2. Übungsbesprechung Programmkonstruktion

Karl Gmeiner

karl@complang.tuwien.ac.at

October 27, 2011

Rückblick und Ausblick

Übungsblatt 1

- Über 500 Abgaben (von 1000 Anmeldungen)
- Fehlende Matrikelnummer/Name, mehrfache Gruppen, falsche Matrikelnummer
- Grammatik
  - `Ausdruck = Wert { "+" Wert }`
    `| Wert { "*" Wert }`
  - `Wert "+" Wert "+" Wert` ist ein Ausdruck
  - `Wert "+" Wert "*" Wert` ist kein Ausdruck: Auf `Wert "+" Wert` kann nur beliebig oft `"+" Wert` folgen.

Test

- 10. November; Gruppen von 13:07 - 13:53 und 14:07 - 14:53
- Aufgaben ähnlich zu Übungsblatt 1&2, Kontrollfragen zu Kapitel 1&2

switch-case-Anweisung

```
switch ( Variable ) { // Typen von Variable eingeschraenkt
case Wert1:
    /* Variable == Wert1 */
    break;
```

switch-case-Anweisung

```
switch ( Variable ) { // Typen von Variable eingeschraenkt
case Wert1:
    /* Variable == Wert1 */
    break;
case Wert2:
    /* Variable == Wert2 */
    /* hier kein break */
case Wert3:
    /* Fall-through [schlechter Stil!]  */
    /* Variable == Wert2 || Variable == Wert3 */
    break;
```

switch-case-Anweisung

```
switch ( Variable ) { // Typen von Variable eingeschraenkt
case Wert1:
    /* Variable == Wert1 */
    break;
case Wert2:
    /* Variable == Wert2 */
    /* hier kein break */
case Wert3:
    /* Fall-through [schlechter Stil!]  */
    /* Variable == Wert2 || Variable == Wert3 */
    break;
default:
    /* Variable ist weder Wert1, Wert2 noch Wert3 */
    break;
}
```

Aufgabe 1.1 - if in switch-case umwandeln

if-Anweisung

```
int wochentag = /* ...  */;

boolean arbeitstag;

if(wochentag == 6) {
    // Samstag
    arbeitstag = false;
} else if(wochentag == 7) {
    // Sonntag
    arbeitstag = false;
} else {
    arbeitstag = true;
}
```

Aufgabe 1.1 - if in switch-case umwandeln

if-Anweisung

```
int wochentag = /* ...  */;

boolean arbeitstag;

if(wochentag == 6) {
    // Samstag
    arbeitstag = false;
} else if(wochentag == 7) {
    // Sonntag
    arbeitstag = false;
} else {
    arbeitstag = true;
}
```

switch-case-Anweisung

```
boolean arbeitstag;

switch(wochentag) {
case 6 :  // Samstag
    arbeitstag = false;
    break;
case 7 :  // Sonntag
    arbeitstag = false;
    break;
default:
    arbeitstag = true;
    break;
}
```

Aufgabe 1.1 - if in switch-case umwandeln

if-Anweisung

```
int wochentag = /* ...  */;

boolean arbeitstag;

if(wochentag == 6) {
    // Samstag
    arbeitstag = false;
} else if(wochentag == 7) {
    // Sonntag
    arbeitstag = false;
} else {
    arbeitstag = true;
}
```

switch-case-Anweisung

```
boolean arbeitstag;

switch(wochentag) {
case 6 :  // Samstag
case 7 :  // Sonntag
    // Schlechter Stil!
    arbeitstag = false;
    break;
default:
    arbeitstag = true;
    break;
}
```

Aufgabe 1.1 - if in switch-case umwandeln

if-Anweisung

```
int wochentag = /* ...  */;

boolean arbeitstag;

if(wochentag == 6) {
    // Samstag
    arbeitstag = false;
} else if(wochentag == 7) {
    // Sonntag
    arbeitstag = false;
} else {
    arbeitstag = true;
}
```

switch-case-Anweisung

```
boolean arbeitstag;

switch(wochentag) {
case 6 :  // Samstag
case 7 :  // Sonntag
    // Schlechter Stil!
    arbeitstag = false;
    break;
default:
    arbeitstag = true;
    break;
}
```

C#: case-Block muss mit *Jump*-Anweisung enden (z.B. `break`, `return`).

