

C++

Protokoll des Anfangsunterrichts am Gymnasium Hermeskeil Januar - Juni 2000

Die Schülerinnen und Schüler des gymHerm haben den ersten C++Kurs ihres Fachlehrers protokolliert, indem sie die links stehenden Quelltexte der Beispielprogramme des Unterrichts in der rechten Spalte mit ihren eigenen Worten kommentiert haben.

Das C++Konzept und der heutige Sprachstandard findet sich als Literaturhinweis am Ende des Protokolls. Weitere Anregungen findet man bei den Informatik-Fachbereichen der Universitäten, etwa unter

hal.iwr.uni-heidelberg.de/lehre/inf1-ws02/index.html

Überarbeitung 2003
Georg und Mario Spengler

Für diesen C++ Kurs wurde das plattformübergreifende Programmiersystem Codewarrior verwendet, welches es elegant erlaubt, alle header-Dateien „unsichtbar“ zu implementieren.

Bei anderen Systemen, die auf dem GNU gcc-Compiler basieren und damit den C++ Standard einhalten, auch beim heute am gymHerm verwendeten Project Builder, werden alle verwendeten Header-Dateien zu Beginn eines Programms aufgelistet.

Die bisher von uns verwendeten Header sind:

Header	Bemerkungen
<code>#include <iostream></code>	Ein- Ausgabe,
<code>#include <iomanip></code>	Formatierungen
<code>#include <fstream></code>	Files, Dateiverarbeitung
<code>#include <vector></code>	Methoden der Klasse vector
<code>#include <string></code>	Methoden der Klasse string
<code>#include <sstream></code>	Strings in Streams leiten
<code>#include <ctime></code>	SystemZeit
<code>#include <cstdlib></code>	Zufallsgenerator, ...
<code>#include <algorithm></code>	Methoden der Klasse algorithm etwa swap, sort, find, ...
<code>using namespace std;</code>	Alle weiteren Programmteile sind dem Namensbereich std der C++ Standardbibliothek untergeordnet.

Schlüsselwörter in C++

Kommentar

Peter M.

1. einfache Datentypen

- int i, nr;
- short kurz;
- long lang;
- unsigned short z, s;
- unsigned long zeile, spalte;
- float zahl
- double i, nr;(genauer als int)
- bool daten
- char buchstabe, zeichen
- void dummy; // unspezifizierter Typ
z.B. Rückgabewert bei Prozeduren

Siehe Protokoll Seite 11

Siehe Protokoll Seite 21, 24
Siehe Protokoll Seite 22, 24

2. Zusammengesetzte Datentypen

- int feld[8]
- string stundenplan[10][5];
- struct person { int nummer;int name;};

Siehe Protokoll Seite 22, 18
Siehe Protokoll Seite 22, 24
Siehe Protokoll Seite 18, 20

3. Klassen

- string zeile, wort;
- vector v, liste;
- Zeit aktuell; // selbst definiert
- Matrix m1, m2; // selbst definiert

Siehe Protokoll Seite 23,24,25
Siehe Protokoll Seite 22, 24
Siehe Protokoll Seite 3, 27
Siehe Protokoll Seite 26, 28

4. Schleifen

- while while (Bedingung) {Anweisung}
- do do {Anweisung} while(Bedingung)
- for for(int i=1;i<n;i++){...}

Siehe Protokoll Seite 3, 6, 12
Siehe Protokoll Seite 6, 12
Siehe Protokoll Seite 3, 6

5. Verzweigungen

- switch - case switch (Bedingung) {
case:{Anweisungen;};break; }
- if-else if (Bedingung) {Anweisung 1}
else {Anweisung 2}

Siehe Protokoll Seite 13, 15

Siehe Protokoll Seite 14, 19

6. Operatoren

7. Standardfunktionen

- 8. Sonstige main, try, catch

Siehe Protokoll Seite 16,
Siehe Protokoll Seite 17,
Siehe Protokoll Seite 20, 30

```

#include MSLHeaders++.h // in GymHerm stationary
using namespace std; // ----- Mittelwert 1

int main(int argc, char* argv[]) // Mittelwert 1
{

int      n, note, summ, qsum;
float    mittel, sigma;

n = summ = qsum = 0;

do {
    n = n + 1;
    cout << n << ". Note : "; cin >> note;
    summ = summ + note;
    qsum = qsum + note * note;

} while (note!=0);

n = n - 1; // weil eins zu viel addiert wurde

```

1. Beispiel

Christof

Namensbereich "std" verwenden
kurze Beschreibung des Inhalts, eingefügt als Kommentar hinter
Kommentarzeichen ("//")

Aufruf der Hauptprozedur main(int argc, char* argv[]), ist
gleichzeitig Programmstart; Das Argument wird gegebenenfalls
vom Betriebssystem verwendet, kann aber oft entfallen.
Die offene geschweifte Klammer markiert den Start.

Alle verwendeten Variablen werden deklariert:
zuerst den Typ (int = Ganze Zahlen, float=Fließkommazahlen),
dann eine Liste der Variablen.
Jede Anweisung und jeder Block endet mit Semikolon (";").

Variablen werden mit 0 initialisiert

Beginn einer "do"-Schleife mit "do".
Generell werden { mehrere Befehle } mit geschweifeten
Klammern zusammengefasst.
In der Variable "n" werden die Schleifendurchläufe und damit die
Anzahl der eingegebenen Noten gezählt.
Eine Aufforderung zur Noteneingabe wird in der Form "n. Note:"
über die Prozedur "cout <<" ausgegeben, die Eingabe wird über
die Prozedur "cin >>" in die Variable "note" eingelesen. "note" wird
zur Variable "summ" addiert und wieder in "summe" gespeichert;
das Quadrat der "note" wird zu der Variable "qsum" addiert und in
"qsum" gespeichert.
Ende des Schleifeninhalts wird durch "}" markiert gefolgt von der
Bedingung zur Wiederholung, hier: wenn "note" ≠ 0 .

Wegen der Note 0 wurde eins zu viel gezählt,

```

mittel = static_cast<float>(summ) / n;

sigma = sqrt(static_cast<float>(qsum)/n-mittel* mittel);

cout << endl; cout << endl;

cout << "Mittelwert          = " << setw(6) <<
setprecision(4) << fixed << mittel << endl;

cout << "Standardabweichung = " << setw(6) <<
setprecision(4) << fixed << sigma  << endl;

return EXIT_SUCCESS;    // return 0;

}

```

Der Wert der short-Variable "summ" wird mit der Funktion "static_cast" in einen float-wert konvertiert, durch "n" geteilt und der Quotient in die Variable "mittel" gespeichert.

sqrt steht für square root !

