8. Übung

1 Rekursive Funktionen
2 Felder
3 Übungen
4 Lösungen

1. Funktionen

Nach Variablen, Schleifen und Ausgaben jetzt ein neues Konzept:

Funktionen

Teile von Algorithmen extra beschreiben (außerhalb von main)!

Vorteile:

- Besser Lesbarkeit
- Wiederverwendbarkeit
- Leichter Wartbarkeit

Generelle Syntax:

```
Rückgabetyp Funktionsname(Parameterliste) { Anweisungen }
```

1 Rekursive Funktionen

Ein rekursive Funktion in der Programmierung ist eine Funktion, die sich während der Abarbeitung selber aufruft. Sie führt nur dann zum Ziel, wenn es eine wohldefinierte Abbruchbedingung gibt.

Beispiel 1 – Berechnung der Fakultät - 2 Lösungswege

- *N! – Multipliziere alle Zahlen von 1 bis N miteinander – (Iteration)*

- *N! = N * (N-1)! – Wenn die Fakultät von N-1 bekannt, multipliziere diese mit N (Rekursion)*

- *Vorsicht bei Aufgabenstellungen mit linker Seite und Summe:*
  *(N+1)! = (N+1) * N! – Wenn die Fakultät von N bekannt, multipliziere diese mit N+1*

1 Rekursive Funktionen

Ein rekursive Funktion in der Programmierung ist eine Funktion, die sich während der Abarbeitung selber aufruft. Sie führt nur dann zum Ziel, wenn es eine wohldefinierte Abbruchbedingung gibt.

Beispiel 1 – Berechnung der Fakultät - 2 Lösungswege

- *N! – Multipliziere alle Zahlen von 1 bis N miteinander – (Iteration)*
- *N! = N * (N-1)! – Wenn die Fakultät von N-1 bekannt, multipliziere diese mit N (Rekursion)*

```
int  faki( int z) { /* Berechnung mit iterativer Loesung */
    int i, erg = 1;
    for(i=1; i<=z; i++) /* Multipliziere die Zahlen von 1 bis Z */
      erg = erg*i;
    return erg;
}
int  fakr( int z) { /* Berechnung mit rekursiver Loesung */
    int erg;
    if (z == 0) erg = 1; /* Abbruchbedingung 0! ist 1 */
     else erg = z * fakr( z-1); /* Multipliziere Z mit Ergebnis der Berechnung */
    return erg;
}
int  fakro( int z) { /* Berechnung mit rekursiver Lösung – optimiert */
     if (z <= 0) return 1; /* keine unendliche Schleife bei negativen Zahlen */
     else return z * fakro(z-1); /* rekursive Berechnung bei return*/
}
```

1 Rekursive Funktionen

Nutzung und Darstellung des Aufrufes

```
int  fak( int z) {
     if (z == 0) return 1;
     else return z * fak(z-1);
}
 int main() {
     int n, zahl;
    n = fak(4);
}
```

In main –

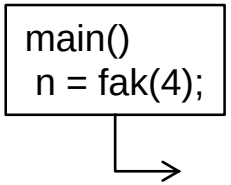
Berechnung eines Ausdruckes zur Bestimmung des Wertes der Variablen n.

Bei Berechnung des Ausdruckes Aufruf einer Funktion fak mit einem Parameter!

```
fak(4)
 4 ist != 0
 return 4 * fak(3);
```

