

# Klausur Grundgebiete der Informatik 1

SS 09

(BPO07/DPO04)

Datum: 15.09.2009

*Keine offizielle Lösung!*

<b>Name:</b>	.....
<i>(in Druckschrift)</i>	
<b>Matr.Nr.:</b>	.....
<b>Unterschrift:</b>	.....

*Keinerlei Gewähr auf Richtigkeit!*

Aufgabe	Max. Punkte	Korrektur		Einsicht	
		Punkte	Kürzel	Punkte	Kürzel
1	10				
2	9				
3	12				
4	5				
5	19				
6	12				
7	13				
8	20				
$\Sigma$	100				

Aufgabe 1 – Wissensfragen

Beantworten Sie die folgenden Fragen möglichst prägnant!

- (a) Gegeben sei das folgende Program. Geben Sie die Ausgabe des Programms an:

```

1 #include <stdio.h>
2
3 int A[6] = {5, 4, 2, 3, 1, 0};
4
5 int main()
6 {
7     int *p;
8     p = &A[2];
9     printf("%d", *(p+2));
10    return 0;
11 }
```

Ausgabe: 1

- (b) Gegeben sei folgende Aussage: BubbleSort  $\in O(n^4)$ . Ist diese Aussage korrekt? Gibt es eine schärfere obere Schranke?

*ist korrekt, aber schärfere obere Schranke wäre z.B.  $O(n^2)$*

- (c) Welche Komplexität weist die Binäre Suche auf und was ist die Voraussetzung um diesen Algorithmus einzusetzen?

*$O(\log n)$ , Voraussetzung: sortiertes Feld*

- (d) Was berechnet der Kruskal Algorithmus?

*berechnet minimalen Spannbaum*

- (e) In der Vorlesung haben Sie zwei verschiedene Darstellungsformen für Graphen kennengelernt. Welche dieser Arten hat im Normalfall (d.h. für  $|E| \ll |V|^2$ ) den geringeren Speicherbedarf?

*Adjazenz-liste*

- (f) Wie heißt der in der Vorlesung vorgestellte Sortieralgorithmus, der nur benachbarte Elemente miteinander vertauscht?

*Bubble sort*

- (g) Wie bezeichnet man einen Algorithmus, der bei der Suche nach einer optimalen globalen Lösung in jedem Schritt stets die beste lokale Möglichkeit wählt?

*Greedy*

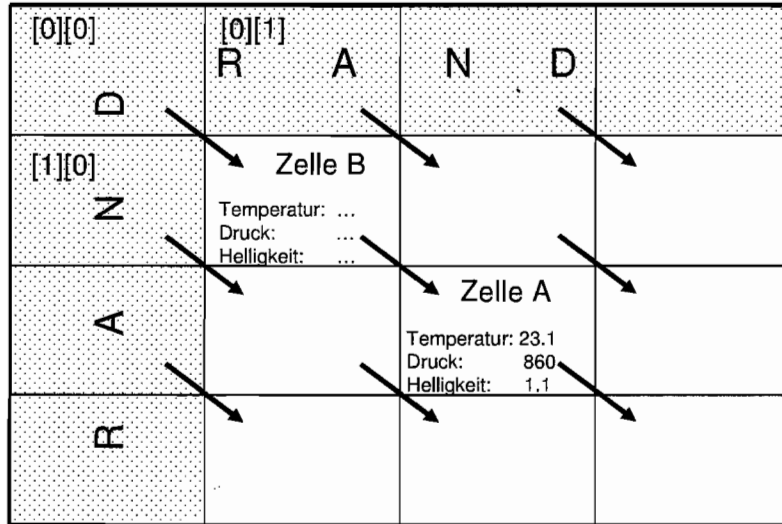
- (h) Welchen Fehler verursacht das Dereferenzieren eines NULL Zeigers bei Ausführung in einem modernen Betriebssystem?