Zwei Zeilenumbrüche.

Der Text "Mittelwert = " wird ausgegeben, die Vorkommastellen werden auf 6 festgelegt, die Nachkommastellen auf 4, durch "fixed" werden die Kommas sauber übereinander geschrieben, der Wert der Variablen "mittel" wird ausgegeben und ein Zeilenumbruch erzwungen.

Wie oben.

Weist „main“ auf die Hauptebene des Betriebssystems zurück. wobei der Rückgabewert einem Systemfehlercode entspricht (0 bedeutet : kein Fehler)

Die geschlossene geschweifte Klammer markiert das Ende.

```

// Mittelwert und Standardabweichung nach allen Eingaben
using namespace std;

int main(void) // -----
{
    vector<int>      x;
    short           n,note,summ,qsum;
    float          mittel,sigma;
    unsigned long   i;

    n = summ = qsum = 0;

    do
    {
        n = n + 1;
        cout << n << ". Note :"; cin >> note ;
        x.push_back(note);
    } while ( note != 0 );

    n = n - 1;
    x. pop_back();

    for(i = 0; i < n; i++) summ = summ + x. at (i);
    mittel= static_cast<float>(summ)/n;

    for (i = 0 ; i < n ; i++)
    { qsum = qsum+(x.at(i)-mittel)*(x.at(i)-mittel);
    }
    sigma = sqrt (static_cast<float>(qsum)/n);

    cout<< "Mittelwert= " << setw (6) << setprecision
(4) << fixed << mittel << endl;
    cout<< "Standardabweichung= " << setw (6) <<
setprecision(4) << fixed << sigma << endl;

    return EXIT_SUCCESS;
}

```

2. Beispiel

Helena

vector : Anordnung von Zahlen mit integer- Werten

i : Index für eine Vektorkomponente

Nullzuordnung (alles wird gleich Null gesetzt)

Speicherung der Note in den Vektor:

1. die VektorKlasse x aufrufen mit „x.“

2. Anfügen einer Note als Vektorkomponente mit push_back
while(note!=0): Eingabeende mit der Zahl 0

push_back: Hereinholen der Note

pop_back : Herausholen der Note

Schleife wird für n-Stück durchzählt und summiert die alte Summe
zur jeweiligen neuen Vektorkomponente.

x.at(i): Bedeutung: die i-te VektorKomponente.

for - Schleife: i = 0 Erster Index
 i < n Letzter Index
 i = i+1 Nächster Index

Somit wird der Anfangs- und Endpunkt sowie die Schrittweite der
Schleife definiert. In der 2. for-Schleife ist i++ eine Abkürzung für
den Befehl "addiere 1" (i=i+1).

sqrt:Abkürzung für "sqare_root!" und bedeutet die Quadratwurzel
aus dem Klammerterm.

Programmaufbau eines C++ Programms

Ein C++ Programm besteht aus:

- einer Kommentarzeile mit dem Programmnamen
- den include-Anweisungen für die Header-Dateien
- einem Kopf (using)
- der Definition von Prototypen
- der Deklaration von Funktionen und Unterprogrammen und
- dem Hauptprogramm main.

0. Am Anfang des Programms steht normalerweise der Name des Algorithmus als einzeliger Kommentar.
(siehe unter 7.)

1.

```
#include <iostream>
#include <iomanip>
.....
```

2. Alle weiteren Programmteile sind dem Namensbereich std der C++ Standardbibliothek untergeordnet.

```
using namespace std;
```

3. Jetzt folgt das Hauptprogramm, welches mit

```
int main ()
{
```

und einer einer geöffneten geschweiften Klammer beginnt und am Ende des Programms mit einer geschlossenen geschweiften Klammer } schließt.
(siehe 4.)

3.1. Innerhalb dieses Algorithmus werden zunächst die Variablen festgelegt, ähnlich wie im Struktogramm. Variablen gleichen Typs werden durch Komma getrennt.

3.2. Hierauf folgt die Initialisierung der Variablen, dann eventuell nach den EVA-Prinzip die

3.3 Eingabe, die mit cin << beginnt. Es folgt die

3.4 Verarbeitung im eigentlichen Algorithmus; die

3.5 Ausgabe wird durch cout << eingeleitet.

4. Bevor die abschließende geschweifte Klammer gesetzt wird, schließt man den Hauptteil main mit

```
return EXIT_SUCCESS;
}
```

5. Hinter jedem Prototyp und jedem Befehl hat ein Semikolon zu stehen, vor einer sich öffnenden geschweiften Klammer dagegen nicht.

6. Äußere Form (nicht relevant für den Ablauf):
Zeilen, die logisch miteinander verbunden sind, sollten direkt aufeinanderfolgend stehen, während Zeilen, die keinen logischen Zusammenhang haben, durch eine Leerzeile getrennt werden.
Bei Unterprogrammen, Wiederholungen oder Verzweigungen wird mit der Tab-Taste eingerückt.

7. Kommentare

7.1 // Ein einzeliger Kommentar (2 Schrägstriche)

7.2 /* Ein mehrzeiliger Kommentar (Beginn: Schrägstrich Stern)
kann sich durchaus über viele Zeilen

erstrecken (Ende : Stern Schrägstrich) */

E ingabe, V erarbeitung und A usgabe eines C++Programms

Michael Z.

```
int a, b, c, n, v;  
float r, s, t;
```

Eingabe: durch Input -Streams

```
cin >> a >> b ;  
cin >> t;
```

liest die ganzzahligen Variablen a und b, bzw. eine Kommazahl t (*Dezimalpunkt tippen !*) ein und speichert die eingelesene Größe in der entsprechenden Variablen.

Verarbeitung:

In diesem Teil erfolgt nun die Abarbeitung des Algorithmus durch Anweisungen.

Jede Anweisung wird durch Semikolon abgeschlossen.

Anweisungen sind Zuweisungen, Schleifen, Verzweigungen, Unterprogrammaufrufe oder Anweisungsfolgen.

Eine spezielle Anweisung ist die Zuweisung:

```
n = n + 1; oder abgekürzt n++;
```

Generell gilt für Zuweisungen, daß der rechts stehende Ausdruck der links stehenden Variablen zugeordnet wird.

