

Mehrere Parameter

- Wir haben bisher an die Funktion einen Parameter übergeben und haben einen Wert zurückgeliefert.
- Man kann an eine Funktion mehrere Parameter übergeben.
- Diese können auch verschiedene Typen haben. Die Typen der Werte beim Benutzen der Funktion müssen aber genau mit den Typen bei der Definition übereinstimmen.
- Es ist nicht möglich, mehr als einen Wert zurückzuliefern.
- Beispiel: Funktion, die das Maximum zweier Zahlen ermittelt und die größte zurückliefert.

```

5 int liesInt( string frage ) {
6     int antwort;
7     cout << frage << ": ";
8     cin >> antwort;
9     return antwort;
10 }
11
12 int maximum( int a, int b ) {
13     int groesste;
14     if( a > b ) {
15         groesste = a;
16     } else {
17         groesste = b;
18     }
19     return groesste;
20 }
21
22 int main() {
23     int z1, z2, z3, z4;
24     z1 = liesInt( "Zahl1" );
25     z2 = liesInt( "Zahl2" );
26     z3 = liesInt( "Zahl3" );
27     z4 = liesInt( "letzte Zahl" );
28
29     cout << "maximum von " << z1 << " und " << z2 << ": " <<
30         maximum(z1,z2) << endl;
31     cout << "maximum aller Zahlen: " <<
32         maximum( maximum(z1,z2), maximum(z3,z4 ) ) << endl;
33 }

```

Ausführungsreihenfolge der Zeilen:

22 – 23 – 24 (der Parameter frage von liesInt bekommt den Wert „Zahl1“) – 5 – 6 – 7 – 8 – 9 (liesInt liefert den Wert der Variablen antwort zurück) – 25 (der Parameter frage von liesInt bekommt den Wert „Zahl2“) – 5 – 6 – 7 – 8 – 9 (liesInt liefert den Wert der Variablen antwort zurück) – 26 (der Parameter frage von liesInt bekommt den Wert „Zahl3“) – 5 – 6 – 7 – 8 – 9 (liesInt liefert den Wert der Variablen antwort zurück) – 27 (der Parameter frage von liesInt bekommt den Wert „letzte Zahl“) – 5 – 6 – 7 – 8 – 9 (liesInt liefert den Wert der Variablen antwort zurück) – 29/30 (der Parameter a von maximum bekommt den Wert von z1, der Parameter b von maximum bekommt den Wert von z2) – 12 – 13 – 14 – (15 oder 17) – 19 (maximum liefert den Wert der Variablen groesste zurück) – 31/32 (der Parameter a von maximum bekommt den Wert von z1, der Parameter b von maximum bekommt den Wert von z2) – 12 – 13 – 14 – (15 oder 17) – 19 (maximum liefert den Wert der Variablen groesste zurück) – immer noch 31/32 – (der Parameter a von maximum bekommt den Wert von z3, der Parameter b von maximum bekommt den Wert von z2) – 12 – 13 – 14 – (15 oder 17) – 19 (maximum liefert den Wert der Variablen groesste zurück) – immer noch 31/32 – (der Parameter a von maximum bekommt den vorher zurückgelieferten Wert von maximum(z1,z2), der Parameter b von maximum bekommt den vorher zurückgelieferten Wert von maximum(z3,z4)) – 12 – 13 – 14 – (15 oder 17) – 19 (maximum liefert den Wert der Variablen groesste zurück) – 33

Noch ein Beispiel: größter gemeinsamer Teiler

```
#include <iostream>
using namespace std;

int ggT( int a, int b ) {
    while( a != b ) {
        if( a > b ) {
            a = a - b;
        } else {
            b = b - a;
        }
    }
    return a;
}

int main() {
    cout << ggT( 24, 9 ) << endl;
    cout << ggT( 36, 342 ) << endl;
    cout << ggT( 4588, 3747 ) << endl;
}
```

Anmerkungen am Rande ...

- Man kann auch die eingebauten Rechenoperatoren als Funktionen verstehen. Diese werden jedoch anders geschrieben (nämlich so, wie wir es gewohnt sind, der sog. infix-Notation).