Aufgabe 1.1 - if in Bedingungsoperator umwandeln

if-Anweisung

```
boolean arbeitstag;

if(wochentag == 6) { // Samstag
    arbeitstag = false;
} else if(wochentag == 7) { // Sonntag
    arbeitstag = false;
} else {
    arbeitstag = true;
}
```

Aufgabe 1.1 - if in Bedingungsoperator umwandeln

if-Anweisung

```
boolean arbeitstag;

if(wochentag == 6) { // Samstag
    arbeitstag = false;
} else if(wochentag == 7) { // Sonntag
    arbeitstag = false;
} else {
    arbeitstag = true;
}
```

Bedingungsoperator und Boolsche Operationen

```
boolean arbeitstag = wochentag == 6 ?  false :
                     wochentag == 7 ? false : true;
```

Aufgabe 1.1 - if in Bedingungsoperator umwandeln

if-Anweisung

```
boolean arbeitstag;

if(wochentag == 6 || wochentag == 7) { // Samstag, Sonntag
    arbeitstag = false;
} else {
    arbeitstag = true;
}
```

Bedingungsoperator und Boolsche Operationen

```
boolean arbeitstag = wochentag == 6 ?  false :
                     wochentag == 7 ? false : true;
```

Aufgabe 1.1 - if in Bedingungsoperator umwandeln

if-Anweisung

```
boolean arbeitstag;

if(wochentag == 6 || wochentag == 7) { // Samstag, Sonntag
    arbeitstag = false;
} else {
    arbeitstag = true;
}
```

Bedingungsoperator und Boolsche Operationen

```
boolean arbeitstag = wochentag == 6 || wochentag == 7 ?
                     false :  true;
```

Aufgabe 1.1 - if in Bedingungsoperator umwandeln

if-Anweisung

```
boolean arbeitstag;

if(wochentag == 6 || wochentag == 7) { // Samstag, Sonntag
    arbeitstag = false;
} else {
    arbeitstag = true;
}
```

Bedingungsoperator und Boolsche Operationen

```
boolean arbeitstag = ! ( wochentag == 6 || wochentag == 7 );
```

Aufgabe 1.2.1 - switch-case in if umwandeln

switch-case-Anweisung 1

```
int i = /* ...  */;

switch(i) {
    case 0:  i++;
    case 1:  i += 2;
    case 2:  i += 3;
    default:  i += 4;
}
```

Aufgabe 1.2.1 - switch-case in if umwandeln

switch-case-Anweisung 1

```
int i = /* ...  */;

switch(i) {
    case 0:  i++;
    case 1:  i += 2;
    case 2:  i += 3;
    default:  i += 4;
}
```

if-Anweisung

```
int i = /* ...  */;

        if(i == 0) {
            i++;
        }
```

Aufgabe 1.2.1 - switch-case in if umwandeln

switch-case-Anweisung 1

```
int i = /* ...  */;

switch(i) {
    case 0:  i++;
    case 1:  i += 2;
    case 2:  i += 3;
    default:  i += 4;
}
```

if-Anweisung

```
int i = /* ...  */;


    if(i == 0 || i == 1) {
        if(i == 0) {
            i++;
        }
        i += 2;
    }
```

Aufgabe 1.2.1 - switch-case in if umwandeln

switch-case-Anweisung 1

```
int i = /* ...  */;

switch(i) {
    case 0:  i++;
    case 1:  i += 2;
    case 2:  i += 3;
    default:  i += 4;
}
```

if-Anweisung

```
int i = /* ...  */;

if(i == 0 || i == 1 || i == 2) {
    if(i == 0 || i == 1) {
        if(i == 0) {
            i++;
        }
        i += 2;
    }
    i += 3;
}
```