Mehrere Anweisungen werden durch geschweifte Klammern zu einer Anweisungsfolge zusammengefasst; etwa für folgende Zuweisungen:

```
{ b = b + 3*c;  
  c = 5 - b;  
  a = b + c;  
  b = a % 10;  
  r = 3.14159;  
  s = static_cast<float> (a+b);  
  v = static_cast<int> (r);  
}
```

Dabei bedeuten

% den Rest bei der Ganzzahldivision (*a mod 10*) und `static_cast<ZielTyp>(QuelleVariable)` eine Typenkonversion

Ausgabe: durch Output - Streams

```
cout << a << b << n ;
```

wobei für a,b,n nur ganzzahlige Werte gelten.

Sind Variablen vom Typ float oder double (*Kommazahlen*), so sollte die Ausgabe entsprechend formatiert werden, etwa

```
cout << setw(8) << setprecision(2) << fixed ;  
cout << "π = " << pi << endl;
```

Die Formatierung erfolgt durch Manipulatoren wie:

- setw (v)	für die Vorkommastellen
- setprecision(n)	für die Nachkommastellen
- fixed	für eine feste Stellenzahl
- scientific	für Exponentialschreibweise
- showpoint	zur Anzeige des Dezimalpunkts
- dec, okt, hex	zur Anzeige Dezimal, Oktal, Hexa
- endl	für einen Zeilenumbruch.

Die for- Schleife

Christian

Die "for"- Schleife stellt im Prinzip eine Variante der "while" - Schleife dar. Der Unterschied zwischen beiden besteht im Wesentlichen darin, dass bei der "while" -Schleife nur eine Bedingung abgefragt wird, bei der "for"- Schleife aber gleichzeitig die Arbeitsschritte mitgezählt werden.

Eine "for"- Schleife sieht folgendermaßen aus:

```
for ( Initialisierungsausdruck; Endbedingung; Inkrementierungsausdruck )  
{  
    // Anweisungen..  
}
```

Der Initialisierungsausdruck beinhaltet lediglich die Initialisierung der Zählvariablen (z.B. "i").

Die Endbedingung stellt eine Abfrage dar, die bei jedem Durchlauf prüft, ob die Schleife erneut auszuführen ist oder nicht. Die nach der geschweiften Klammer folgenden Anweisungen werden solange ausgeführt, bis die Endbedingung erfüllt ist.

Beim Inkrementierungsausdruck handelt es sich die Schrittweite zwischen den einzelnen Durchläufen.

Im Allgemeinen haben wir es hier mit Ausdrücke wie "i = i*5", "i = i+10" oder dem häufigen i++ (kurz für i = i+1), also mit mathematischen Operationen zu tun.

Theoretisch können auch beliebig viele Schleifenzähler initialisiert und auch unterschiedlich getestet und inkrementiert werden.

1. Muster : Eine Anweisung

```
for( int i=4711; i <= 4720; i++) cout << i ;
```

In diesem Falle wird i als ganze Zahl definiert und mit 4711 initialisiert. Solange i <= 4720 ist, wird i ausgegeben. Bei jedem Durchlauf der Schleife wird i um 1 erhöht.

2. Muster : Mehrere Anweisungen

```
for ( int i=1; i <= n; i++)  
{  
    a = ( a % b ) * 10;  
    cout << a / b << " " ;  
    if(i % 25 == 0) cout << endl << endl;  
}
```

3. Muster : Andere Datentypen als integer aber vom Typ enumerable(abzählbar)

```
// Ausgabe: Alphabet ABCDEFGHI.....XYZ  
for ( char c = 'A' ; c < 'Z' ; c++) cout << c;  
  
// Ausgabe: Zufallswort mit 5 Buchst, etwa TNFRS  
  
for ( int i = 1 ; i < 6 ; i++)  
{  
    cout<<static_cast<character>(zufall(96,120));  
}
```

Unterprogramme

```
using namespace std; // ----- Wechselgeld

void eingabe( string , float& );
void ausgabe( string , float , string );
void buche ( float& , short , string );

float rest;

void eingabe( string kommentar, float& wert)
{ cout << kommentar << " : "; cin >> wert;
} // eingabe

void ausgabe( string komm1, float wert, string komm2)
{ cout << setw(8) << setprecision(2) << fixed << ;
  cout << komm1 << wert << komm2 << endl;
} // ausgabe

void buche ( short fest, string komm)
{
  while ( rest >= fest)
  {
    rest = rest - fest;
    cout << setw(8) << fest << komm << endl;
  }
} // buche
```

3. Beispiel:

Isabell

Nach „using“ und dem Programmnamen folgt die

Liste der Prototypen (jeweils ohne Variablen, mit Strichpunkt)
in derjenigen Reihenfolge, wie sie im Hauptprogramm „main“
später auftreten. Jeder Prototyp nur durch seine Datentypen
genau beschrieben werden, bei eingabe etwa durch (string,
float&); Durch die Referenzierung „&“ wird der Variablenparameter
an das Hauptprogramm zurückgegeben.

Der Kopf eines jeden Unterprogramms beginnt mit
„void“, dann folgt der
„Name“ des Unterprogramms, dann die „Parameterliste“.
Der { Rumpf } wird durch geschweifte Klammern markiert.
Nach der letzten } sollte das Ende kommentiert werden.
Neben den Typen erscheinen Variablen (Kommentar, Wert)
hinter der ersten Zeile kommt kein Semikolon.

Setzen aller Formatierungen: 8 Gesamtstellen, 2
Nachkommastellen, Feste Stellenzahl, 1. Kommentar, Wert, 2.
Kommentar, neue Zeile mit „endl“.

Buche: was reinkommt, muss ausgegeben werden
while fragt ab, wie groß der Restbetrag ist (z.B. Rest > 500 nein
oder ja)
Der Festbetrag subtrahiert vom Restgeld, ergibt den Rest

```

int main(void) // ----- Wechselgeld
{
    float kosten, zahlung, nach;

    eingabe ( "gekostet", kosten );
    eingabe ( "gezahlt ", zahlung );
    cout << "" << endl << endl;

    while (zahlung < kosten)
    {
        eingabe ( "Bitte nachzahlen", nach );
        zahlung = zahlung + nach;
        cout << "Gesamtzahlung : " << zahlung << endl;
    }

    rest = zahlung - kosten;
    cout << endl << endl;
    ausgabe("Restgeld = ",rest, " DM");

    cout << endl << endl << "buche:" << endl << endl;

    buche ( 1000, " DM");
    buche ( 500, " DM");
    buche ( 200, " DM");
    buche ( 100, " DM");
    buche ( 50, " DM");
    buche ( 20, " DM");
    buche ( 10, " DM");
    buche ( 5, " DM");
    buche ( 2, " DM");
    buche ( 1, " DM");

    rest = rest*100+0.1; cout << endl;

    buche ( 50, " Pf");
    buche ( 10, " Pf");
    buche ( 5, " Pf");
    buche ( 2, " Pf");
    buche ( 1, " Pf");

    return EXIT_SUCCESS;
}

```

Eingabe der Daten
Variable: Einzahlung mit Wert belegen

nach der while-Schleife folgt eine Bedingung
es wird solange eingezahlt, bis der Einzahlungsbetrag größer oder gleich den Kosten ist, d.h. nach der Einzahlung folgt immer eine neue Berechnung

Den Rest berechnet man, indem man die Kosten von der Einzahlung subtrahiert.