Den Ausdruck

$$2+3*4$$

würde man in der Funktionsschreibweise (sog. präfix-Notation)

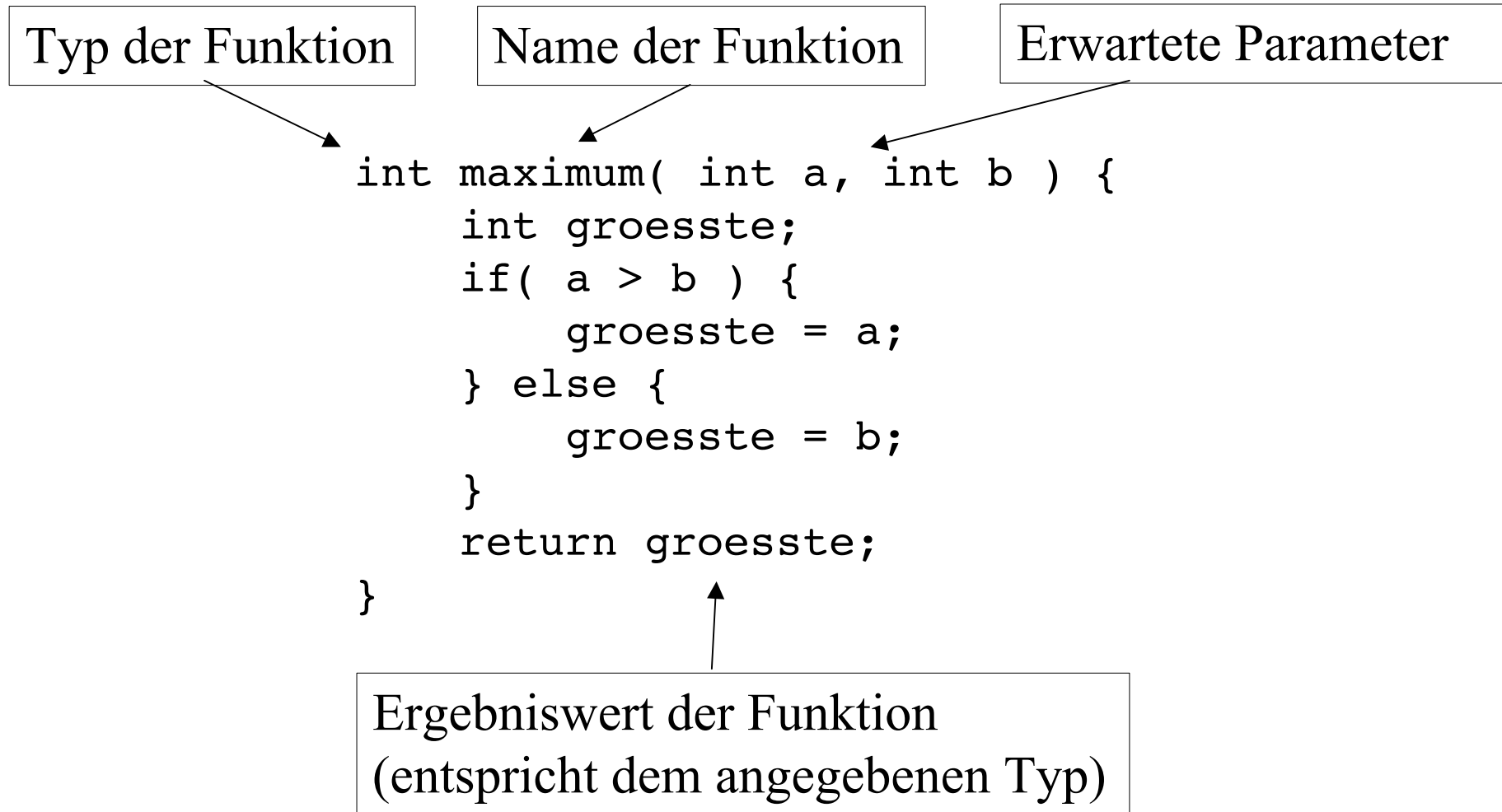
$$+(2,*(3,4))$$

schreiben.