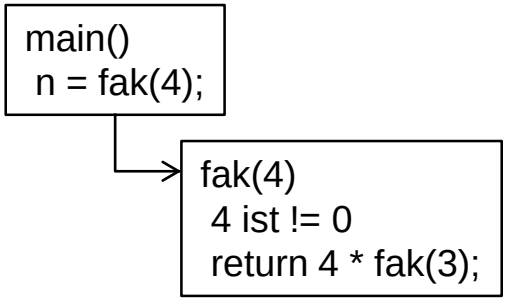
3 Rekursive Funktionen

Darstellung des Aufrufes



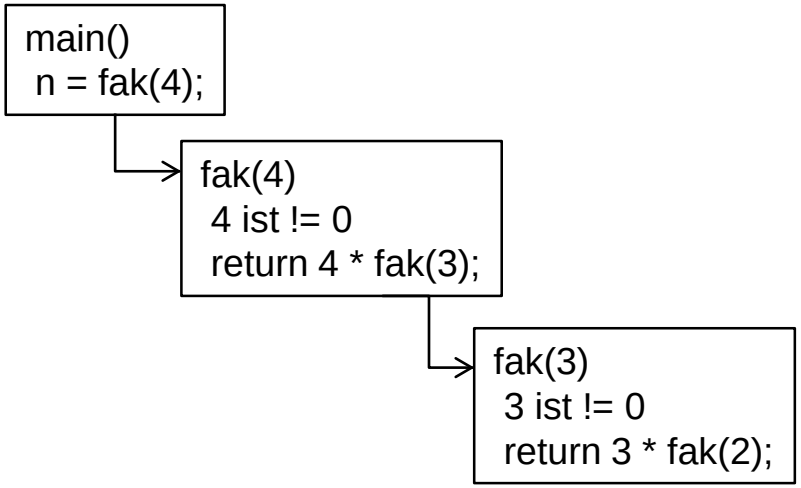
3 Rekursive Funktionen

Darstellung des Aufrufes



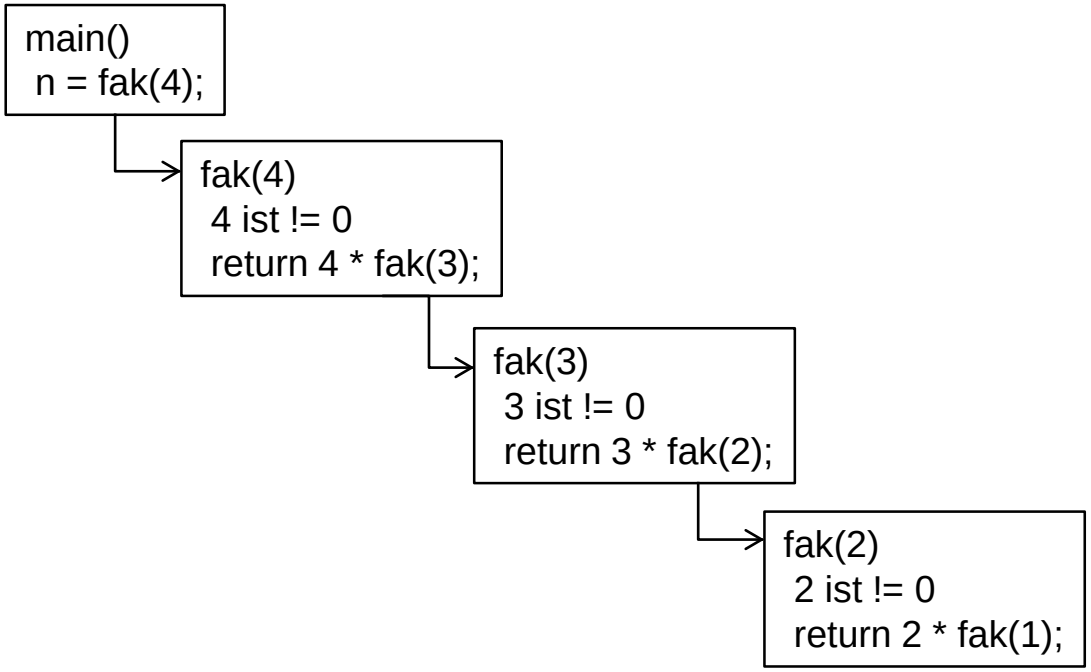
3 Rekursive Funktionen

Darstellung des Aufrufes



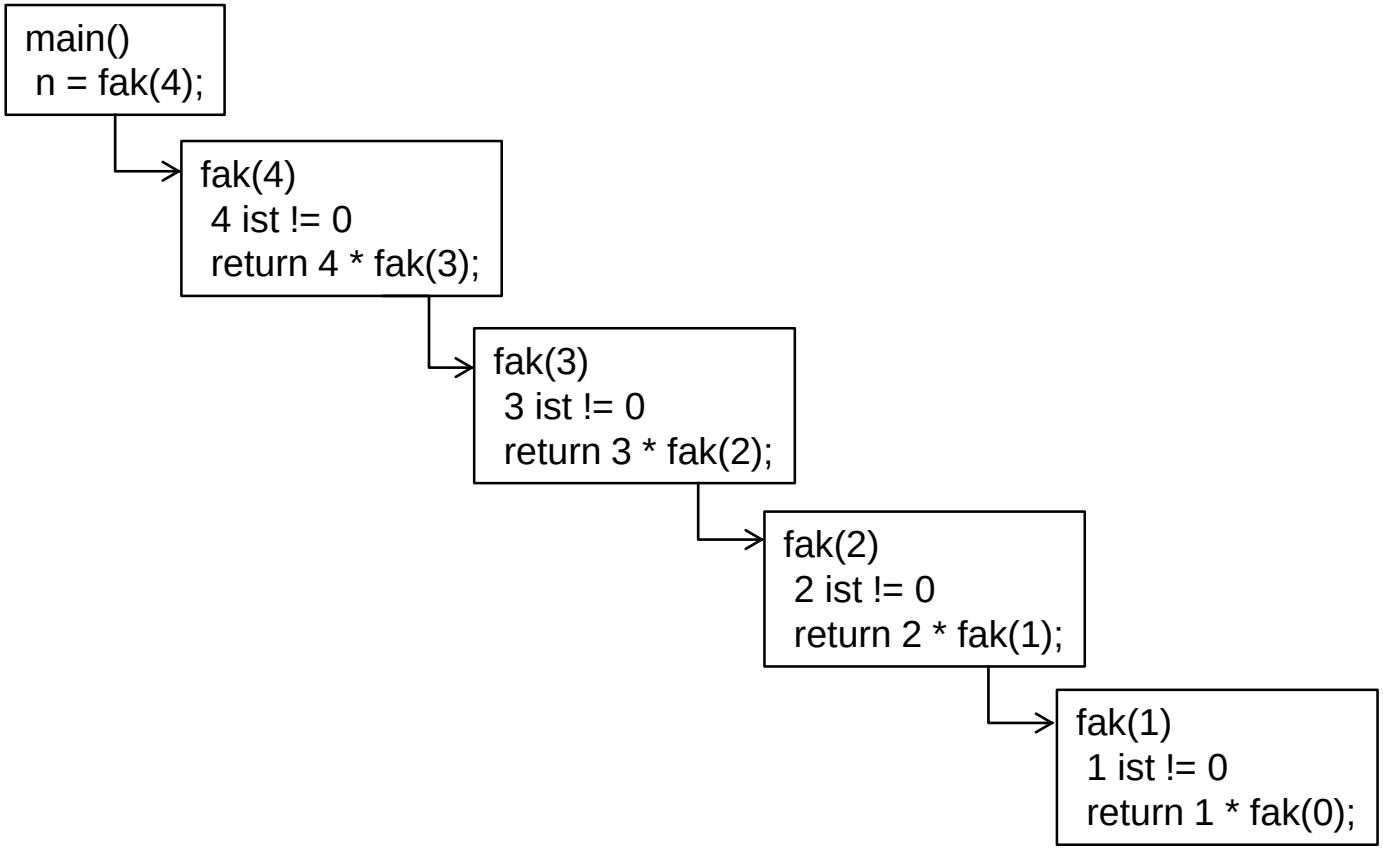
3 Rekursive Funktionen

Darstellung des Aufrufes



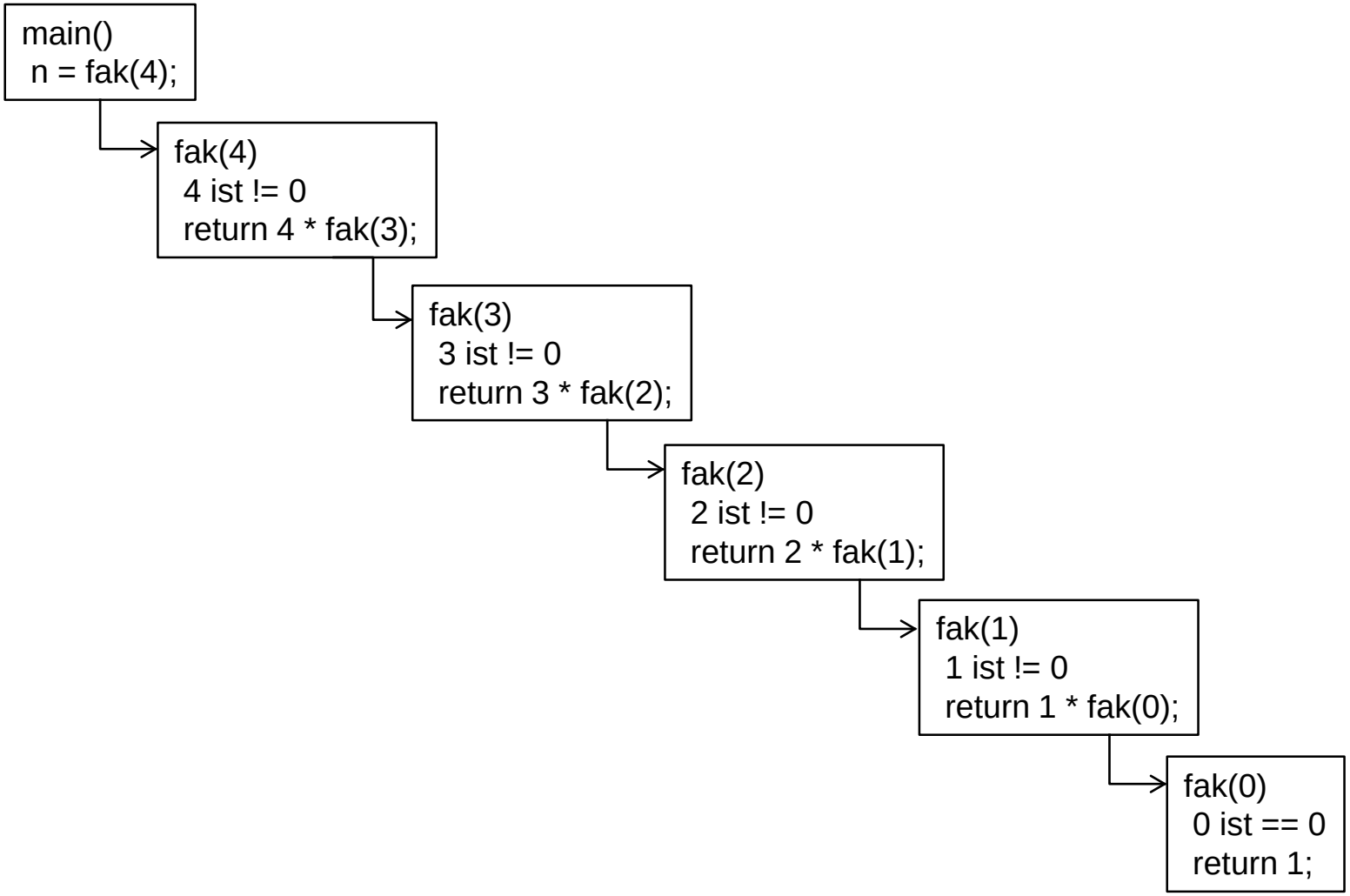
3 Rekursive Funktionen

Darstellung des Aufrufes



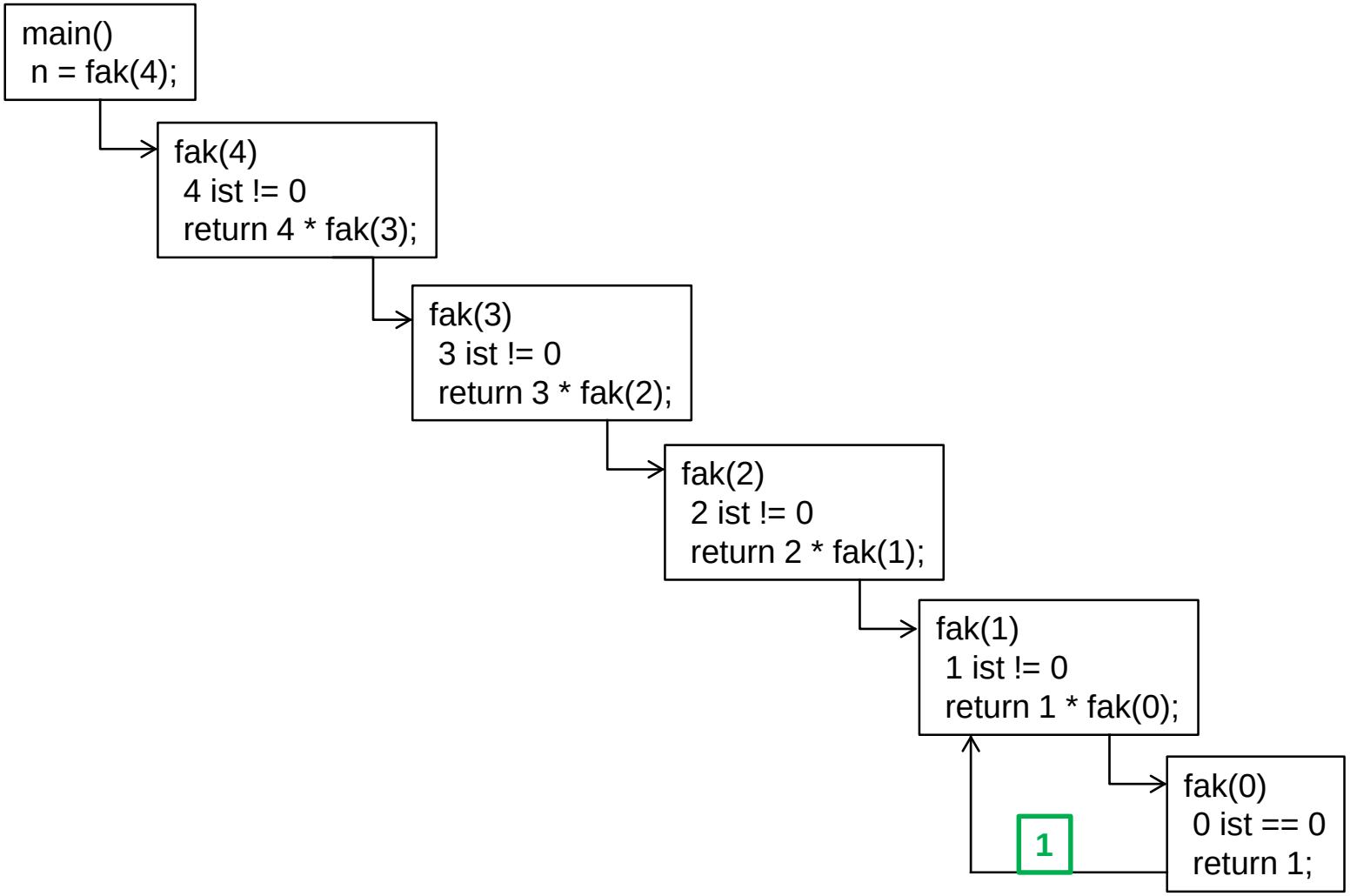
3 Rekursive Funktionen

Darstellung des Aufrufes



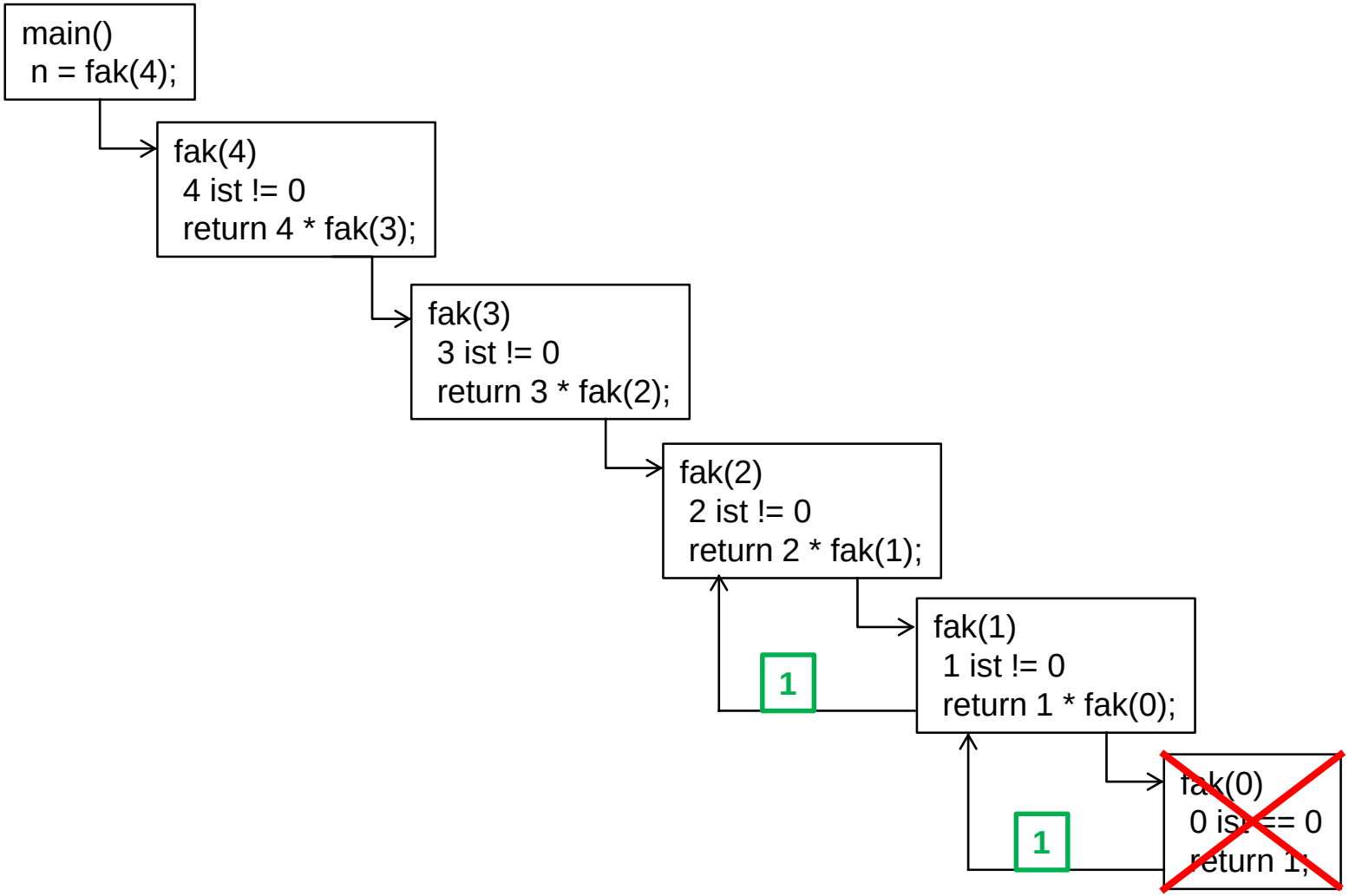
3 Rekursive Funktionen

Nutzung und Darstellung des Aufrufes



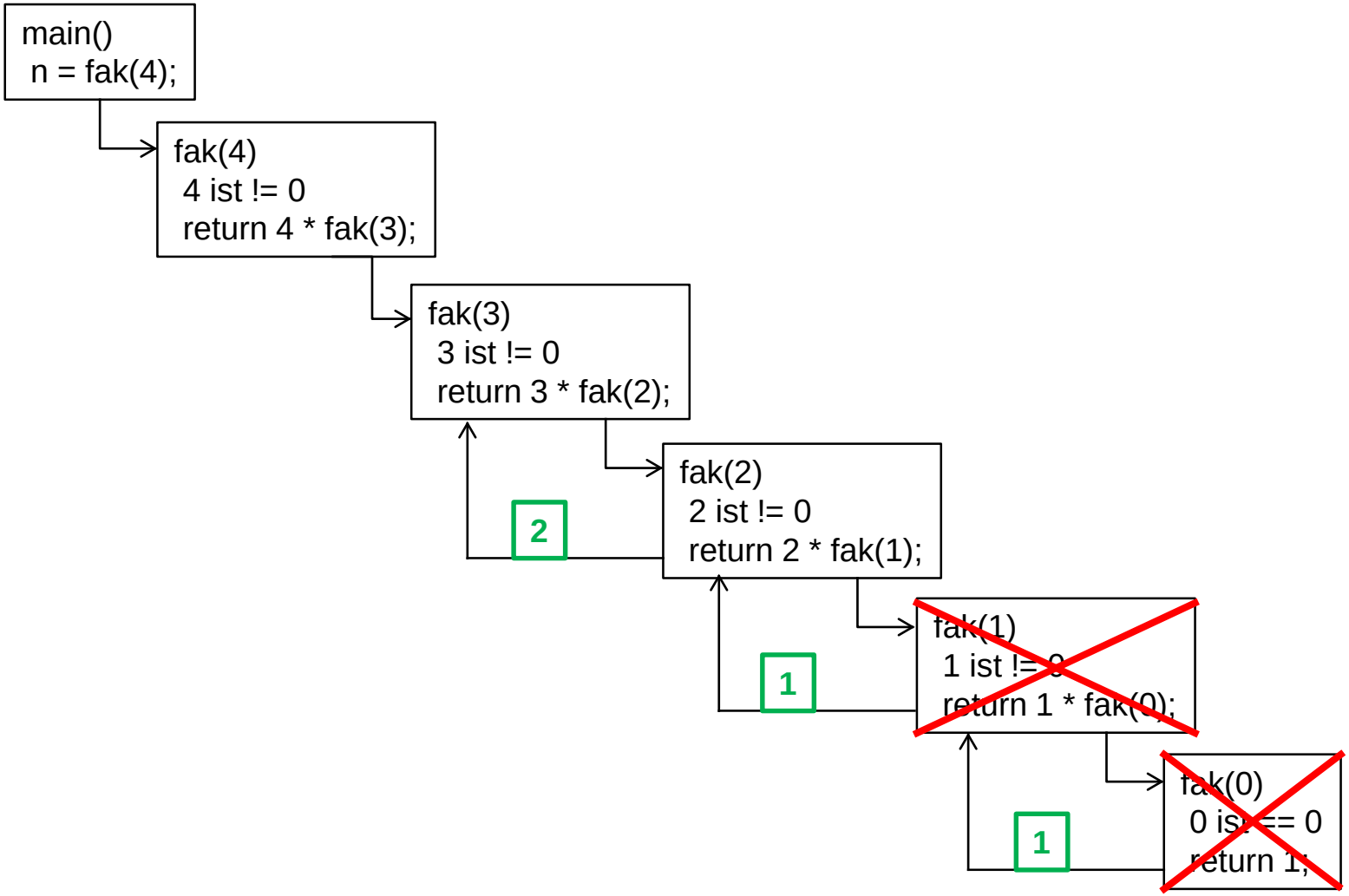
3 Rekursive Funktionen

Nutzung und Darstellung des Aufrufes



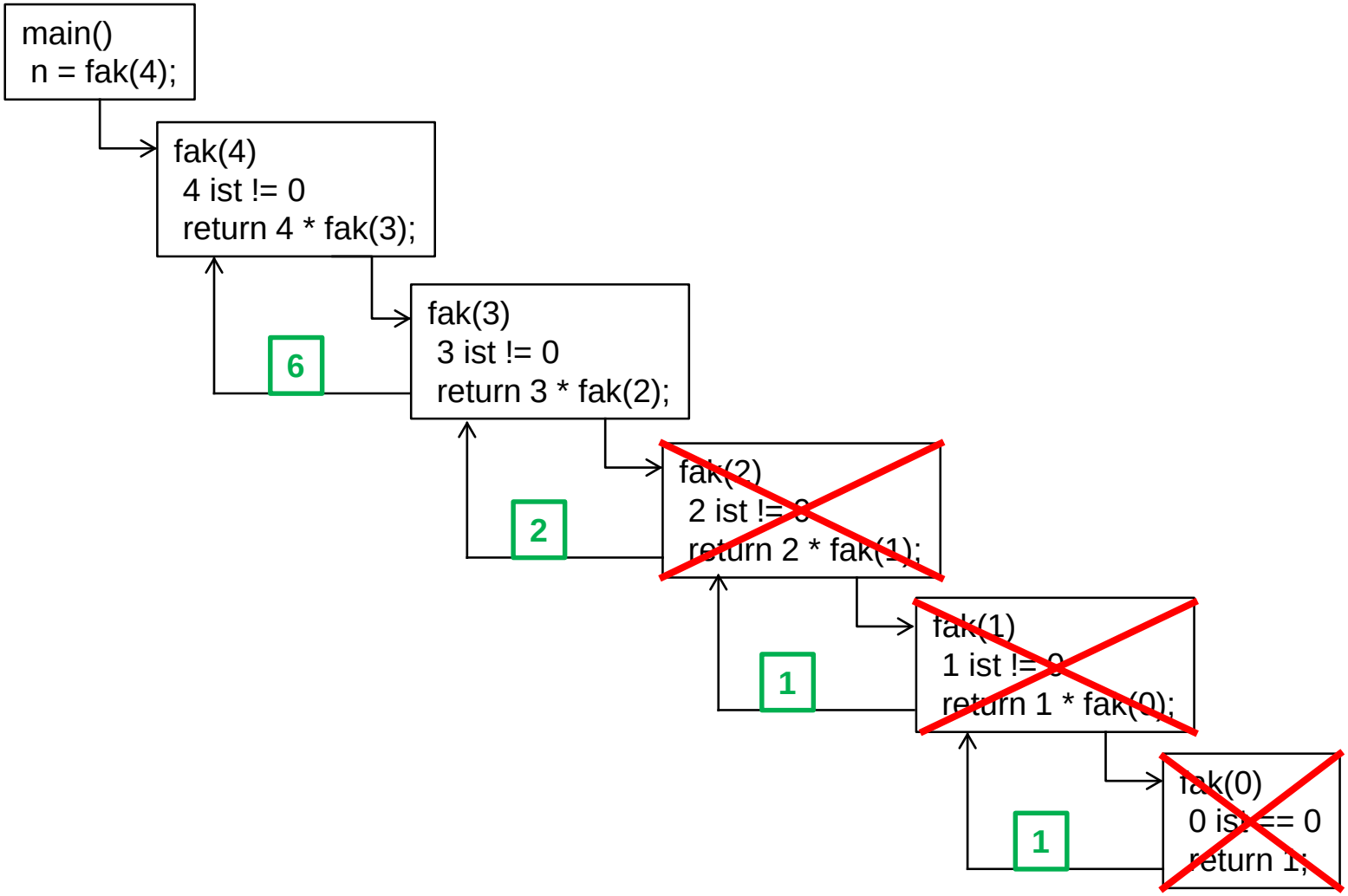
3 Rekursive Funktionen

Nutzung und Darstellung des Aufrufes



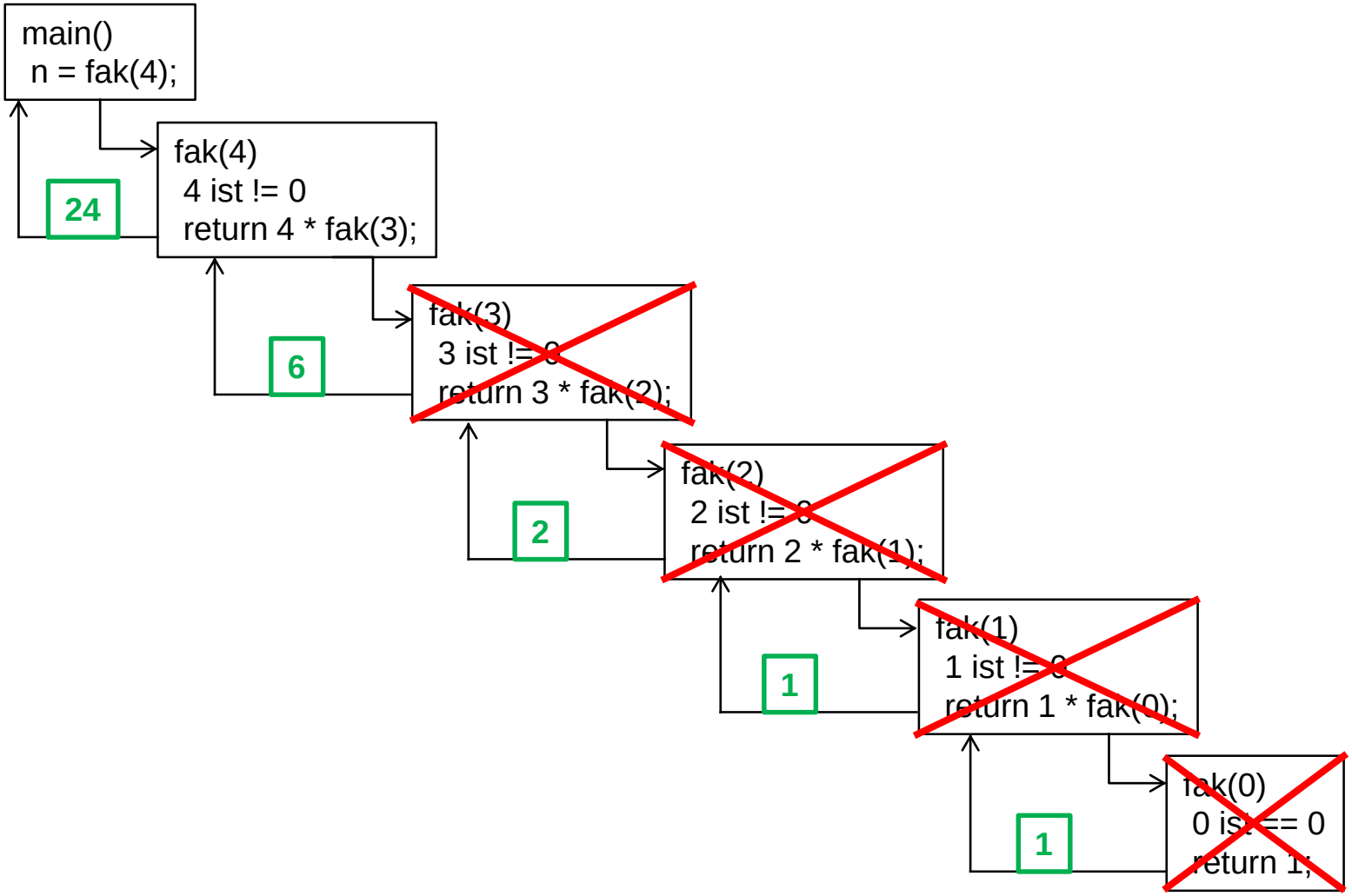