Eingabeteil wird von Ausgabeteil getrennt
Ausgabe: 2 strings (Typennamen), 1 Kommazahl
Kommentar, Wert und Einheit

Es wird mit "DM" kommentiert, später mit "Pf"
Ausgabe in Scheinen und Markstücken

Der Rest wird genommen als $\text{alter Rest} * 100 > \text{Pfennig}$

Ausgabe in Pfennigen
von buche kommt eine feste Zahl

```

using namespace std; // ----- Divison -----

int main (void)
{   int a,b,n,i,rest;

    cout << "a = "; cin >> a;
    cout << "b = "; cin >> b;
    cout << endl << endl;
    cout << "Stellenanzahl = "; cin >> n;
    cout << endl << endl;
    cout << a << "/" << b << " = " << a/b << ", " ;
    cout << endl << endl;

    for ( i=1 ; i <= n ; i++)
    {   rest = a % b;

        a = rest * 10;
        cout << a / b << " ";
        if ( i % 25 == 0 ) cout << endl << endl;
    }//for
    cout << endl << endl;
    return EXIT_SUCCESS;
}

```

```

using namespace std;// ----- ggT kgV -----

int ggT(int, int);
int kgV(int, int);
void eingabe(string, int&);
void ausgabe(string, int);

```

4. Beispiel: Ganzahldivision, ggT, kgV

Dennis

Form der Eingabe

Genauigkeit

Ausgabeformatierung

Bedingung der "for-Schleife"

"%" steht für modulo, es wird also die Division ausgeführt und mit dem einfachen "=" dem "Rest" zugeordnet. Wie bei der normalen Division wird der rest mit 10 multipliziert, um die Nachkommastellen zu berechnen.

Das doppelte "==" heißt ist gleich.

Die Schreibweise des Ergebnisses und die Festlegung, nach 25 Ziffern eine neue Zeile zu beginnen.

Nach "using....."und dem Programmnamen folgt

die Liste der Prototypen (jeweils ohne Variablen, mit Strichpunkt) in derjenigen Reihenfolge, wie sie in im Hauptprogramm "main" später auftreten. Jeder Prototyp muss unten genau beschrieben werden. Das "&" heisst, dass der Wert berechnet und dann weiterverwendet wird.

```

int ggT(int a, int b)
{
    int rest, li, re;
    li = a;
    re = b;

do {
    rest = li % re;
        li = re;
        re = rest;
    } while (rest != 0);
    return li;
} // ggT

int kgV(int a, int b)
{
    return a*b/ggT(a,b);
}

void eingabe(string text, int& zahl)
{
    cout << text;
    cin >> zahl;

}

void ausgabe(string text, int zahl)
{
    cout << text << Zahl << endl;
}

int main (void)
{
    int z1, z2;
    eingabe("a = ", z1);
    eingabe("b = ", z2);
    cout << endl;
    ausgabe("ggT = ", ggT(z1,z2));
    ausgabe("kgV = ", kgV(z1,z2));

    return EXIT_SUCCESS;
}

```

Der Kopf einer Funktion beginnt mit dem Typ, dann folgt der Name des Unterprogramms, dann die Parameter.
Der Rumpf wird durch geschweifte Klammern markiert.
Nach der letzten Klammer sollte das Programm kommentiert werden.

"do-Schleife": Die Division "a/b" bzw. (da hier keine Variablen verwendet werden "li/re") werden durchgeführt, bis der rest gleich Null ist.
Das letzte "li" ist der ggT und wird ins Hauptprogramm zurückgegeben.

Die Funktion kgV wird in diesem Unterprogramm berechnet.
Hierbei wird die oben definierte Funktion ggT verwendet .

Während die Funktionen ggT und kgV mit dem Argumenttyp „int“ beginnen, werden die Eingaben in einem echten Unterprogramm (void) erfasst. Man beachte, dass die Variable „zahl“ mit „&“ referenziert sein muss, da ihr Wert ins Hauptprogramm zurückgegeben werden muss. (wie return bei Funktionen) Das Gleiche gilt für die Ausgaben. Hier treten keine referenzierten Variablen auf, da keine Rückgabe nach „main“ erfolgt.

Das Hauptprogramm beginnt mit der Aufforderung die Zahlen einzugeben, einer Leerzeile und dem Ergebnis.

Man beachte, dass das echte Unterprogramm ausgabe die Funktionen ggT und kgV direkt aufrufen kann.

Datentypen

- unsigned short int
- short int
- unsigned long
- long int
- char
- float
- double

Wertebereiche

unsigned short int :	0-65,535	(16 Bit)
short int:	-32,768- 32,767	(16 Bit)
unsigned long int:	0-4,294,967,925	(32 Bit)
long int:	-2,147,483,648- 2,147,483,648	(32 Bit)
char:	256 Zeichenwerte	(8 Bit)
float:	1,2e-38- 3,4e 38	(32 Bit)
double:	2,2e- 308 bis 1,8e 3088	(64 Bit)

unsigned short int umfasst den Bereich der ganzen positiven Zahlen bis 65,535, die erste Stelle wird nicht als Vorzeichen genutzt.

short int sowohl negative als auch positive Zahlen (siehe Wertebereich)

unsigned long ebenfalls Bereich der positiven Zahlen, nur mit größerem Speicherplatz als bei unsigned short int.

Jenny

long int

positive und negative Zahlen, umfasst ebenfalls einen größeren Bereich als short int. Die erste Stelle wird als Vorzeichen genutzt, (null=positiv; eins=negativ).

char

Jeder Zahl ist ein Schriftzeichen zugeordnet (0-255) z.B. 65=A

float

Gleitkommazahlen, umfasst auch den Bereich der Dezimalzahlen, 32 Bit: die erste Stelle ist das Vorzeichen, die nächsten acht sind Exponenten und die nächsten 23 sind Mantisse

double

selbe Funktion wie float, nur das doppelte an Speicherplatz.