*Unbekannte Ausnahme:*

*Zugriffsverletzung beim Schreiben an Position  $0 \times 00000000$ .*

Aufgabe 2 – Arrays

Nehmen Sie an, dass für die Modellierung der Temperatur, des Drucks und der Helligkeit eines Raumes ein 2-dimensionales Array benötigt wird. Jede Zelle dieses Arrays wird anhand einer Struktur mit 3 Fließkommazahlen repräsentiert (siehe Abbildung).



(a) Vervollständigen Sie die Definition der Struktur für eine Zelle

```
typedef struct zelle
{
```

*float Temperature, Pressure, Brightness;*

```
} Zelle;
```

- (b) Nehmen Sie weiter an, dass physikalisch gesehen der Zustand (Temperatur, Druck und Helligkeit) einer Zelle nur vom Zustand der Zelle links oberhalb ihrer selbst abhängt (siehe z.B. die Abhängigkeit der Zelle A in der Abbildung von der Zelle B). Die Gleichungen für die Berechnung eines neuen Zustandes sind:

$$\begin{aligned}Temp_A &= Temp_A + 0.7 \cdot Temp_B \\ Druck_A &= Druck_A + 0.3 \cdot Druck_B \\ Hell_A &= Hell_A + 0.2 \cdot Hell_B\end{aligned}$$

Schreiben Sie eine Funktion, die den Zustand einer Zelle berechnet. Nehmen Sie an, dass diese Zelle sich nicht am Rand befindet. Die Funktion aktualisiert in einem globalen Array Raum die Zelle mit den Koordinaten  $(x, y)$ .

```
#define N 100
Zelle Raum[N][N];
...
```

```
void UpdateCell(unsigned int x, unsigned int y) {
```

```
    Raum[x][y].Temperature += Raum[x-1][y-1].Temperature * 0.7;
    Raum[x][y].Pressure += Raum[x-1][y-1].Pressure * 0.3;
    Raum[x][y].Brightness += Raum[x-1][y-1].Brightness * 0.2;
```

```
}
```

- (c) Schreiben Sie eine Funktion, die alle Zellen des Arrays `Raum` aktualisiert. Durchlaufen Sie das Array dabei von links nach rechts und von oben nach unten. Die Zellen die sich am Rand befinden, sollen nicht aktualisiert werden.

```
void UpdateAllCells(){  
    int i, j;  
    for (i=1; i<N; i++)  
        for (j=1; j<N; j++)  
            UpdateCell(i, j);  
  
}
```

Aufgabe 3 – Fehleranalyse

In jedem der folgenden ANSI-C Programme befinden sich Fehler. Diese Fehler können verschiedener Art sein. Tragen Sie die Fehler in der folgenden Tabelle ein.

Geben Sie in der zweiten Spalte die Zeile an. Beschreiben Sie in der dritten den Fehler und geben Sie in der vierten oder fünften Spalte an, ob der Fehler zur Zeit des Kompilierens oder zur Ausführungszeit auftritt, indem Sie die zutreffende Spalte mit einem X markieren.

Teil	Zeile(n)	Beschreibung des Fehlers	Laufzeit	Compile Zeit
a	5/9	Hinter geschlossenen schließenden Klammern von Funktion kein Semikolon!		X
b	4	Abbruchbedingung fehlt $\Rightarrow$ Endlosrekursion	X	
c	5	a zeigt auf einen Heap, von nicht	X	
d	15	Division durch 0	X	
	16	integer mit %f ausgegeben	X	
e (1)	5	# vor Präprozessoranweisung <del>fehlt</del>		X
e (2)	3/4	Strings der Wochentage müssen in "		X

(a) Programm:

```

1 #include <stdio.h>
2 double funcl(double p){
3     double res=p*1000;
4     return res;
5 };
6 float funcl(float p){
7     float res=p*1000;
8     return res;
9 };
10 int main() {
11     double a=3;
12     double b;
13     b=a+funcl(a);
14     printf("result= %f", b);
15     return 0;
16 }

