $\|n\| \leq \|x\|$ und $\|r\| < 1$. Liefert als Funktionswert r und legt n als `double`-Zahl in dem Speicherplatz ab, auf den *np* zeigt.

Es gibt gleichnamige Funktionen, bei der Ergebnis- und Parametertyp jeweils durch `float` oder durch `long double` ersetzt sind.

Beispiel: *Zu $x \in \mathbb{R}$ die Zerlegung $x = y \cdot 2^n$ mit $\frac{1}{2} \leq \|y\| < 1$ und $n \in \mathbb{Z}$ bestimmen*

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double x,y;
  int n;

  cout << "x: ";
  cin  >> x;

  // x in y*2^n zerlegen:
  y = frexp(x,&n);

  cout << "x=" << x << " y=" << y << " n=" << n << endl;

  return 0;
}
```

Ausgabe für $x = 6$: `x=6 y=0.75 n=3`