Aufgabe 1.2.1 - switch-case in if umwandeln

switch-case-Anweisung 1

```
int i = /* ...  */;

switch(i) {
    case 0:  i++;
    case 1:  i += 2;
    case 2:  i += 3;
    default:  i += 4;
}
```

if-Anweisung

```
int i = /* ...  */;

if(i == 0 || i == 1 || i == 2) {
    if(i == 0 || i == 1) {
        if(i == 0) {
            i++;
        }
        i += 2;
    }
    i += 3;
}
i += 4;
```

Aufgabe 1.2.2

switch-case-Anweisung 2

```
int i = /* ...  */;

switch(i) {
    case 0:  i++;
    case 1:  i += 2;
             break;
    case 2:  i += 3;
    default:  i += 4;
}
```

Aufgabe 1.2.2

switch-case-Anweisung 2

```
int i = /* ...  */;

switch(i) {
    case 0:  i++;
    case 1:  i += 2;
             break;
    case 2:  i += 3;
    default:  i += 4;
}
```

if-Anweisung

```
int i = /* ...  */;


    if(i == 0) {
        i++;
    }
```

Aufgabe 1.2.2

switch-case-Anweisung 2

```
int i = /* ...  */;

switch(i) {
    case 0:  i++;
    case 1:  i += 2;
             break;
    case 2:  i += 3;
    default:  i += 4;
}
```

if-Anweisung

```
int i = /* ...  */;

if(i == 0 || i == 1) {
    if(i == 0) {
        i++;
    }
    i += 2;
}
```

Aufgabe 1.2.2

switch-case-Anweisung 2

```
int i = /* ...  */;

switch(i) {
    case 0:  i++;
    case 1:  i += 2;
             break;
    case 2:  i += 3;
    default:  i += 4;
}
```

if-Anweisung

```
int i = /* ...  */;

if(i == 0 || i == 1) {
    if(i == 0) {
        i++;
    }
    i += 2;
} else {
    if(i == 2) {
        i += 3;
    }
    i += 4;
}
```

Aufgabe 2.1 - Fakultät

for-Schleife

```
int n = /* ...  */;

int fac = 1;

for(int i = 1; i < n; i++) {
    fac *= i;
}
```

Aufgabe 2.1 - Fakultät

for-Schleife

```
int n = /* ...  */;

int fac = 1;

for(int i = 1; i < n; i++) {
    fac *= i;
}
```

while-Schleife

```
int n = /* ...  */;

int fac = 1;

int i = 1;
while(i < n) {
    fac *= i;
    i++;
}
```

Aufgabe 2.1 - Fakultät

for-Schleife

```
int n = /* ...  */;

int fac = 1;

for(int i = 1; i < n; i++) {
    fac *= i;
}
```

do-while-Schleife

```
int n = /* ...  */;

int fac = 1;

int i = 1;
do {
    fac *= i;
    i++;
} while(i < n);
```

Aufgabe 2.1 - Fakultät

for-Schleife

```
int n = /* ...  */;

int fac = 1;

for(int i = 1; i < n; i++) {
    fac *= i;
}
```

do-while-Schleife

```
int n = /* ...  */;

int fac = 1;

int i = 1;
do {
    fac *= i;
    i++;
} while(i < n);
// Fall n <= 1?
```

Aufgabe 2.2 - Heron-Verfahren

while-Schleife

```
int n = /* ...  */;

int i = 0;

double a = 2;
double sqrtA = 1;

while(i <= n) {
    sqrtA =
        (sqrtA+a/sqrtA)/2.;
    i++;
}
```

Aufgabe 2.2 - Heron-Verfahren

while-Schleife

```
int n = /* ...  */;

int i = 0;

double a = 2;
double sqrtA = 1;

while(i <= n) {
    sqrtA =
        (sqrtA+a/sqrtA)/2.;
    i++;
}
```

do-while-Schleife

```
int n = /* ...  */;

int i = 0;

double a = 2;
double sqrtA = 1;

do {
    sqrtA =
        (sqrtA+a/sqrtA)/2.;
    i++;
} while(i <= n)
```