```

(b) Programm:

```
1 #include <stdio.h>
2 int factorial(int number) {
3     int res;
4     res = number * factorial(number - 1);
5     return res;
6 }
7
8 void main(void) {
9     int number=5;
10    printf("Factorial of %d is %d", number,
11          factorial(number));
12 }
```

(c) Programm:

```
1 #include <stdio.h>
2 int * a, b;
3 int main() {
4     b=5;
5     *a=b;
6     printf("result = %d", *a);
7     return 0;
8 }
```

(d) Programm:

```
1 #include <stdio.h>
2 #define DIM 5
3 int min(int arr[DIM]){
4     int min=arr[0];
5     int i;
6     for (i=1; i<DIM; i++){
7         if (arr[i]<min){
8             min=arr[i];
9         }
10    }
11    return min;
12 }
13 int main(){
14    int arr[DIM]={0, 1, 2, 3, 4};
15    int i = 100/min(arr);
16    printf ("result= %f", i) ;
17 }
```

(e) Programm:

```
1 #include <stdio.h>
2 typedef enum {mon,tue,wed,thu,fri,sat,sun} days;
3 char *theday[] = {monday, tuesday, wednesday,
4                  thursday, friday, saturday, sunday};
5 define WEEKSTART theday[mon]
6 int main()
7 {
8     printf("week start from %s", WEEKSTART);
9     return 0;
10 }
```

Aufgabe 4 – Funktionen und Module

Betrachten Sie das folgende Programm:

```

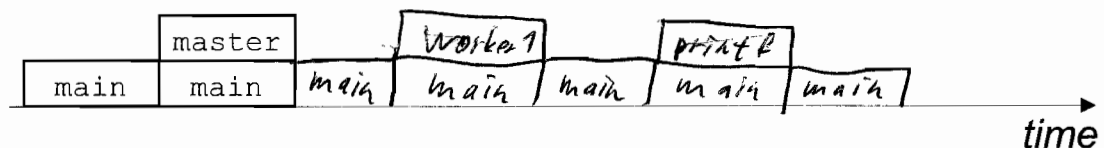
1 #include <stdio.h>
2
3 typedef int (*f)(int);
4
5 int worker1(int input)
6 {
7     return input + 100;
8 }
9
10 int worker2(int input)
11 {
12     return input + 500;
13 }
14
15 f master(int input)
16 {
17     if (input > 10)
18         return worker1;
19     else
20         return worker2;
21 }
22
23 int main()
24 {
25     printf("%d\n", master(12)(8));
26 }

```

(a) Welche Ausgabe erzeugt das Programm?

108

(b) Die folgende Skizze zeigt den Call Stack der Funktion nach den ersten beiden Ausführungsschritten. Vervollständigen Sie die Zeichnung, so dass sie den Call Stack des gesamten Programms darstellt.



Aufgabe 5 – Laufzeitanalyse

(a) Berechnen Sie mit Hilfe der folgenden Rechenregeln der O-Notation

$$(1) O(c) = O(1)$$

$$(2) O(c * f(n)) = O(f(n))$$

$$(3) O(f(n)) + O(f(n)) = O(f(n))$$

$$(4) O(O(f(n))) = O(f(n))$$

$$(5) g(n) = a_k * n^k + a_{k-1} * n^{k-1} + \dots + a_0 \Rightarrow O(g(n)) = O(n^k)$$

$$(6) O(f(n)) * O(g(n)) = O(f(n) * g(n))$$

$$(7) O(f(n)) + O(g(n)) = O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$$

das Laufzeitverhalten der folgenden Funktionen. Geben Sie dabei für jeden Schritt die verwendete Rechenregel an.