3 Rekursive Funktionen

Nutzung und Darstellung des Aufrufes



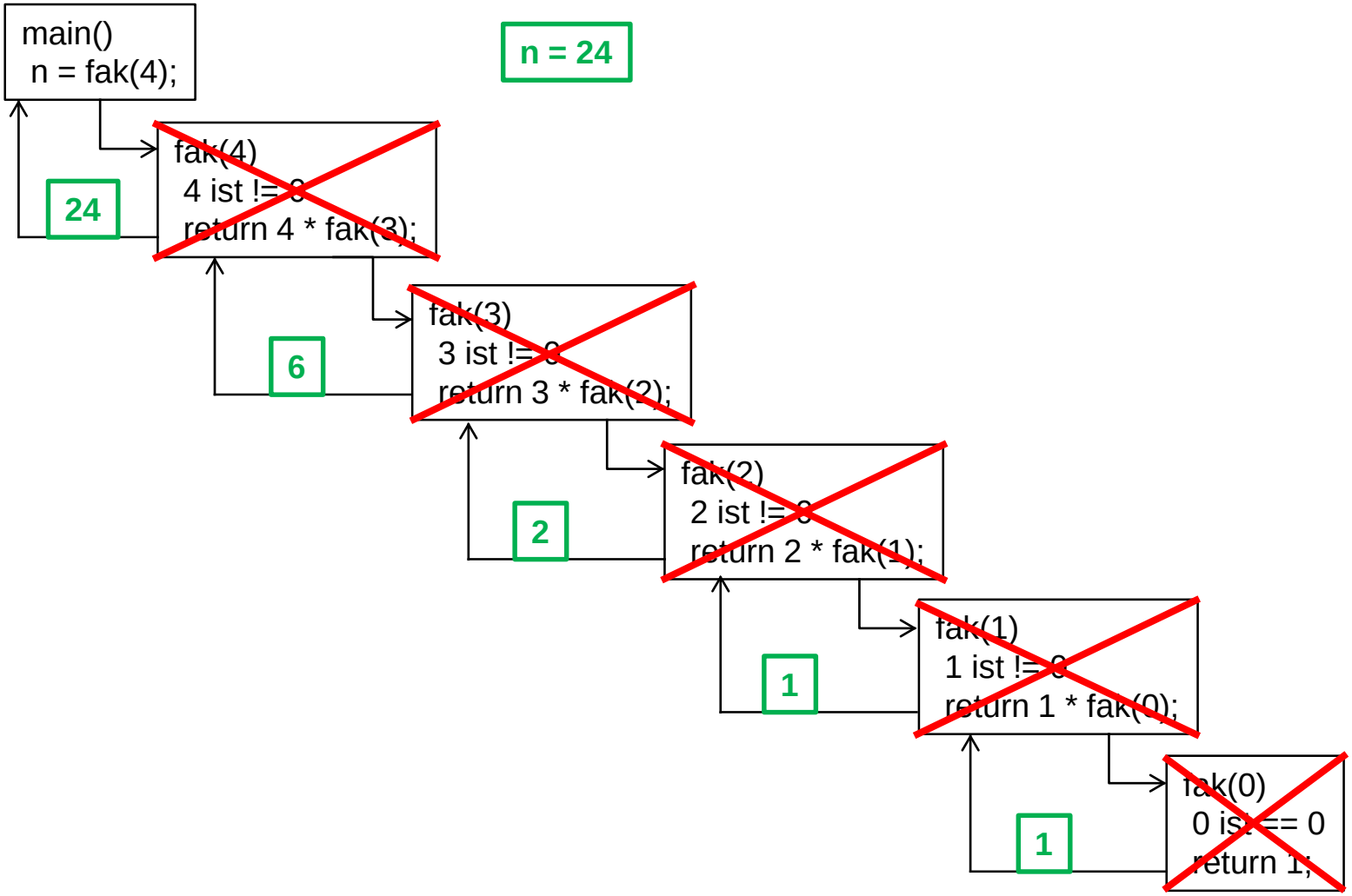
3 Rekursive Funktionen

Nutzung und Darstellung des Aufrufes



3 Rekursive Funktionen

Nutzung und Darstellung des Aufrufes



1 Rekursive Funktionen

Bei einer rekursiven Lösung des Problems wird meist eine Schleife der Iteration durch den rekursiven Aufruf ersetzt.

Beispiel 2 – Ausgabe einer Datei von Standardeingabe Lösung mit iterativer Funktion

```
#include <stdio.h>

void leseZeichen(){
   int c;
   while((c = getchar())!= EOF) /* Solange nicht Ende der Datei */
     putchar(c); /* Gib  das gespeicherte Zeichen aus. */
}


int main (){
   leseZeichen(); /* Aufruf der Funktion */
   printf("\n");
   return 0;
}
```

1 Rekursive Funktionen

Bei einer rekursiven Lösung des Problems wird meist eine Schleife der Iteration durch den rekursiven Aufruf ersetzt.