Einige HP-Taschenrechner benutzen die postfix-Notation:

$$3\ 4\ *\ 2\ +$$

Übersicht Funktionsdefinition



Softwareerstellung mit Funktionen

- Mit Funktionen lassen sich große Aufgaben *schrittweise verfeinern*.
- Man beschreibt grob den Algorithmus zur Lösung des Problems durch einige Funktionen.
- Danach formuliert man die Funktionen aus, ggfls. indem man wieder Funktionen benutzt, usw.
- Irgendwann besteht eine Funktion nur noch aus Anweisungen und ist damit „fertig“.

Mehrdimensionale Felder

- Die bisher vorgestellten Felder waren eindimensional:

```
int feld[10];
```

- Felder können weitere Dimensionen besitzen, z.B. Zweidimensional:

```
int matrix[10][8];
```

- Vorstellbar als Tabelle (wie z.B. in Excel)
- Zugriff durch Angabe beider Indexe
- Beliebig viele weitere Dimensionen möglich
 - 3-dimensionale Koordinaten/Vektoren
 - Preislisten
 - ...

Netzplantechnik (stark vereinfacht)

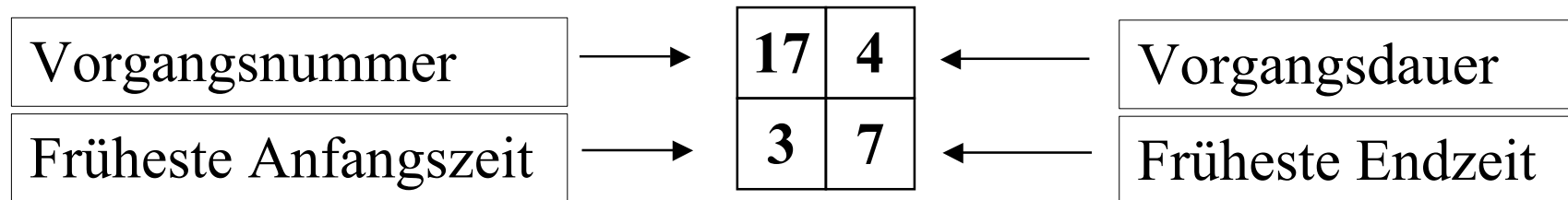
Ein **Netzplan** ist die “graphische oder tabellarische Darstellung von Abläufen und der Abhängigkeiten.” (DIN 69900, Teil 1)

Netzplantechnik umfasst “alle Verfahren zur Analyse, Planung, Steuerung und Überwachung von Abläufen auf der Grundlage der Graphentheorie, wobei Zeit, Kosten, Einsatzmittel bzw. Ressourcen und weitere Einflussgrößen berücksichtigt werden können.”

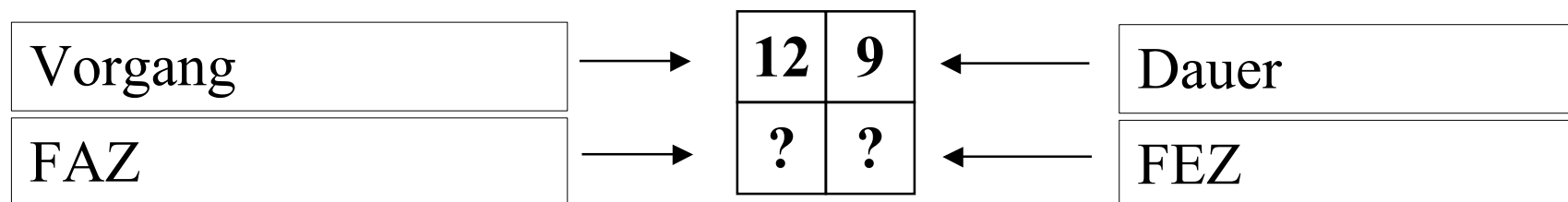
**Netzplantechnik im Bereich der Produktion = PPS
(Produktionsplanung und -steuerung)**

Ein Vorgang

Ein Vorgang ist ein Ablaufelement, das ein bestimmtes Geschehen beschreibt.