Aufgabe 2.2 - Heron-Verfahren

while-Schleife

```
int n = /* ...  */;

int i = 0;

double a = 2;
double sqrtA = 1;

while(i <= n) {
    sqrtA =
        (sqrtA+a/sqrtA)/2.;
    i++;
}
```

do-while-Schleife

```
int n = /* ...  */;

int i = 0;

double a = 2;
double sqrtA = 1;

do {
    sqrtA =
        (sqrtA+a/sqrtA)/2.;
    i++;
} while(i <= n)
// Fall n < 0 ?
```

Aufgabe 2.2 - Heron-Verfahren

while-Schleife

```
int n = /* ...  */;

int i = 0;

double a = 2;
double sqrtA = 1;

while(i <= n) {
    sqrtA =
        (sqrtA+a/sqrtA)/2.;
    i++;
}
```

do-while-Schleife

```
int n = /* ...  */;

int i = 0;

double a = 2;
double sqrtA = 1;

if(i <= n) {
    do {
        sqrtA =
        (sqrtA+a/sqrtA)/2.;
        i++;
    } while(i <= n)
}
```

Aufgabe 3 - Euklidischer Algorithmus

Programm-Code

```
int m = 1234;
int n = 567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	1234	567

Aufgabe 3 - Euklidischer Algorithmus

Programm-Code

```
int m = 1234;
int n = 567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	1234	567
m = m - n	667	567

Aufgabe 3 - Euklidischer Algorithmus

Programm-Code

```
int m = 1234;
int n = 567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	1234	567
m = m - n	667	567
m = m - n	100	567

Aufgabe 3 - Euklidischer Algorithmus

Programm-Code

```
int m = 1234;
int n = 567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	1234	567
`m = m - n`	667	567
`m = m - n`	100	567
`n = n - m`	100	467

Aufgabe 3 - Euklidischer Algorithmus

Programm-Code

```
int m = 1234;
int n = 567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	1234	567
m = m - n	667	567
m = m - n	100	567
n = n - m	100	467
n = n - m	100	367
n = n - m	100	267
n = n - m	100	167
n = n - m	100	67

Aufgabe 3 - Euklidischer Algorithmus

Programm-Code

```
int m = 1234;
int n = 567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	1234	567
m = m - n	667	567
m = m - n	100	567
n = n - m	100	467
n = n - m	100	367
n = n - m	100	267
n = n - m	100	167
n = n - m	100	67
m = m - n	33	67

Aufgabe 3 - Euklidischer Algorithmus

Programm-Code

```
int m = 1234;
int n = 567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	1234	567
m = m - n	667	567
m = m - n	100	567
n = n - m	100	467
n = n - m	100	367
n = n - m	100	267
n = n - m	100	167
n = n - m	100	67
m = m - n	33	67
n = n - m	33	34

Aufgabe 3 - Euklidischer Algorithmus

Programm-Code

```
int m = 1234;
int n = 567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	1234	567
m = m - n	667	567
m = m - n	100	567
n = n - m	100	467
n = n - m	100	367
n = n - m	100	267
n = n - m	100	167
n = n - m	100	67
m = m - n	33	67
n = n - m	33	34
m = m - n	33	1

Aufgabe 3 - Euklidischer Algorithmus

Programm-Code

```
int m = 1234;
int n = 567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	1234	567
m = m - n	667	567
m = m - n	100	567
n = n - m	100	467
n = n - m	100	367
n = n - m	100	267
n = n - m	100	167
n = n - m	100	67
m = m - n	33	67
n = n - m	33	34
m = m - n	33	1
...	...	...
m = m - n	1	1

Euklidischer Algorithmus - Verletzen der Zusicherung

Programm-Code

```
int m = -1234;
int n = -567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	-1234	-567

Euklidischer Algorithmus - Verletzen der Zusicherung

Programm-Code

```
int m = -1234;
int n = -567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	-1234	-567
n = n - m	-1234	667

Euklidischer Algorithmus - Verletzen der Zusicherung

Programm-Code

```
int m = -1234;
int n = -567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	-1234	-567
n = n - m	-1234	667
n = n - m	-1234	1901