Typkonvertierungen:

- bez1 = (Typ_von_bez1) Ausdruck;
- bez1 = (Typ_von_bez1) bez2;
- bez1 = static_cast < Zieltyp > (bez2);

Bedeutung :

bez1 Objekt, dem das umgewandelte Objekt bez2 zugewiesen wird
 ZielTyp Angabe des Typs, in den bez2 umgew. werden soll
 bez2 Umzuwandelndes Objekt

Beispiele

- int bez1 = (int) 3.14*sin(1.3);
- float bez1 = (float) bez2;
- int bez1 = static_cast<int> (3.14159);

Die WHILE - Schleife

Michael Sch.

Schleife mit Anfangsabfrage

Die While-Schleife ist eine Schleife die den Befehl immer nach überprüfen einer Bedingung ausführt.

```
while ( Bedingung ) {Anweisungen ;}
```

In C++ wird diese Schleife mit dem Befehl " while " - daher der Name - was übersetzt "solange" bedeutet, eingeleitet. Hinter diesem "while" folgt in runden Klammern die Bedingung, die erfüllt sein soll, damit der Befehl ausgeführt wird. Dann folgt die Befehlskette, die ausgeführt werden soll, in geschweiften Klammern. Wenn es sich nur um einen Befehl handelt können die geschweiften Klammern weggelassen werden.

Beispiel (einzeilig):

```
while ( n > 3 ) cout << " richtig " ;
```

Beispiel (mehrzeilig):

```
while ( x < 0 ) {  
    s == a * b ;  
    i== a+b;          cout << s ;  
    cout << endl ;   cout << i ;          cout <<  
endl ;  
}
```

Die DO - Schleife

Schleife mit Endabfrage

Es besteht auch die Möglichkeit die While - Schleife umzudrehen, also zuerst den Befehl auszuführen und dann die Bedingung zu prüfen.

```
do { Anweisungen ; } while ( Bedingung )
```

Dies geschieht, indem man vor die Befehlskette ein „do“ setzt und das " while“ einfach hinter die Befehlskette setzt. Deshalb nennt man diese Schleife auch die Do -Schleife.

Beispiel (einzeilig):

```
do n = n+5; while ( n < 60);
```

Beispiel (mehrzeilig):

```
•  
••  
•••  
••••  
•••••  
••••••
```

Das folgende Programmfragment erzeugt nebenstehende Ausgabe:

```
z = 0;  
do {  
    z = z+1;  
    for (s=0; s<z; s++) cout << "•";  
    cout << endl;  
} while (z==6);
```

```
using namespace std;// ----- Wochentage -----
```

```
int zufall(int, int); // Prototyp
```

```
int zufall(int a, int b) // Deklaration  
{  
    return (a+rand()%(b-a+1));  
}
```

```
int main (void)
```

```
{  
    int zahl;
```

```
    srand ( time(NULL) );
```

```
    cout << "Bitte geben Sie eine Zahl ein: ";  
    cin >> zahl;  
    cout << endl << endl;
```

```
    switch(zahl)  
    {  
        case 1: cout << "Montag";    break;  
        case 2: cout << "Dienstag";  break;  
        case 3: cout << "Mittwoch";  break;  
        case 4: cout << "Donnerstag"; break;  
        case 5: cout << "Freitag";   break;  
        case 6: cout << "Samstag";   break;  
        case 7: cout << "Sonntag";   break;  
        default:cout << "Blödmann";  break;  
    } // switch
```

```
    cout << endl;
```

```
    return EXIT_SUCCESS;
```

```
}
```

Zufallszahlen

Sabine

rand liefert eine Zufallszahl zwischen 0 und $2^{31}-1$, allerdings sind diese Zufallszahlen etwa für Noten unbrauchbar, sie sollten normiert werden durch eine Funktion Zufall, welche man etwa für Noten mit `zufall (1,6)` für MSS-Punkte mit `zufall (0,15)` und für Schuhgrößen mit `zufall (35,45)` aufrufen kann

`srand(time(NULL))` startet den Zufallsgenerator mit akt. Zeit

`a % b`
bedeute den Rest bei Division von a durch b (a modulo b)

Fallunterscheidungen

Eine Fallunterscheidung ganz allgemein:

```
switch ( Ausdruck) {  
    case wert1 : { Anweisungen; }; break;  
    case wert2 : { Anweisungen; }; break;  
    case wert3 : { Anweisungen; }; break;  
    default   : { Anweisungen; };  
}
```

Auflistung der Fälle. " break" , damit direkt ans Ende gesprungen wird und nicht mehrfach wiederholt wird. Bei einem Fall. der nicht in der case-Liste zu finden ist, wird "default" aufgerufen.

Semikolon „ ; „ nach jeder Anweisung und nach jeder Vereinbarung , jedoch nicht vor „ { “ und nicht nach „ } “.

Die bedingte Anweisung

if (...) { ... } else { ... }

Jochen

Beispiel:

Aufbau:

Eine Verzweigung mit der if - Anweisung besteht aus

- dem Schlüsselwort if,
- einer Bedingung,
- einer Anweisung und gegebenenfalls aus
- dem Schlüsselwort else und
- einer Ausweichanweisung.

Syntax:

2.1 if - Verzweigungen bestehen nur aus einer Bedingung und einer (mehreren) Anweisung(en) :

```
if ( Bedingung ) { Anweisungen ; }
```

Wenn die Bedingung erfüllt ist, wird die Anweisung ausgeführt, ansonsten wird sie übersprungen.

2.2 if - else - Verzweigungen bestehen zusätzlich aus einer Anweisung, die nur ausgeführt wird, wenn die erste Anweisung nicht erfüllt ist.

```
if ( Bedingung ) { Anweisungen ; } else { Anweisungen ; }
```