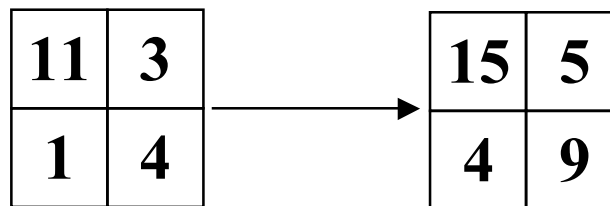
In einem Netzplan können die früheste Anfangszeit (FAZ) und die früheste Endzeit (FEZ) unbekannt sein:



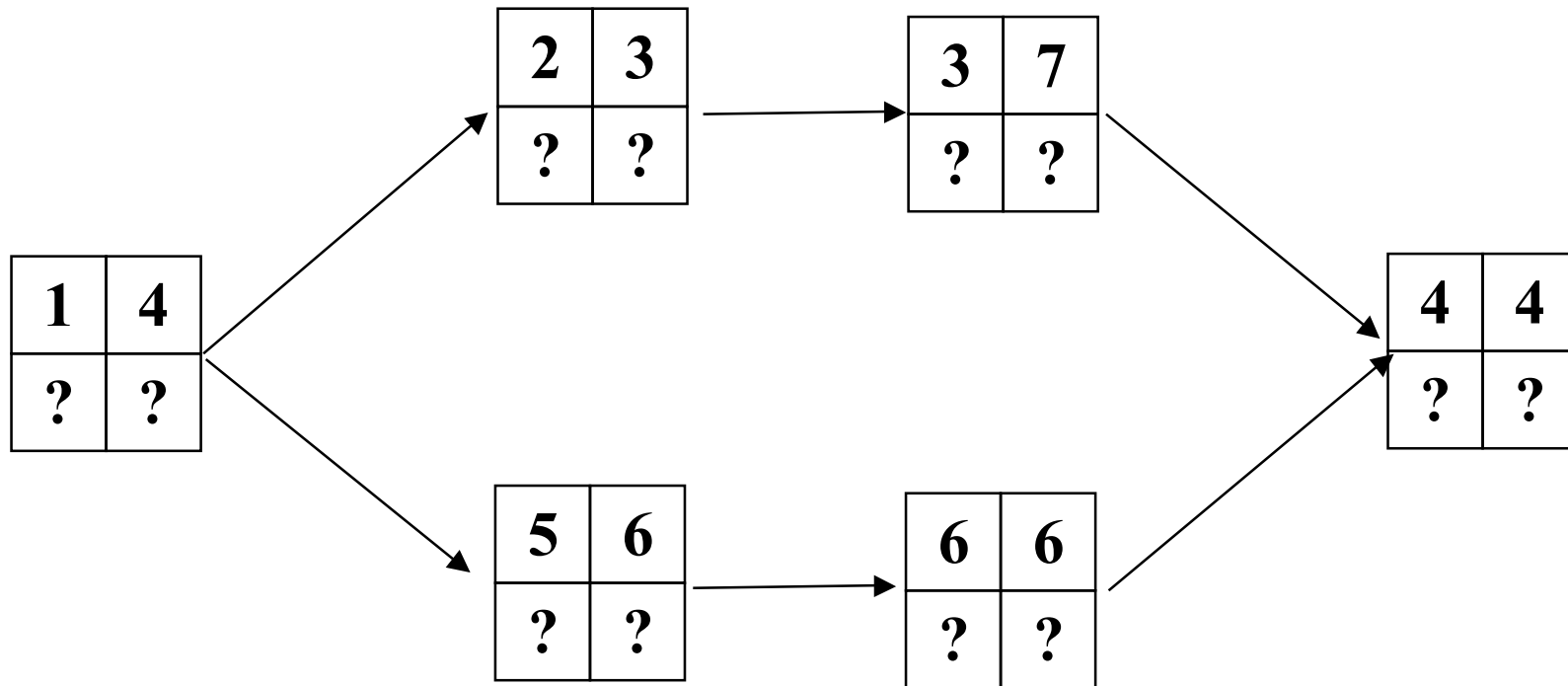
Anordnungsbeziehungen

Eine Anordnungsbeziehung kennzeichnet die logische Abhängigkeit zwischen Vorgängen.

Hier vereinfacht: Wenn ein Vorgang beendet ist, kann der nächste Vorgang begonnen werden.



Ein Netzplan als Grafik



Wenn Vorgang 1 zum Zeitpunkt 1 beginnt, wann ist der letzte Vorgang frühestens beendet?