Euklidischer Algorithmus - Verletzen der Zusicherung

Programm-Code

```
int m = -1234;
int n = -567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	-1234	-567
n = n - m	-1234	667
n = n - m	-1234	1901
n = n - m	-1234	3135

Euklidischer Algorithmus - Verletzen der Zusicherung

Programm-Code

```
int m = -1234;
int n = -567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	-1234	-567
n = n - m	-1234	667
n = n - m	-1234	1901
n = n - m	-1234	3135
n = n - m	-1234	4369

Euklidischer Algorithmus - Verletzen der Zusicherung

Programm-Code

```
int m = -1234;
int n = -567;

while (m != n)
    if (m > n)
        m = m - n;
    else //es gilt n>m
        n = n - m;
```

Variablenzustände

Anweisung	m	n
	-1234	-567
n = n - m	-1234	667
n = n - m	-1234	1901
n = n - m	-1234	3135
n = n - m	-1234	4369
n = n - m	-1234	5603

Aufgabe 4 - Fibonacci-Zahlen

Rekursiv `rfib`

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return rfib(n - 1) +
        rfib(n - 2);
}
```

Iterativ `ifib`

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    int a = 0;
    int b = 1;
    for(int i=2; i<=n; i++) {
        b = a + b;
        a = b - a;
    }
    return b;
}
```

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	`rfib(6)`	`rfib(5) + rfib(4)`

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	`rfib(6)`	`rfib(5) + rfib(4)`
?	1	`rfib(5)`	?
?	1	`rfib(4)`	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	`rfib(6)`	`rfib(5) + rfib(4)`
2	1	`rfib(5)`	`rfib(4) + rfib(3)`
?	2	`rfib(4)`	?
?	2	`rfib(3)`	?
?	1	`rfib(4)`	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
?	3	rfib(3)	?
?	3	rfib(2)	?
?	2	rfib(3)	?
?	1	rfib(4)	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
4	3	rfib(3)	rfib(2) + rfib(1)
?	4	rfib(2)	?
?	4	rfib(1)	?
?	3	rfib(2)	?
?	2	rfib(3)	?
?	1	rfib(4)	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
4	3	rfib(3)	rfib(2) + rfib(1)
5	4	rfib(2)	rfib(1) + rfib(0)
?	5	rfib(1)	?
?	5	rfib(0)	?
?	4	rfib(1)	?
?	3	rfib(2)	?
?	2	rfib(3)	?
?	1	rfib(4)	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
4	3	rfib(3)	rfib(2) + rfib(1)
5	4	rfib(2)	rfib(1) + rfib(0)
6	5	rfib(1)	return 1
?	5	rfib(0)	?
?	4	rfib(1)	?
?	3	rfib(2)	?
?	2	rfib(3)	?
?	1	rfib(4)	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
4	3	rfib(3)	rfib(2) + rfib(1)
5	4	rfib(2)	rfib(1) + rfib(0)
6	5	rfib(1)	return 1
7	5	rfib(0)	return 0
?	4	rfib(1)	?
?	3	rfib(2)	?
?	2	rfib(3)	?
?	1	rfib(4)	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
4	3	rfib(3)	rfib(2) + rfib(1)
5	4	rfib(2)	rfib(1) + rfib(0)
6	5	rfib(1)	return 1
7	5	rfib(0)	return 0
8	4	rfib(1)	return 1
?	3	rfib(2)	?
?	2	rfib(3)	?
?	1	rfib(4)	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
4	3	rfib(3)	rfib(2) + rfib(1)
5	4	rfib(2)	rfib(1) + rfib(0)
6	5	rfib(1)	return 1
7	5	rfib(0)	return 0
8	4	rfib(1)	return 1
9	3	rfib(2)	rfib(1) + rfib(0)
?	9	rfib(1)	?
?	9	rfib(0)	?
?	2	rfib(3)	?
?	1	rfib(4)	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
4	3	rfib(3)	rfib(2) + rfib(1)
5	4	rfib(2)	rfib(1) + rfib(0)
6	5	rfib(1)	return 1
7	5	rfib(0)	return 0
8	4	rfib(1)	return 1
9	3	rfib(2)	rfib(1) + rfib(0)
10	9	rfib(1)	return 1
11	9	rfib(0)	return 0
?	2	rfib(3)	?
?	1	rfib(4)	?