Beispiel 2 – Ausgabe einer Datei von Standardeingabe Lösung mit rekursiver Funktion

```
#include <stdio.h>

void leseZeichenR(){
   int c;
   if((c = getchar())!= EOF) { /* Wenn das gelesene Zeichen nicht EOF ist  */
     putchar(c);   /* Gib  das gespeicherte Zeichen aus.*/
     leseZeichenR();  /* lies das naechste Zeichen mit rekursiver Funktion */
   }
}
int main (){
   leseZeichenR(); /* Aufruf der Funktion */
   printf("\n");
   return 0;
}
```

1 Rekursive Funktionen

Was passiert, wenn in der Funktion leseZeichen die beiden Anweisungen putchar und der rekursive Aufruf von leseZeichen() getauscht werden?

Beispiel 2 – Ausgabe einer Datei von Standardeingabe Lösung mit rekursiver Funktion

```
#include <stdio.h>

void leseZeichenR1(){
   int c;
   if((c = getchar())!= EOF) { /* Wenn das gelesene Zeichen nicht EOF ist */
    putchar(c);   /* Gib  das gespeicherte Zeichen aus.*/
    leseZeichenR1();  /* lies das naechste Zeichen mit rekursiver Funktion
    putchar(c);   /* Gib das gespeicherte Zeichen aus.*/
   }
}
int main (){
   leseZeichenR1(); /* Aufruf der Funktion */
   printf("\n");
   return 0;
}
```

1 Rekursive Funktionen

Was passiert, wenn in der Funktion leseZeichen die beiden Anweisungen putchar und der rekursive Aufruf von leseZeichen() getauscht werden?

Beim EOF erfolgt die Beendigung der Aufrufkette und die ganzen offenen Funktonsaufrufe werden von hinten nach vorn durch Ausgabe der Zeichen beendet. Die gesamte Datei wird rückwärts ausgegeben!

Ausgabe einer Datei von Standardeingabe Lösung mit rekursiver Funktion

```
#include <stdio.h>

void leseZeichenR1(){
   int c;
   if((c = getchar())!= EOF) { /* Wenn das gelesene Zeichen nicht EOF ist */
    putchar(c);   /* Gib  das gespeicherte Zeichen aus.*/
    leseZeichenR1();  /* lies das naechste Zeichen mit rekursiver Funktion
    putchar(c);   /* Gib das gespeicherte Zeichen aus.*/
   }
}
int main (){
   leseZeichenR1(); /* Aufruf der Funktion */
   printf("\n");
   return 0;
}
```

2. Felder

Nach einfachen Variablen jetzt auch Variable für mehrere Werte!
Deklaration

Datentyp Variablenname[Kapazitaet];

Vor der Verwendung eines Arrays muss die Größe feststehen!
Arrays in C können ihren aktuellen Füllstand nicht selbst bestimmen!

```
/* Variable fuer 100 int- Werte */
    int f[100];
/* Variable fuer 100 char- Werte */
    char puffer[100];
/* Variable fuer 100 x 100 char- Werte */
     char feld[100][100];
/* Weise dem 10sten Eintrag den Wert 42 zu ( Zaehlung beginnt bei 0) */
     f[9] = 42;

/* Weise dem ersten beiden Einträgen die Zeichen ‚T‘ und ‚r‘ zu; */
     puffer[0] = 'T';
     puffer[1] = 'r';

/* Steht in der 3- Zeile und im 4. Zeichen ein ‚T‘ ? */
    if(feld[2][3] == 'T')
```