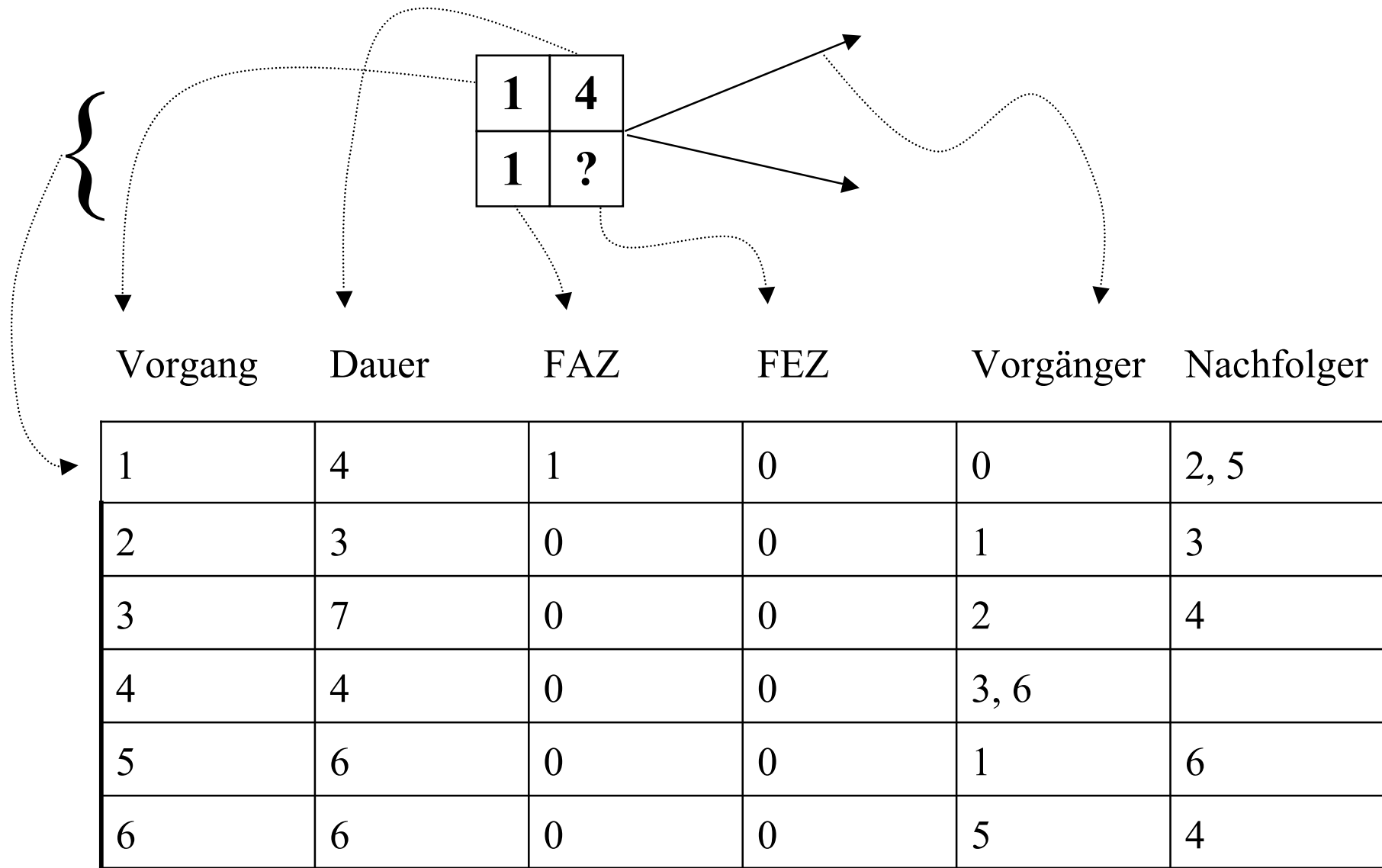
Aufgabe

- Schreiben Sie ein Programm, das für einen gegebenen Netzplan den frühestmöglichen Endzeitpunkt berechnet.