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
4	3	rfib(3)	rfib(2) + rfib(1)
5	4	rfib(2)	rfib(1) + rfib(0)
6	5	rfib(1)	return 1
7	5	rfib(0)	return 0
8	4	rfib(1)	return 1
9	3	rfib(2)	rfib(1) + rfib(0)
10	9	rfib(1)	return 1
11	9	rfib(0)	return 0
12	2	rfib(3)	rfib(2) + rfib(1)
...	...	...	...
17	1	rfib(4)	rfib(3) + rfib(2)
...	...	...	...

Fibonacci-Zahlen rekursiv

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    return
        rfib(n-1)+
        rfib(n-2);
}
```

1x rfib(6),
1x rfib(5),
2x rfib(4),
3x rfib(3),
5x rfib(2),
8x rfib(1),
5x rfib(0).

Index	Von	Aufruf	Anweisung
1	–	rfib(6)	rfib(5) + rfib(4)
2	1	rfib(5)	rfib(4) + rfib(3)
3	2	rfib(4)	rfib(3) + rfib(2)
4	3	rfib(3)	rfib(2) + rfib(1)
5	4	rfib(2)	rfib(1) + rfib(0)
6	5	rfib(1)	return 1
7	5	rfib(0)	return 0
8	4	rfib(1)	return 1
9	3	rfib(2)	rfib(1) + rfib(0)
10	9	rfib(1)	return 1
11	9	rfib(0)	return 0
12	2	rfib(3)	rfib(2) + rfib(1)
...	...	...	...
17	1	rfib(4)	rfib(3) + rfib(2)
...	...	...	...

Fibonacci-Zahlen iterativ

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    int a = 0;
    int b = 1;
    for(int i=2; i<=n; i++) {
        b = a + b;
        a = b - a;
    }
    return b;
}
```

i	a	b
-	0	1

Fibonacci-Zahlen iterativ

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    int a = 0;
    int b = 1;
    for(int i=2; i<=n; i++) {
        b = a + b;
        a = b - a;
    }
    return b;
}
```

i	a	b
-	0	1
2	1	1

Fibonacci-Zahlen iterativ

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    int a = 0;
    int b = 1;
    for(int i=2; i<=n; i++) {
        b = a + b;
        a = b - a;
    }
    return b;
}
```

i	a	b
-	0	1
2	1	1
3	1	2

Fibonacci-Zahlen iterativ

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    int a = 0;
    int b = 1;
    for(int i=2; i<=n; i++) {
        b = a + b;
        a = b - a;
    }
    return b;
}
```

i	a	b
-	0	1
2	1	1
3	1	2
4	2	3

Fibonacci-Zahlen iterativ

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    int a = 0;
    int b = 1;
    for(int i=2; i<=n; i++) {
        b = a + b;
        a = b - a;
    }
    return b;
}
```

i	a	b
-	0	1
2	1	1
3	1	2
4	2	3
5	3	5

Fibonacci-Zahlen iterativ

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    int a = 0;
    int b = 1;
    for(int i=2; i<=n; i++) {
        b = a + b;
        a = b - a;
    }
    return b;
}
```

i	a	b
-	0	1
2	1	1
3	1	2
4	2	3
5	3	5
6	5	8
7	8	13
8	13	21
9	21	34
10	34	55
11	55	89
12	89	144

Fibonacci-Zahlen iterativ

```
 if(n == 0) {
    return 0;
} else if(n == 1) {
    return 1;
} else {
    int a = 0;
    int b = 1;
    for(int i=2; i<=n; i++) {
        b = a + b;
        a = b - a;
    }
    return b;
}
```

i	a	b
-	0	1
2	1	1
3	1	2
4	2	3
5	3	5
6	5	8
7	8	13
8	13	21
9	21	34
10	34	55
11	55	89
12	89	144

- `rfib(12)` würde 144 Mal `rfib(1)` aufrufen!
- Tabelle hätte 465 Zeilen