Diese Verzweigung wird verwendet, wenn eine Fallunterscheidung zwischen zwei Fällen vorliegt.

```
// Quadratische Gleichung
```

```
using namespace std;
```

```
int main ( void)
```

```
{ float a, b, c, x, D;
```

```
cout << "Quadratische Gleichung" << endl;
```

```
cout << "ax^2 + bx + c = 0" << endl << endl;
```

```
cout << "welches a: " ; cin >> a;
```

```
cout << "welches b: " ; cin >> b;
```

```
cout << "welches c: " ; cin >> c;
```

```
cout << endl << "Lösungen von " ;
```

```
cout << a << "x^2 + " << b << "x + " << c ;
```

```
cout << " = 0" << endl << endl;
```

```
if ( a == 0 ) {
```

```
if ( b == 0 ) {
```

```
if ( c == 0) cout << " IL = IR ";
```

```
else cout << " IL = {} ";
```

```
}
```

```
else cout << " IL = { " << -c / b << " } ";
```

```
}
```

```
else {
```

```
x = -b / 2 / a ;
```

```
D = (b * b - 4 * a * c ) / 4 / a / a ;
```

```
if ( D == 0 ) cout << "IL = { " << x << " }";
```

```
if ( D <= 0 ) cout << " IL = {} " ;
```

```
if ( D >= 0 ) {
```

```
cout << "IL = { " << x + sqrt (D);
```

```
cout << "; " << x - sqrt (D) << " }";
```

```
}
```

```
return EXIT_SUCCESS;
```

```
} // Quadratische Gleichung
```

```
// Übungsaufgabe Notenstatistik
```

```
using namespace std;
```

```
int main (void)
{
    char zeichen;
    int    note,i,anz1, anz2, anz3, anz4, anz5, anz6;

    zeichen = '•';
    anz1 = anz2 = anz3 = anz4 = anz5 = anz6 = 0;

    do {
        cout << "Note : " ;cin >> note;
        cout << endl;
        switch (note) {
            case 1: anz1 = anz1 + 1;break;
            case 2: anz2 = anz2 + 1;break;
            case 3: anz3 = anz3 + 1;break;
            case 4: anz4 = anz4 + 1;break;
            case 5: anz5 = anz5 + 1;break;
            case 6: anz6 = anz6 + 1;break;
            default: cout << "ungültig" << endl;break;
        } // Zählen und speichern der Noten

    } while (note != 0); // Eingabe der Noten

    cout << endl << "    1: ";
    for ( i=1;i<=anz1; i++) cout << zeichen;
    cout << endl << "    2: ";
    for ( i=1;i<=anz2; i++) cout << zeichen;
    cout << endl << "    3: ";
    for ( i=1;i<=anz3; i++) cout << zeichen;
    cout << endl << "    4: ";
    for ( i=1;i<=anz4; i++) cout << zeichen;
    cout << endl << "    5: ";
    for ( i=1;i<=anz5; i++) cout << zeichen;
    cout << endl << "    6: ";
    for ( i=1;i<=anz6; i++) cout << zeichen;
    cout << endl << endl; // Ausgabe der Noten

    return EXIT_SUCCESS;
}
```

Beispiel : Notenstatistik

Volkmar W.

Nach der Initialisierung der Variablen beginnt das eigentliche Hauptprogramm.

Vereinbarung von Variablen vom Typ Zeichen

```
char zeichen;
```

die Zuweisung erfolgt mit einfachem Hochkomma

```
zeichen = '•';
```

Es besteht aus einer "do"- Schleife mit Endabfrage und einer "switch"- Fallunterscheidung.

Die Schleife mit Endabfrage fordert solange eine Eingabe einer Note zwischen 1 und 6, bis man 0 eingibt. Bei der Eingabe einer anderen Zahl wird diese nicht gewertet. Die Noten werden mit der "switch" - Anweisung gezählt und in den Variablen anz1, ... , anz6 gespeichert.

Anschließend wird jede Note durch ein Zeichen, z. B. einen Knödel "•", in einer Tabelle ausgegeben.

Beispiel für eine Endausgabe des Programms

```
1: ●●●●
2: ●●
3: ●●●●●●
4: ●●●●
5: ●●
6: ●
```

Operatoren

Natalie P.

In C++ gibt es mehr als 40 Operatoren. Die wichtigsten sind :

Arithmetische Operatoren

+ addiert zwei Operanden
- subtrahiert zwei Operanden
* multipliziert zwei Operanden
/ dividiert zwei Operanden
Ganzzahldivision, falls beide Operanden vom Typ int
Gleitkommadivision, falls ein Operand eine Fließkommazahl ist
% ermittelt den Rest einer Ganzzahldivision

Zuweisungsoperatoren (Abkürzungen)

Folgt den Operatoren ein „=“, so wird der Wert gleichzeitig einer Variablen zugeordnet, z.B.

a+=5 steht für a = a + 5;
b-=3 steht für b = b - 3;
c*=6 steht für c = c * 6 ;
d/=2 steht für d = d / 2;
a++ steht für a = a + 1;
b-- steht für b = b - 1;

Logische Operatoren:

not ! Verneinung
and && logisches und
or || logisches oder

Auch bei logischen Operatoren findet bei Verwendung eines angehängten „=-“ Zeichens eine Zuweisung statt.

Vergleichsoperatoren

== testet zwei Operanden auf Gleichheit
!= ungleich
not_eq ungleich
< kleiner als
> größer als
<= kleiner gleich
>= größer gleich

Bitweise Operatoren:

<< Linksverschiebung
>> Rechtsverschiebung
bitand & bitweises und
bitor | bitweises oder
^ bitweises XOR (exklusives oder)
~ bitweises Komplement

Beispiel dezimal	dual		Ergebnis dual	Ergebnis dezimal
5 << 3	101	<<3	101000	40
29 >> 2	11101	>>2	111	7
5&14	0101 &1110		0100	4
5 14	0101 &1110		1111	15
5^14	0101 &1110		1011	11
~11	~1101		0010	2

StandardFunktionen

```
const float PI = 3.14159; ----> Wert
float PI = 4*atan(1);

cout << sin(PI/2);          ----> 1
cout << cos(PI);           ----> -1
cout << tan(PI/4);         ----> 1

cout << asin(1);          ----> PI/2
cout << acos(1);          ----> 0
cout << atan(1);          ----> PI/4

cout << abs(-3);          ----> 3
cout << ceil(3.689);      ----> 4
cout << floor(3.126);     ----> 4

int vor; double nach;

nach = modf(PI,vor);      ---->
cout << vor << "•" << nach; 3 • 0.14159

cout << fmod(PI,0.7);     ----> 0.487989505

cout << sqrt(4);          ----> 2
cout << pow(2,3);         ----> 8

cout << log10(4711);      3.673113
cout << log (4711);      8.457656
cout << exp(1);          2.718282
```

Kommentar

Sebastian M.