Probleme:

- Wie repräsentieren wir einen Vorgang?
- Wie repräsentieren wir den Netzplan?
- Wie lassen wir den Netzplan eingeben?
- Wie berechnen wir die Start/Endzeitpunkte?

Der Netzplan als Tabelle



Entwurfsentscheidungen

- Auch leere Felder müssen einen Wert haben: Wir wählen den Wert 0, welcher kennzeichnet „hat keinen Wert“.
- Die Spalte „Nachfolger“ ist redundant, daher überflüssig.
- Die Spalte „Vorgänger“ darf nur einen Wert enthalten. Wir fügen weitere Spalten hinzu, um mehrere Vorgänger zuzulassen.
- Wir wählen als Datenstruktur ein 2-dimensionales Feld.
 - In der Breite beschränkt, d.h. nicht beliebig viele Vorgänger
 - In der Höhe beschränkt, d.h. nicht beliebig viele Vorgänge

Der Netzplan als Tabelle

Eigenschaften 

Vorgänge



1	4	1	0	0	0	0	0
2	3	0	0	1	0	0	0
3	7	0	0	2	0	0	0
4	4	0	0	3	6	0	0
5	6	0	0	1	0	0	0
6	6	0	0	5	0	0	0
0	0	0	0	0	0	0	0

Der zentrale Algorithmus

- Setze den Netzplan auf den Ausgangszustand
- Solange es noch Vorgänge ohne FEZ gibt:
 - Wenn ein Vorgang eine FAZ, aber keine FEZ hat, berechne $FEZ = FAZ + \text{Dauer}$
 - Wenn ein Vorgang keine FAZ hat und alle Vorgänger eine FEZ haben, setze FAZ auf das Maximum aller FEZ der Vorgänger
- Ermittle aus allen Vorgängen die maximale FEZ

```

/*
 * Programm zur Berechnung des Groessten FEZ in einem Netzplan
 * (c) Dr. Herwig Henseler, 2008
 * VL EDV3, HS Bremerhaven
 */
#include <iostream>
#include <fstream>
using namespace std;

const int MAX_VORGAENGE = 7;
const int MAX_BREITE    = 8;

// Die Spalten bekommen Namen zur besseren Lesbarkeit
const int VORGANG      = 0;
const int DAUER        = 1;
const int FAZ          = 2;
const int FEZ          = 3;
const int VORGAENGER  = 4; // ab hier die Vorgaenger des Vorgangs

int main() {
    int netzplan[MAX_VORGAENGE][MAX_BREITE];

    netzplan_startzustand(netzplan);
    netzplan_fuellen(netzplan);
    cout << "Groesster FEZ: " << max_fez(netzplan) << endl;
}

```

```

// lies den Startzustand des Netzplanes aus einer Datei netzplan.txt

void netzplan_startzustand(int netzplan[MAX_VORGAENGE][MAX_BREITE]) {
    ifstream datei;
    datei.open( "netzplan.txt" );
    for( int x = 1; x < MAX_VORGAENGE; x++ ) {
        for( int y = 0; y < MAX_BREITE; y++ ) {
            datei >> netzplan[x][y];
        }
    }
    datei.close();
}

```

Inhalt der Datei netzplan.txt:

```

1 4 1 0 0 0 0 0
2 3 0 0 1 0 0 0
3 7 0 0 2 0 0 0
4 4 0 0 3 6 0 0
5 6 0 0 1 0 0 0
6 6 0 0 5 0 0 0

```

```

// Berechne alle noch fehlenden Daten im Netzplan.
// Solange noch ein FEZ fehlt, muessen wir erneut
// alle Vorgaenge betrachten

void netzplan_fuellen(int netzplan[MAX_VORGAENGE][MAX_BREITE]) {
    while( fez_fehlt(netzplan) ) {
        // Schleife ueber alle Vorgaenge
        for( int i = 1; i < MAX_VORGAENGE; i++ ) {

            // FEZ fehlt
            if( netzplan[i][FEZ] == 0 ) {
                netzplan[i][FEZ] = fez( netzplan, i );
            }

            // FAZ fehlt
            if( netzplan[i][FAZ] == 0 ) {
                netzplan[i][FAZ] = vorgaenger_max( netzplan, i );
            }
        }
        // dump( netzplan );
    }
}

