

# Algo-Übungsklausur

## Aufgabe 1:

- a) Erkläre die Begriffe Terminiert, Determiniert und Determinismus
- b) Gib die Reihenfolge an, in der die Werte ausgegeben werden, wenn bei einem LIFO-Prinzip folgende Funktionen der Reihe nach aufgerufen werden:

in(5) – in(3) - in(7) - out() - out() - in(6) - out() – in(8) - in(4) - out()

Die Funktion in() fügt dabei den Wert in der Klammer in die Liste ein, out() gibt den ersten Wert in der Liste aus und **löscht** diesen aus der Liste.

- c) Was bedeutet AVL-Eigenschaft? Zeichne einen Beispielbaum mit der AVL-Eigenschaft und einen, der die AVL-Eigenschaft nicht besitzt!

d)

```
void rechne(int a, int* b)
```

```
{  
    a = a*4;  
    *b = *b + a;  
    cout << a << ";" << *b << " ";  
    b = &a;  
    cout << *b << endl;  
}
```

```
main()
```

```
{  
    int x = 2;  
    int y = 5*x;  
    int z = 4;  
    cout << x << ";" << y << ";" << z << endl;  
    rechne(x, &z);  
    cout << x << ";" << y << ";" << z << endl;  
    rechne(y, &x);  
    getch();  
}
```

- e) Fibonaccizahlen zeichnen sich dadurch aus, dass die n-te Fibonaccizahl die Addition der beiden Vorgänger ist. Die ersten beiden Fibonaccizahlen sind als 1 definiert. Somit ist die 3.Fibonaccizahl die 1. + die 2. Fibonaccizahl, also  $1+1 = 2$ . Die Reihe der Fibonaccizahlen sieht also wie folgt aus: 1, 1, 2, 3, 5, 8, 13, 21, 34....

Schreibe die Funktion `int fib(int n)`, die die n-te Fibonaccizahl zurückgibt.  
(Tipp: Das Anlegen eines Arrays wäre hier von Vorteil)

## Aufgabe 2:

a) Was versteht man unter einer rekursiven Funktion und was darf bei einer rekursiven Funktion nicht fehlen?

b) Was gibt diese Funktion aus?

```
void zahl(int n)
{
    if(n < 1) return;

    cout << n << endl;
    zahl(n-1);
}
```

c) Schreibe die Funktion so um, dass die Reihenfolge der Ausgabe umgekehrt wird.

d) Schreibe die **rekursive** Funktion **int** quersumme(**int** zahl), die die Quersumme der übergebenen Zahl n zurückgibt.

e) Was ist beim Bucketsort bezüglich der zu sortierenden Werte zu beachten?

f) Welche **Laufzeitkomplexität** haben folgende Algorithmen?

Bucketsort:

Bubblesort:

Quicksort:

g) Welche Laufzeitkomplexität haben wir beim Suchen eines Elementes in einer Liste?

h) Schreibe die Funktion **void** loesche\_Nachfolger(Element\* p), die den Nachfolger des Elements, auf das p zeigt, löscht.

### Aufgabe 3:



- 0 = Schleswig Holstein
- 1 = Mecklenburg Vorpommern
- 2 = Niedersachsen
- 3 = Sachsen Anhalt
- 4 = Berlin
- 5 = Nordrhein Westfalen
- 6 = Hessen
- 7 = Thüringen
- 8 = Sachsen
- 9 = Rheinland Pfalz
- 10 = Saarland
- 11 = Baden-Württemberg
- 12 = Bayern

a) Schreibe eine Adjazenzmatrix auf, die darstellt, ob Bundesländer benachbart sind oder nicht. Dabei entsprechen die Länder den oben zugeordneten Zahlen. Eine 1 im Tabelleneintrag soll bedeuten, dass eine Nachbarschaft vorliegt, eine 0 bedeutet das Gegenteil.

b) Die Strukturen für die Adjazenzliste seien Folgende:

```
struct Bundesland{
    string name;
    Benachbart* Liste;
    Bundesland* next;
};

struct Benachbart{
    Bundesland* Nachbar;
    Benachbart* next;
};
```

Zeichne die zugehörige Adjazenzliste, zeichne aber nur die Adjazenzen für Schleswig Holstein und Sachsen ein.

c) Zurück zur Adjazenzmatrix: Schreibe die Funktion **int** anz\_Adjazenzen(**int** x), die für eine gegebene Stadt x prüft, wie viele Nachbarländer sie hat und diese Anzahl zurückgibt.

d) Schreibe die Funktion **void** Laender\_mit\_x\_Bundesländern(**int** x). Diese Funktion soll alle Bundesländer durchgehen und die Bundesländer, die x Nachbarn haben **ausgeben**(= auf die Konsole hinschreiben, NICHT **return**, sondern **cout**). Im Programm sind die Adjazenzen im **global definierten** Array **int** graph[N][N]; eingespeichert.

**Viel Erfolg!!!**