(i)  $5k + 32k^2 + k^3$

$$O(5k + 32k^2 + k^3) \stackrel{(5)}{=} O(k^3)$$

(ii)  $(k+3)(2k+k^2) = 2k^2 + 3k^3 + 6k + 3k^2 = 3k^3 + 5k^2 + 6k$

$$O(3k^3 + 5k^2 + 6k) \stackrel{(5)}{=} O(k^3)$$

(iii)  $100k + 3k^7 + 6k^2 + 2^k$

$$O(100k + 3k^7 + 6k^2 + 2^k) \stackrel{(7)}{=} O(2^k)$$

- (b) Mittels einer Taylorreihenentwicklung lassen sich viele Funktionen als Potenzreihen approximieren. Im Falle der Funktion  $f(x) = e^x$  sieht die Potenzreihe wie folgt aus:

$$f(x) = \sum_{n=0}^k \frac{x^n}{n!} \quad (1)$$

In C ist diese Näherung unter Verwendung der Funktionen `Factorial`, `Pow` und `Exp` realisiert.

```

1  double Factorial (unsigned int n)
2  {
3      unsigned int i;
4      double result;
5
6      result = 1;
7
8      for (i = 1; i <= n ; i++)
9          result *=i;
10
11     return result;
12 }
13
14 double Pow(double base, unsigned int n)
15 {
16     unsigned int i;
17     double result;
18
19     result = 1.0;
20
21     for (i = 0; i < n; i++)
22         result *= base;
23
24     return result;
25 }
26
27 double Exp(double x, unsigned int k)
28 {
29     unsigned int n;
30     double result;
31
32     result = 0.0;
33
34     for (n = 0; n <= k; n++)
35         result += (Pow(x,n) / Factorial(n));
36
37     return result;
38 }

```

- (i) Bestimmen Sie unter Verwendung der O-Notation die Komplexität der Funktionen `Factorial()`, `Pow()` und `Exp()`.

*Factorial:  $O(n)$*

*Pow:  $O(n)$*

*Exp:  $O(k^k)$*

- (ii) Die Funktion `Exp()` lässt sich effizienter implementieren, indem man die Funktionalität von `Pow()` und `Factorial()` direkt in eine neue Funktion `Exp_fast()` integriert. Der iterative Charakter der Funktion `Exp()` erlaubt eine effiziente Bestimmung von  $x^n$  und  $n!$  unter der Zuhilfenahme der vorher bestimmten Werte.

Tipps:

- $n! = (n - 1)! * n$  und  $0! = 1$
- $a^b = a^{(b-1)} * a$  und  $a^0 = 1$

Verbessern Sie die Funktion `Exp_fast()` unter Berücksichtigung der oben genannten Hinweise.

```
double Exp_fast(double x, unsigned int k)
{
```

```
    unsigned int n;
    double factorial, pow, exp result;
    factorial = 1; pow = 1; result = 0.0;
    for (n=0; n <= k; n++) {
        if (n > 0)
            factorial *= n;
            pow *= x;
            result += pow / factorial
    }
    return result;
}
```

- (iii) Bestimmen Sie die Komplexität der verbesserten Funktion `Exp_fast()`.

*$O(k)$*

Aufgabe 6 – Sortieralgorithmen: Quicksort

- (a) Betrachten Sie den in der Vorlesung vorgestellten Quicksort Algorithmus. Welches Laufzeitverhalten weist dieser im schlechtesten Fall auf?

$$O(n^2)$$

- (b) Gegeben seien die folgenden Zahlen:

{3, 2, 4, 5, 1}

Geben Sie eine Vorsortierung an, die bei der Verwendung von Quicksort den schlechtesten Fall hervorruft.

{1, 2, 3, 4, 5}

- (c) Der Quicksort Algorithmus benutzt die Funktion `exchange` um zwei Werte in einem Feld zu vertauschen. Implementieren Sie diese.

```
void exchange(int *i, int *j)
{
```

```
    int temp;
```

```
    temp = *i;
```

```
    *i = *j;
```

```
    *j = temp;
```

```
}
```

(d) Die folgende Tabelle zeigt die Inhalte eines Feldes vor und nach der Ausführung des Quicksort Algorithmus. Führen Sie in den freien Zeilen den Algorithmus so aus, wie er in der Vorlesung vorgestellt wurde. Beachten Sie dabei die folgenden Hinweise:

- Nutzen Sie jeweils den rechten Rand des zu partitionierenden Bereichs als Trennelement.
- Unterstreichen Sie vor jedem Aufruf von `partition` den Bereich, der diesem Aufruf übergeben wird.
- Tragen Sie den Zustand des Arrays nach jedem Aufruf von `exchange` in die nächste Zeile der Tabelle ein.
- Markieren Sie in jeder Zeile die Elemente, die zuvor vertauscht worden sind.