Mathematische Funktionen

Sinus (Argumente im Bogenmaß)

Kosinus (Argumente im Bogenmaß)

Tangens (Argumente im Bogenmaß)

ArcusSinus --> Wert im Bogenmaß

ArcusCosinus --> Wert im Bogenmaß

ArcusTangens --> Wert im Bogenmaß

Absolutbetrag von int-Werten
(fabs());labs() analog)

Runden auf nächste Ganzzahl

AbRunden auf nächste Ganzzahl

Aufteilung einer Gleitkommazahl in
int VorkommaAnteil (2. Parameter) und
double NachkommaAnteil (Rückgabewert)

gibt den Rest der Division als double
zurück

Quadratwurzel
Potenzieren des 1. mit dem 2. Argument

Zehnerlogarithmus (etwa zur Ermittlung der Stellenzahl)
Logarithmus Naturalis
e- Funktion

Funktionen

Prototyp

```
FunktionsTyp FunktionsName (ParameterTypen);
```

Funktionsdefinition

```
FunktionsTyp FunktionsName (ParameterListe)
{
    Anweisungen;
    ....
    ....
    return Funktionswert;
} // FunktionsName
```

Beispiel:

```
char verschiebe (char, int );

char verschiebe(char c,int v)
{ if (isdigit(c)) return (char)( 48+(c-48+v)%10);
  else if (isupper(c)) return (char)(65+(c-65+v)%26);
  else if(islower(c)) return (char)(97+(c-97+v)%26);
  else return c;
} // verschiebe
```

Kommentar

Martin K.

Prototypen werden deshalb definiert, damit später die Reihenfolge bei der Unterprogrammvereinbarung beliebig sein kann. Auch FunktionsNamen dürfen nur aus Buchstaben, Ziffern und dem Zeichen „_“ bestehen.
Der FunktionsTyp entspricht dem Wertebereich der Funktion.
Die ParameterTypen entsprechen den Definitionsbereichen.

Die Parameterliste ist eine durch Komma getrennte Auflistung der Parametertypen, gefolgt von einem Variablennamen.

return gibt den Funktionswert vom FunktionsTyp an das Hauptprogramm oder andere aufrufende Programmteile zurück.

Der als Kommentar wiederholte FunktionsName kennzeichnet das Ende der Funktionsvereinbarung.

Die Funktion verschiebe verschiebt ein char c um einen int- Wert v. Sie ordnet dabei zu:

Eingabe	Rückgabe
char, int	-----> char

Ziffern werden modulo 10,
Klein- und Großbuchstaben modulo 26 ,
alle anderen Zeichen gar nicht verschoben (return c)

Verbunde

```
using namespace std ; // ----- Schülerdaten -----

struct person
{
    string    name;
    string    plan[14][5];
    float     taschengeld;
};

int main (void)

{
    const N=2;
    unsigned long i,k;
    int zeile, spalte;
    person schueler[N];

    srand(time(NULL));

    for ( i=0; i<N; i++ ) {
        cout << i+1 << ". Name : ";
        cin >> schueler[i].name;
        schueler[i].taschengeld = 50+ rand()%150;

        // Stundenplan erzeugen
        for (zeile= 0; zeile <14; zeile++)
            for (spalte=0; spalte<5; spalte++)
                schueler[i].plan[zeile][spalte]=" ---- ";
        // Stundenplan ändern
        schueler[i].plan[0][0]= " Mon  ";
        schueler[i].plan[0][1]= " Die  ";
        schueler[i].plan[0][2]= " Mit  ";
        schueler[i].plan[0][3]= " Don  ";
        schueler[i].plan[0][4]= " Fr   ";
        schueler[i].plan[4][1] = " if 1 ";
        schueler[i].plan[2][2] = " if 1 ";
        schueler[i].plan[9][4] = " if 1 ";
        schueler[i].plan[10][4]= " if 1 ";
    }
}
```

Kommentar

Oliver S.

Ein Verbund ist eine Zusammenfassung von Daten verschiedenen Typs unter einem Ganzen.

Beispiel einer Typenvereinbarung: string name, string plan, float taschengeld werden im Verbund struct person zusammengefasst.

Es ist keine Variablen- sondern eine Typenvereinbarung. Plan [14] [5] bedeutet, dass der zu erstellende Stundenplan 14 Zeilen und 5 Spalten hat.

Hauptprogramms mit Variablenvereinbarung und

Initialisieren des Zufallsgenerators mit der aktuellen Systemzeit.

Eingabe des Schülernamens mit der Hausnummer [i] sowie Erzeugung seines Taschengeldes per Zufallsgenerator.

Zeilen- und Spaltenweise Erzeugung des Stundenplanes und Belegung aller Felder mit „---“.

Überschreiben der Wochentage und Stunden.

Festlegung der Informatikstunden durch Überschreiben von 4 Stunden.

```

    for ( spalte=0; spalte<5; spalte++) {
        schueler[i].plan[1][spalte] = "-----";
        schueler[i].plan[8][spalte] = "-----";
        schueler[i].plan[13][spalte]= "-----";
    } // for spalte=0
} // for i=0
cout << endl << endl;

for ( i=0; i<N; i++ ) {

    // Schülerdaten ausgeben
    cout << "      " << schueler[i].name;
    for ( k=schueler[i].name.length(); k<20; k++)
        cout << " ";
    cout << "●●● " << setw(5);
    cout << schueler[i].taschengeld;
    cout << " DM ●●●" << endl << endl;

    // Stundenplan ausgeben
    for (zeile= 0; zeile <14; zeile++) {

        if ( (zeile>1) and (zeile<8) )
            cout << setw(2) << zeile-1 << ". " ;
        if ( (zeile>8) and (zeile<13) )
            cout << setw(2) << zeile-2 << ". " ;
        if ( (zeile<2) or (zeile==8) or
(zeile==13))
            cout << "      " ;

        for (spalte=0; spalte<5; spalte++)
            cout << "|" <<
                schueler[i].plan[zeile][spalte];
            cout << "|" << endl;
    } // for (zeile= 0;
    cout << endl << endl;

} // for ( i=0;

return EXIT_SUCCESS;

} // main Schülerdaten

```

Setzen der Striche für die Mittagspause.

Damit alle Folgeausgaben bündig stehen, werden nach dem Schülernamen 20 - Länge des Schülernamens Leerzeichen (blank) ausgegeben; danach nun rechtsbündig die Ausgabe des Taschengeldes mit Knödeln „●“ und „DM“.

Zeilenweises Ausgeben des Stundenplanes mit

Zeilennummerierung, jedoch nicht bei

- Zeile 0,1 : Überschrift
- Zeile 8 : Mittagspause
- Zeile 13 : Feierabend

Spaltenweises Ausgeben mit

- Festsetzen der Längsstriche
- Inhalte
- Festsetzen der Längsstriche

des Stundenplanlayouts.