2. Felder

Beispiel: Folgende Textdatei soll in eine Variable feld eingelesen werden:
Dies ist\n
ein Beispiel\n
zum testen!\n
<EOF>

Da wir nicht wissen, wie viele Zeilen der Nutzer und wieviel Zeichen pro Zeile der Nutzer eingeben wird, müssen wir Annahmen tätigen. Wir wählen ein Feld mit 5 Zeilen zu je 13 Zeichen!

```
/* Variable fuer 5 x 13 char- Werte */
     char feld[5][13];
```

Wir können uns die Variable feld nach einlesen aller Zeichen ohne Newlines so vorstellen:
(wir lassen Feldelemente ungenutzt, da variable Zeilenlängen nicht möglich sind!)

feld	0	1	2	3	4	5	6	7	8	9	10	11	12
0	D	i	e	s		i	s	t					
1	e	i	n		B	e	i	s	p	i	e	l	
2	z	u	m		t	e	s	t	e	n	!		
3													
4													

Oder wir betrachten das feld als fortlaufende Speicherstellen im Rechner:

feld	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
0	D	i	e	s		i	s	t						e	i	n		B	e	i	s	p	i	e	l		z	u	m		t	e	s	t	e	n	!		...

2. Felder und Funktionen

Berechnung von Teilersummen

Bei einer vollkommenen Zahl ist die Zahl gleich der Summe ihrer Teiler kleiner als die Zahl!

z.B. Teilersummen berechnen und in einem Feld speichern!

	f
0	1
1	1
2	1
3	1
4	3
5	1
6	6
7	1
8	7
9	4
10	8
11	1
12	16
13	1
14	10
15	9

Nutzung von `int f[16]; // Variable fuer 16 int - Werte`

```
int tsum(int n){ /* Funktion zur Berechnung */
    int i, g;
    int erg = 1; /* 1 ist immer Teiler (auch bei Primzahlen) */
    for (i=2; i < n; i++)  /* Alle Zahlen durchmustern */
       if(n%i==0)  /* Teiler gefunden */
               erg+=i; /* Teiler addieren */
    return erg; /* Rueckgabe der Teilersumme */
}
int main(){
    int j;
    int f[16]; /* Felddeklaration alle 16 Werte nicht initialisiert */
    for(j=0; j < 16; j++)
          f[j]= tsum(j);
}
```

2. Felder und Funktionen

Berechnung von Teilersummen

Kapzität der Felder muss größer als der Füllstand sein!

Teilersummen berechnen und in einem Feld speichern!

```
#include <stdio.h>
#define MAX 10000

/* globale Felddefinition (ausserhalb von main und jeder anderen Funktion */
int feld[MAX]; /* Variable fuer  MAX int - Werte */


int tsum(int n){ /* Funktion zur Berechnung */
    int i, g;
    int erg = 1; /* 1 ist immer Teiler (auch bei Primzahlen) */
      for (i=2; i < n; i++)  /* Alle Zahlen bis zur Wurzel aus n durchmustern */
       if(n%i==0)  /* Teiler gefunden */
               erg+=i; /* Teiler addieren */
    return erg; /* Rueckgabe der Teilersumme */
}
int main(){
      int j;
      int f[MAX]; /* Felddeklaration alle 10000 Werte nicht initialisiert */
      for(j=0; j < MAX; j++) {
          f[j] = tsum(j);
               /* feld[j] = tsum(j); */
      }

      printf("Vollkommene Zahlen sind: ");
      for(j=0; j < MAX; j++)
        if(j==f[j]) /* Zahl ist Summe ihrer echten Teiler */
                           printf(" %d, ", f[j]);
      printf("\n");
}
```

2. Felder und Funktionen

Vorsicht Fehler

Kein Schutz von Feldgrenzen!

Keine Möglichkeit in der Programmiersprache C zur Laufzeit die Feldgröße zu ermitteln!

```
int fgh ( int anz, int f[]) {
   int i, j, r;
   r = 0;
   j= sizeof(f)/sizeof(int); /* wird schon zur Uebersetzungszeit entschieden,
                                daher hier unbrauchbar */
   return j;
 }
int main() {
   int x, i;
   int a[] = {3, 5, 12, 7, 4};
   x = fgh(5, a);
   printf("x =%d\n", x); /* x = 1 ??? */
   printf(" sizeof a = %d \n", sizeof(a)); /* = 20 Bytes */
   printf(" sizeof a = %d \n", sizeof(a)/sizeof(int) ); /* = 5 (int) */
   for(i=0; i<10;i++) /* Kein Schutz von Feldgrenzen a hat nur 5 – Elemente */
          printf("a[%d]= %d\n", i, a[i]);
 return 0;
```

2. Felder und Funktionen-Vorsicht Fehler

```
#include <stdio.h>
/* Achtung Fehler */
int fgh ( int f[]) {
   /* Feldgroesse kann nicht in Funktion bestimmt werden !*/
   int j;
   j= sizeof(f)/sizeof(int); /* wird schon zur Uebersetzungszeit entschieden, daher unbrauchbar */
   return j;
 }
int main() {
   int a[] = {3, 5, 12, 7, 4};/* Feld der Laenge initialisiert */
   int x = 9, i, j, k, l;
   int b[5]; /* Feld der Laenge 5  nicht initialisiert */
   for(i=0; i<5;i++) // Ausgabe der Elemente
          printf("a[%d]= %d, b[%d]= %d,\n", i, a[i], i, b[i]);
   printf("\n");
   printf("x = %d (ok), j = %d ???\n", x); /* x = 9, j = ??? */
   printf("sizeof a = %d Bytes\n", sizeof(a)); /* Feldgroesse = 20 Bytes */
   /* Feldgroesse = 20 Bytes / Groesse von int 4 Bytes = 5 */
   printf("sizeof a/sizeof int  = %d (Feldlaenge hier berechenbar)\n", sizeof(a)/sizeof(int) );

   printf("Rueckgabe aus Funktion: = %d Feldlaenge dort unbekannt!\n", fgh(a)) ; /* = 1 ? (int) */

   printf("Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente\n");
   for(i=0; i<10;i++) /* Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente - Ausgabe*/
          printf("a[%d]= %d, b[%d]= %d,\n", i, a[i], i, b[i]);
   printf("\n");
   printf("Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente\n");
   for(i=0; i<10;i++) /* Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente - umspeichern */
      b[i] = a[i];
   printf("\n");
   printf("Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente\n");
   for(i=0; i<10;i++) /* Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente - Ausgabe  */
          printf("a[%d]= %d, b[%d]= %d,\n", i, a[i], i, b[i]);
 return 0;
 }
```