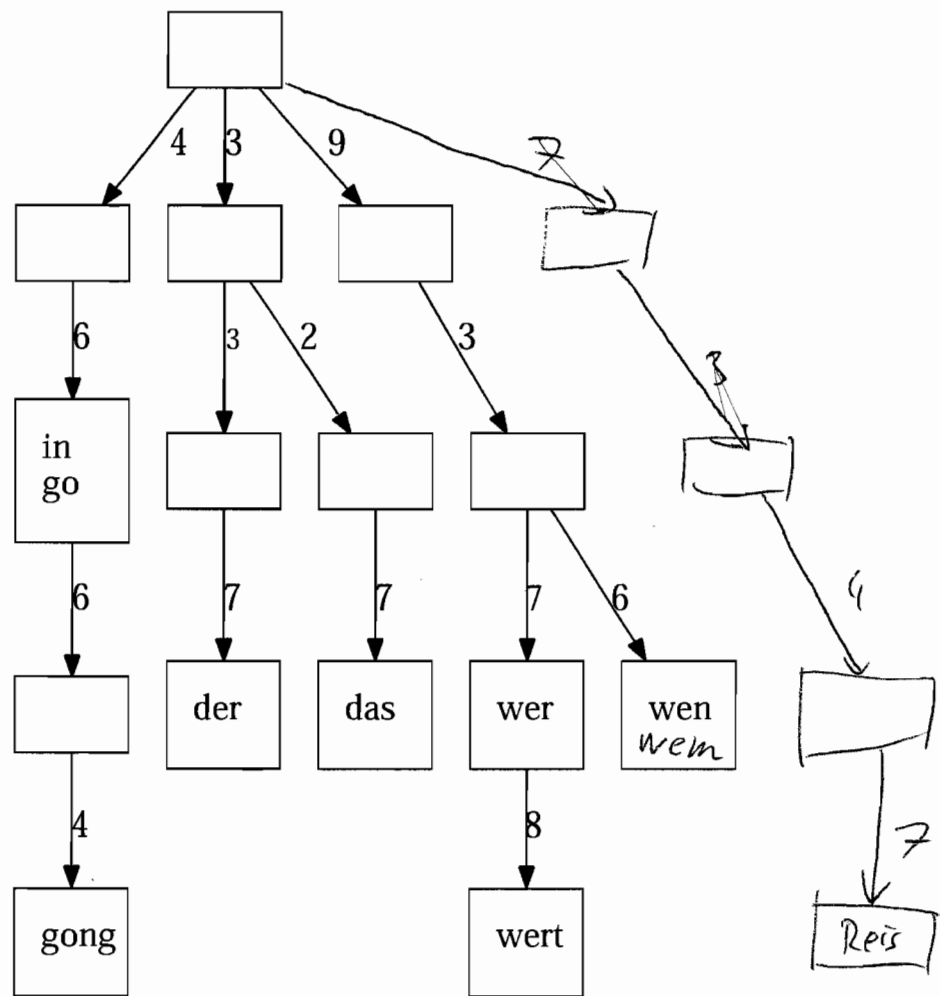
5	7	15	10	3	4	6
5	④	15	10	3	⑦	6
5	4	③	10	⑩	7	6
5	4	3	⑥	15	7	⑩
③	4	⑤	6	15	7	10
3	4	5	6	15	7	10
3	4	5	6	⑦	⑩	15
3	4	5	6	7	⑩	⑩
3	4	5	6	7	10	15