```

```
// fehlt in diesem netzplan noch irgendeine FEZ?

bool fez_fehlt(int netzplan[MAX_VORGAENGE][MAX_BREITE]) {
    bool fehlt = false;
    for( int i = 1; i < MAX_VORGAENGE; i++ ) {
        if( netzplan[i][FEZ] == 0 ) {
            fehlt = true;
        }
    }
    return fehlt;
}
```

```
// berechne den FEZ eines Vorganges. Liefert 0, wenn
// der FEZ noch nicht berechnet werden kann

int fez( int netzplan[MAX_VORGAENGE][MAX_BREITE], int i ) {
    int wert = 0;
    if( netzplan[i][FAZ] != 0 ) {
        // FEZ kann ermittelt werden
        wert = netzplan[i][FAZ] + netzplan[i][DAUER];
    }
    return wert;
}
```

```

// schaue nach, ob alle vorgaenger von Vorgang i eine FEZ haben
// liefert 0, wenn mindestens ein Vorgang keine FEZ hat, das
// Maximum aller FEZ sonst
int vorgaenger_max( int netzplan[MAX_VORGAENGE][MAX_BREITE], int i ){
    int vorgaenger_max = 0;
    bool vorhanden = true;
    for( int v = VORGAENGER; v<MAX_BREITE && netzplan[i][v]!=0; v++ ){
        int vorgaenger = netzplan[i][v];
        int vorgaenger_fez = netzplan[vorgaenger][FEZ];
        if( vorgaenger_fez != 0 ) {
            // vorgaenger hat FEZ
            if( vorgaenger_max < vorgaenger_fez ) {
                vorgaenger_max = vorgaenger_fez;
            }
        } else {
            // dieser vorgaenger hat keine FEZ
            vorhanden = false;
        }
    }

    // falls auch nur einer nicht vorhanden ist, geben wir 0 zurueck
    if( !vorhanden ) {
        vorgaenger_max = 0;
    }
    return vorgaenger_max;
}

```

```
// alle FEZ berechnet, Netzplan ist vollstaendig
int max_fez( int netzplan[MAX_VORGAENGE][MAX_BREITE] ) {

    int max_fez = netzplan[1][FEZ];
    for( int i = 2; i < MAX_VORGAENGE; i++ ) {
        if( max_fez < netzplan[i][FEZ] ) {
            max_fez = netzplan[i][FEZ];
        }
    }
    return max_fez;
}
```

```

/*
 * Programm zur Berechnung des Groessten FEZ in einem Netzplan
 * (c) Dr. Herwig Henseler, 2008
 * VL EDV3, HS Bremerhaven
 */
#include <iostream>
#include <fstream>
using namespace std;

const int MAX_VORGAENGE = 7;
const int MAX_BREITE = 8;

// Die Spalten bekommen Namen zur besseren Lesbarkeit
const int VORGANG = 0;
const int DAUER = 1;
const int FAZ = 2;
const int FEZ = 3;
const int VORGAENGER = 4; // ab hier die Vorgaenger des Vorgangs

int main() {
    int netzplan[MAX_VORGAENGE][MAX_BREITE];

    netzplan_startzustand(netzplan);
    netzplan_fuellen(netzplan);
    cout << "Groesster FEZ: " << max_fez(netzplan) << endl;
}

// lies den Startzustand des Netzplanes aus einer Datei netzplan.txt
void netzplan_startzustand(int netzplan[MAX_VORGAENGE][MAX_BREITE]) {
    ifstream datei;
    datei.open( "netzplan.txt" );
    for( int x = 1; x < MAX_VORGAENGE; x++ ) {
        for( int y = 0; y < MAX_BREITE; y++ ) {
            datei >> netzplan[x][y];
        }
    }
    datei.close();
}

// fehlt in diesem netzplan noch irgendeine FEZ?
bool fez_fehlt(int netzplan[MAX_VORGAENGE][MAX_BREITE]) {
    bool fehlt = false;
    for( int i = 1; i < MAX_VORGAENGE; i++ ) {
        if( netzplan[i][FEZ] == 0 ) {
            fehlt = true;
        }
    }
    return fehlt;
}

// berechne den FEZ eines Vorganges. Liefert 0, wenn
// der FEZ noch nicht berechnet werden kann
int fez( int netzplan[MAX_VORGAENGE][MAX_BREITE], int i ) {
    int wert = 0;
    if( netzplan[i][FAZ] != 0 ) {
        // FEZ kann ermittelt werden
        wert = netzplan[i][FAZ] + netzplan[i][DAUER];
    }
    return wert;
}