2. Felder und Funktionen-Vorsicht Fehler

```
#include <stdio.h>
/* Achtung Fehler */
int fgh ( int f[]) {
   /* Feldgroesse kann nicht in Funktion bestimmt werden !*/
   int j;
   j= sizeof(f)/sizeof(int); /* wird schon zur Uebersetzungszeit entschieden
   return j;
 }
int main() {
   int a[] = {3, 5, 12, 7, 4};/* Feld der Laenge 4 initialisiert */
   int x = 9, i, j, k, l;
   int b[5]; /* Feld der Laenge 5  nicht initialisiert */
   for(i=0; i<5;i++) // Ausgabe der Elemente
          printf("a[%d]= %d, b[%d]= %d,\n", i, a[i], i, b[i]);
   printf("\n");
   printf("x = %d (ok), j = %d ???\n", x); /* x = 9, j = ??? */
   printf("sizeof a = %d Bytes\n", sizeof(a)); /* Feldgroesse = 20 Bytes */
   /* Feldgroesse = 20 Bytes / Groesse von int 4 Bytes = 5 */
   printf("sizeof a/sizeof int  = %d (Feldlaenge hier berechenbar)\n", sizec

   printf("Rueckgabe aus Funktion: = %d Feldlaenge dort unbekannt!\n", fgh(a

   printf("Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente\n");
   for(i=0; i<10;i++) /* Kein Schutz von Feldgrenzen a und b haben nur 5 - E
          printf("a[%d]= %d, b[%d]= %d,\n", i, a[i], i, b[i]);
   printf("\n");
   printf("Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente\n");
   for(i=0; i<10;i++) /* Kein Schutz von Feldgrenzen a und b haben nur 5 - E
      b[i] = a[i];
   printf("\n");
   printf("Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente\n");
   for(i=0; i<10;i++) /* Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente - Ausgabe  */
          printf("a[%d]= %d, b[%d]= %d,\n", i, a[i], i, b[i]);
 return 0;
 }
```

```
a[0]= 3, b[0]= 2003817040,
a[1]= 5, b[1]= 2008855563,
a[2]= 12, b[2]= -2,
a[3]= 7, b[3]= 2003792858,
a[4]= 4, b[4]= 2003793101,

x = 9 (ok), j = 4 ???
sizeof a = 20 Bytes
sizeof a/sizeof int  = 5 (Feldlaenge hier berechenbar)
Rueckgabe aus Funktion: = 1 Feldlaenge dort unbekannt!
Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente
a[0]= 3, b[0]= 2003817040,
a[1]= 5, b[1]= 2008855563,
a[2]= 12, b[2]= -2,
a[3]= 7, b[3]= 2003792858,
a[4]= 4, b[4]= 2003793101,
a[5]= 9, b[5]= 3,
a[6]= 6, b[6]= 5,
a[7]= 37, b[7]= 12,
a[8]= 2, b[8]= 7,
a[9]= 6356884, b[9]= 4,

Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente

Kein Schutz von Feldgrenzen a und b haben nur 5 - Elemente
a[0]= 9, b[0]= 3,
a[1]= 6, b[1]= 5,
a[2]= 37, b[2]= 12,
a[3]= 2, b[3]= 7,
a[4]= 6356884, b[4]= 4,
a[5]= 9, b[5]= 9,
a[6]= 6, b[6]= 6,
a[7]= 37, b[7]= 37,
a[8]= 2, b[8]= 2,
a[9]= 6356884, b[9]= 6356884,
```

3. Aufgabe zu rekursive Funktionen

Schreiben Sie ein Programm, das eine positive ganze Zahl in das binäre Zahlensystem übersetzt. Dabei dürfen Sie die Binärstellen in umgedrehter Reihenfolge ausgeben (d.h., niederwertigste Stelle zuerst), so dass Sie etwa für die Zahl 13_{10} die Ausgabe 1011_2 erhalten (in normaler Reihung wäre die Ausgabe 1101_2). (***Abwandlung aus Hausaufgabe Serie 2!)***

```
#include <stdio.h>
int main() {
        int n, n1;
        printf("Zahl? ");
        scanf("%d",&n);    /* Eingabe der Dezimalzahl */
        printf("%d[10] = ",n); /* Ausgabe der Dezimalzahl */
        while(n > 0) { /* Solange Konvertierung nicht beendet */
          n1 = n/2;
          printf("%d",n - 2*n1); /* Ausgabe der binaeren Ziffer */
          n = n1;
        }
        printf("[2]\n");
}
```

Schreiben Sie eine rekursive Funktion, die diese Problem löst!
Wie ist die Ausgabe in korrekter Reihenfolge möglich?

```
#include <stdio.h>
void binaer (int zahl) {/* hier rekursiven Algorithmus programmieren */}
int main() {
        int n, n1;
        printf("Zahl? ");
        scanf("%d",&n);    /* Eingabe der Dezimalzahl */
        printf("%d[10] = ",n); /* Ausgabe der Dezimalzahl */
        binaer(z); /* Ausgabe der ziffernfolge */
         printf("\n");
}
```

3. Erklärung Palindrom

Palindrom: Unterscheidung zwischen ***Wort*** *–* ***Satz****palindrom*

Zeichenkette vorwärts wie rückwärts gelesen gleich.
Unter Umständen Groß- und Kleinschreibung missachten.

Bei Satzpalindromen unter Missachtung auch von Leer- und Trennzeichen

~~*Dies ist ein Text*~~
Der enthaelt Palindrome emordnilaP tleahtne reD
nun
~~*aber*~~
rentner
~~*Rentner*~~
~~*die liebe ist sieger stets rege ist sie bei leid*~~
~~*Die Liebe ist Sieger; stets rege ist sie bei Leid.*~~
~~*Erika feuert nur untreue Fakire*~~
hallo nun ollah
~~*Regallager*~~
~~*Reliefpfeiler*~~
regallager

```
Dies ist ein Text
Der enthaelt Palindrome emordnilaP tleahtne reD
nun
aber
rentner
Rentner
die liebe ist sieger stets rege ist sie bei leid
Die Liebe ist Sieger; stets rege ist sie bei Leid.
Erika feuert nur untreue Fakire
die liebe ist sieger; stets rege ist sie bei leid.
hallo nun ollah
Regallager
Reliefpfeiler
regallager
```

3. Aufgabe Felder

1. Bestimme längste Zeile einer Textdatei unter Nutzung von eindimensionalem Feldern
   - Datei zeilenweise lesen (Zeilenende könnten die Kodierungen 10, 12 und 13 sein) und dabei die längste Zeile und ausgeben.

```
Dies ist ein Text
Der enthaelt Palindrome emordnilaP tleahtne reD
die liebe ist sieger stets rege ist sie bei leid
Die Liebe ist Sieger; stets rege ist sie bei Leid.
Die laengste Zeile war 49 Zeichen lang
```

2. Einlesen einer Textdatei unter Nutzung von 2 dimensionalen Feldern
   - Datei in zweidimensionales Feld lesen
     (als Zeilenende im Feld Wert 0 verwenden)
   - Feld ausgeben
   - Feld mit Funktion `void ausgabe(char f[])`
     ausgeben, die Zeichen aus Feld bis Kodierung 0 ausgibt.
   - Funktion `int berechneTextLaenge (char f[])` zur Berechnung der Länge des Textes (Wann kommt \0)
   - Funktion `int isPalindrome(char f[], int l)` zur Bestimmung ob Text Palindrom (Vergleich ersten mit letzten ,...)

Ausgabe des Textes aus dem Feld
Dies ist ein Text
...
regallager
Ausgabe des Textes aus dem Feld mit Funktion ausgabe
Dies ist ein Text
...
Regallager

Die laengste Zeile war 50 Zeichen lang
Das sind folgende Zeilen:
`Die Liebe ist Sieger; stets rege ist sie bei Leid.`
`die liebe ist sieger; stets rege ist sie bei leid.`