**Aufgabe 7 – Bäume**

Moderne Telefone gestatten die Eingabe von Text, z.B. für SMS, mit Hilfe der Zifferntastatur. Dabei werden jeder Ziffer mehrere Buchstaben zugeordnet, wie im Folgenden zu sehen ist:

1	2 ABC	3 DEF
4 GHI	5 JKL	6 MNO
7 PQRS	8 TUV	9 WXYZ
*	0	#

Oftmals kommt dabei ein Wörterbuch zum Einsatz, welches für eine gegebene Ziffernfolge die daraus generierbaren Vokabeln anzeigt. Um beispielsweise das Wort *wer* einzugeben muss der Benutzer die Ziffernfolge *937* tippen. Die Aufgabe der Software ist es nun, aus der Eingabe *937* die Vokabel *wer* zurückzugewinnen. Eine Möglichkeit diese Funktionalität zu implementieren ist es, die Daten in einem Baum wie dem folgenden zu speichern:



Die Knoten beinhalten eine Liste der Vokabeln in diesem Knoten und die Kanten sind mit der Taste indiziert, die gedrückt werden muss, um in diesen Knoten zu gelangen.

Das folgende Code Fragment zeigt die Datenstruktur eines Baumes der zur Implementierung eines Wörterbuchs für eine solche Texteingabe verwendet werden kann:

```

1 #define N_CHILDREN 10
2 typedef struct node_s {
3     wordlist_p matches;
4     struct node_s* children [N_CHILDREN];
5 } T9node, *T9node_p;

```

Dabei enthält der Eintrag `children[i]` einen Zeiger auf das Kind, das durch Eingabe der Ziffer `i` gewählt wird und den Wert `NULL` falls diese Kante nicht im Wörterbuch existiert. Die Tasten `*` und `#` liefern die Konstante `EOF`  $\notin [0, 9]$  und zeigen an, dass der Benutzer die Eingabe beendet hat.

Weiterhin ist folgende Funktion zur Ausgabe der in einem Knoten enthaltenen Liste `matches` gegeben:

```

1 /* Gibt den Wert aller Eintraege der Liste auf stdout aus. */
2 void wordlist_print(wordlist_p list);

```

### Aufgaben

Gehen Sie bei allen folgenden Aufgaben davon aus, dass nicht zwischen Groß- und Kleinschreibung unterschieden wird.

- (a) Geben Sie die Ziffernfolgen für die Vokabeln *Reis* und *wem* an.

*Reis: 7 → 3 → 4 → 7      wem: 9 → 3 → 6*

- (b) Fügen Sie die Vokabeln *Reis* und *wem* in das angegebene Wörterbuch ein. Nutzen Sie dazu den in der Skizze vorhandenen Platz.
- (c) Die Höhe eines Baumes ist definiert als die Anzahl der Kanten auf dem längsten Pfad des Baumes. Geben Sie die maximale Höhe dieses Baums für den Fall eines beliebigen Wörterbuches an.

*int hoehe = ~~max~~ max (Duden);*

- (d) Die Aufgabe einer Suche in diesem Baum ist es, an Hand einer Ziffernfolge die gesuchten Wörter aus dem Wörterbuch zu suchen, in dem der Baum geeignet durchlaufen wird. Diese Suche geschieht, indem abhängig von der gedrückten Taste ein Nachfolger des aktuellen Knotens gewählt wird.

Im folgenden ist das Skelett einer Funktion gegeben, welche die Funktion `int keypress()` benutzt, um den nächsten Tastendruck vom Benutzer zu erhalten. `keypress()` gibt einen Wert  $\in [0, 9]$  zurück, falls die entsprechende Ziffer eingegeben wurde. Falls der Benutzer die Eingabe beendet hat wird die Konstante `EOF` zurück gegeben.

Erweitern Sie das folgende Codefragment geeignet, um die oben beschriebene Suche zu implementieren. Wenn der Benutzer eine Eingabe getätigt hat, aus der keine gültige Vokabel mehr entstehen kann, geben Sie eine Fehlermeldung aus und brechen ab. Andernfalls geben Sie nach Ende der Eingabe alle gültigen Vokabeln aus.