```

```

// schaue nach, ob alle vorgaenger von Vorgang i eine FEZ haben
// liefert 0, wenn mindestens ein Vorgang keine FEZ hat, das
// Maximum aller FEZ sonst
int vorgaenger_max( int netzplan[MAX_VORGAENGE][MAX_BREITE], int i ) {

    int vorgaenger_max = 0;
    bool vorhanden = true;
    for( int v = VORGAENGER; v < MAX_BREITE && netzplan[i][v] != 0; v++ ) {
        int vorgaenger = netzplan[i][v];
        int vorgaenger_fez = netzplan[vorgaenger][FEZ];
        if( vorgaenger_fez != 0 ) {
            // vorgaenger hat FEZ
            if( vorgaenger_max < vorgaenger_fez ) {
                vorgaenger_max = vorgaenger_fez;
            }
        } else {
            // dieser vorgaenger hat keine FEZ
            vorhanden = false;
        }
    }

    // falls auch nur einer nicht vorhanden ist, geben wir 0 zurueck
    if( !vorhanden ) {
        vorgaenger_max = 0;
    }
    return vorgaenger_max;
}

// Berechne alle noch fehlenden Daten im Netzplan.
// Solange noch ein FEZ fehlt, muessen wir erneut
// alle Vorgaenge betrachten
void netzplan_fuellen(int netzplan[MAX_VORGAENGE][MAX_BREITE]) {
    while( fez_fehlt(netzplan) ) {
        // Schleife ueber alle Vorgaenge
        for( int i = 1; i < MAX_VORGAENGE; i++ ) {

            // FEZ fehlt
            if( netzplan[i][FEZ] == 0 ) {
                netzplan[i][FEZ] = fez( netzplan, i );
            }

            // FAZ fehlt
            if( netzplan[i][FAZ] == 0 ) {
                netzplan[i][FAZ] = vorgaenger_max( netzplan, i );
            }
        }
    }
}

// alle FEZ berechnet, Netzplan ist vollstaendig
int max_fez( int netzplan[MAX_VORGAENGE][MAX_BREITE] ) {

    int max_fez = netzplan[1][FEZ];
    for( int i = 2; i < MAX_VORGAENGE; i++ ) {
        if( max_fez < netzplan[i][FEZ] ) {
            max_fez = netzplan[i][FEZ];
        }
    }
    return max_fez;
}

```

Algorithmus zur Ausgabe des kritischen Pfades

Idee: Durchsuche den Netzplan vom letzten Vorgang ausgehend zum ersten Vorgang. Verfolge immer den Weg, der keinen Puffer gelassen hat.

- Wähle Vorgang v mit größtem FEZ
- Gib Nummer von v aus
- Solange v einen Vorgänger hat
 - Ermittle den Vorgänger mit der größten FEZ, weise ihn v zu
 - Gib Nummer von v aus

Hinweis: Es kann durchaus mehrere kritische Pfade geben.

Vorgangmodell der Softwareentwicklung

- Anforderungsdefinition
 - Vom Anwender erstellt. Informell, lückenhaft, grob
- Pflichtenheft
 - Von Anwender + Softwareingenieur gemeinsam erstellt, Vertragscharakter
- Entwurf
 - Vom Softwareingenieur erstellt. Anwender teilweise involviert.
- Programmierung
 - Softwareingenieur/Programmierer
- Test
 - Softwareingenieur + Anwender
- Einführung/Schulung
 - Softwareingenieur

Was haben wir *immer noch nicht* gemacht?

- Zeigerdatentypen (*, &)
- Lebensdauer/Gültigkeit von Variablen
- Objektorientierung (Klassen, Objekte, Vererbung, Polymorphie)
- Speicherverwaltung (Stack, Heap)
- Ausnahmebehandlung
- Graphische Oberflächen, Betriebssystem-API, Bibliotheken
- ...