**void**

```
search(T9node_p t9){
    int c = 0;
    /* Weitere Deklarationen */
```

```

    c = keypress();
    while ( c != EOF )
    {
        if (t9->t9-> children[c] == NULL) {
            break; printf ("Error 303: Wörter ausgegangen!");
            break;
        }
        else t9 = t9-> children[c];
    }

```

```

    c = keypress();
}
printf("Passende Vokabeln: ");
wordlist-printprint(t9->ma tches);
return;
}
```

**Aufgabe 8 – Techniken zum Algorithmenentwurf**

Gegeben sei die folgende rekursive Definition einer Funktion von den natürlichen in die ganzen Zahlen:

$$\begin{aligned} f(i) &= 0 && \text{für } i \leq 0 \\ f(1) &= 1 \\ f(i) &= f(i-1) + f(i-3) && \text{für } i \geq 2 \end{aligned}$$

- (a) Implementieren Sie die Funktion  $f$  rekursiv in C passend zu folgender Funktionsdeklaration:

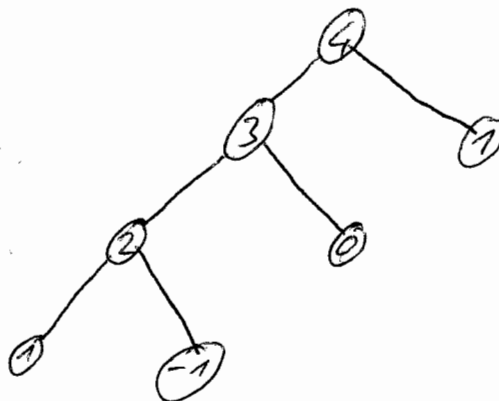
```
int rekursiv(int i);
```

Geben Sie eine einfache und kurze Implementierung an.

```
int rekursiv(int i) {
    if (i <= 0) return 0;
    if (i == 1) return 1;
    return rekursiv(i-1) + rekursiv(i-3);
}
```

- (b) Skizzieren Sie den Baum mit den rekursiven Aufrufen der Funktion `rekursiv()`, wenn der initiale Aufruf `rekursiv(4)` lautet:

- Bezeichnen Sie die Knoten mit dem Wert des Parameters  $i$ .
- Beginnen Sie mit Knoten 4 als Wurzel.
- Knoten  $j$  ist Sohn von Knoten  $i$ , wenn `rekursiv(j)` von `rekursiv(i)` aufgerufen wird.



- (c) Da die rekursive Implementierung viele Berechnungen mehrfach ausführt, soll nun eine iterative Implementierung der Funktion  $f$  erstellt werden.

Überlegen Sie sich dazu zunächst, wie aus bereits vorhandenen Funktionswerten für kleinere Parameter neue Funktionswerte bestimmt werden können.

Bestimmen Sie die beiden fehlenden Funktionsergebnisse in der folgenden Tabelle und umkreisen Sie die Funktionsergebnisse, die Sie gemäß Rekursionsvorschrift für die Berechnung benutzen müssen.

$$-1=0 \quad 0=0 \quad 1=1 \quad 2=1 \quad 3=1$$

Parameter $i$	...	4	5	6	7	8	9	10	11	12	13
Funktionsergebnis $f(i)$	...	2	3	4	6	9	13	19	28	41	60

- (d) Nun soll die iterative Implementierung der Funktion  $f$  in C passend zu folgender Funktionsdeklaration erstellt werden:

```
int iterativ(int i);
```

```
int iterativ(int i) {
```

```
    int a[] = {0, 0, 1}; int erg;
```

```
    if (i > 0) erg = 1; else erg = 0;
```

```
    for (; i > 1; i--) {
```

```
        erg = a[0] + a[2];
```

```
        a[0] = a[1]; a[1] = a[2]; a[2] = erg;
```

```
    }
```

```
    return erg;
```

```
}
```