Prozeduren

```
// Echte Unterprogramme und Manipulationen auf Verbunden
using namespace std;

const N=15;

struct pupil
{   int      nummer;
    string   name;
    bool     sex;
    string   kurs[3];
    int      note[3];
};   // pupil

void eingabe ( int, pupil& );
void aendern ( pupil& );
void ausgabe ( pupil );

void eingabe (int nr, pupil& pp)
{   string geschl;
    pp.nummer=nr+1;
    cout << endl<<nr+1<< ". Name : "; cin >> pp.name;
    cout << "      Geschlecht : "; cin >> geschl;
    pp.sex=(geschl[1]=='m');
    for (int i=0; i<3; i++) {
        cout << "      " << i+1 << ". Fach : ";
        cin >> pp.kurs[i];
    }
} // eingabe;

void aendern (pupil& pp)
{   for (int i=0; i<N; i++)
    if (pp.note[i]<15) pp.note[i]++;
} // aendern;
```

Kommentar

Katrin H.

Eine wesentliche Aufgabe bei Unterprogrammen `void` ist der Austausch von Parametern zwischen Hauptprogramm und (Unterprogrammen. Parameter werden, wenn sie an Funktionen oder Prozeduren übergeben werden, kopiert: Sie arbeiten dann mit dieser Kopie, die man nicht dauerhaft verändern kann.

Bei Referenz - Parametern(&) können innerhalb von Funktionen und Prozeduren Werte verändert werden; die Parameter behalten ihre Werte auch nach dem Verlassen der Prozeduren und Funktionen.

Beispielsweise hat die Prozedur `eingabe` in nebenstehendem Beispiel zwei Parameter:

`int nr`
ist ein WerteParameter (value) und arbeitet mit einer Kopie des Wertes vom Hauptprogramm.

`pupil& pp`
ist ein ReferenzParameter und arbeitet mit dem Original der Variablen vom Hauptprogramm.
Referenzparameter sind notwendig, wenn Daten vom Unterprogramm ins Hauptprogramm transferiert werden sollen, also auch bei `void aendern (pupil& pp)`.

In nebenstehendem Beispiel ist die Variable `pp` ein Verbund, welcher in der Typenvereinbarung `struct` definiert wurde. Wie man leicht sieht, hat `pp` die Komponenten `nummer`, `name`, `sex` und die 3-dimensionalen Felder `kurs[3]` und `note[3]`.

```

void ausgabe (pupil pp) //           siehe unten
{
    cout << pp.nummer << ". Schüler : " ;
    if (pp.sex) cout << "Herr "; else cout << "Frau ";
    cout << pp.name;
    for(int i=0;i<20-pp.name.length();i++) cout <<" ";
    for(int i=0; i<3; i++) cout << pp.kurs[i] << " , ";
    cout << endl;
} // ausgabe;

```

```

int main (void)
{
    pupil      schueler[N];

    cout << "Eingabe" << endl << endl;
    for ( int i=0; i<N; i++) eingabe(i, schueler[i]);

    cout << "Ändern" << endl << endl;
    for ( int i=0; i<N; i++) aendern( schueler[i] );

    cout << "Ausgabe" << endl << endl;
    for ( int i=0; i<N; i++) ausgabe( schueler[i] );

    return EXIT_SUCCESS;

} // Manipulationen auf structures

```

```

void ausgabe (const pupil& pp) //           siehe oben

```

Das Unterprogramm `void ausgabe (pupil pp)` hat nur einen Werteparameter, weil nur Daten vom Hauptprogramm ins Unterprogramm transferiert werden und nicht zurück. Es sorgt in diesem Fall für eine geeignete, formatierte Ausgabe aller Daten des Verbundes `struct pupil`. Die folgende Programmzeile `for (int i=0;i<20-pp.name.length(); i++) cout << " ";` sorgt dafür, dass nach jedem Namen soviel Leerzeichen aufgefüllt werden, dass die folgende Ausgabe bündig erscheint.

Hier werden `N=15` Datensätze vom Typ `pupil` als Variable `schueler` vereinbart.

Die sehr schülerfreundliche Prozedur `aendern` erhöht die MSS-Punktezahl `note` aus dem Verbund `pupil` jedes Schülers genau dann um 1, wenn sie kleiner als 15 ist.

Man beachte, dass hier der Parameter `schueler[i]` vom Hauptprogramm als `pp` an das Unterprogramm übergeben wird, dort als `pupil& pp` verändert und anschliessend (daher Referenz &) wieder als `schueler[i]` ans Hauptprogramm zurückgegeben wird.

Die Verwendung einer als `const` deklarierten Referenz verhindert das Duplizieren der übergebenen Werte. Dies kann bei umfangreichen Verbunden, die oft auch Felder enthalten, Speicherplatz einsparen.

Die Arbeit mit Zeichenketten (string)

```
// -----StringTests-----  
  
Using namespace std;  
  
string      num2str (int);  
int         str2num (string);  
  
string num2str (int num)  
{ ostream ostr;  
  ostr << num;  
  return ostr.str();  
} //num 2 str  
  
int str2num (string sss)  
{ int num;  
  istringstream istr;  
  istr.str(sss);  
  istr >> num;  return num;  
} //str 2 num  
  
int main (void) // ----- main -----  
  
{ int      laen, posi;  
  float    real;  
  long     numm;  
  char     c;  
  string   xstr,ystr,zstr;  
  
xstr ="Berta ist eine gute"; //setzen  
cout <<"xstr:" << xstr << endl;  
  
xstr ="Tante" + xstr + "Frau";//verketteten  
cout <<"xstr:" << xstr << endl;  
  
posi = xstr.find ("gute");  
cout << "xstr.find ("gute"):" << posi << endl;
```

Kommentar

Susanne E.

siehe auch Seite 31

Definition zweier Hilfsfunktionen als Prototyp:
WortSpiel : Number To String

num2str wandelt eine int- Zahl in einen string um

Input : int num
Output : string

str2num wandelt einen string in eine int- Zahl um

Input : string
Output: int num

string ist eine vordefinierte Klasse mit vielen Funktionen,
die im Folgenden teilweise behandelt werden

Dem string wird ein Satz zugeordnet

Verketteten von strings: Konkatination

Es wird das "gute" und dessen Anfangsposition im string gesucht;
Die Variable posi steht für Position

```

xstr = xstr.erase (posi,5);
cout <<"xstr.(0...25):" << xstr << endl;

laen = xstr.length ();
cout <<"xstr.length() :" << laen << endl;

ystr = xstr.substr (6,laen-5);
cout <<"ystr:" << ystr << endl;

posi = ystr.find ("eine");
cout <<"ystr.find ('eine'):" << posi << endl;

ystr = ystr.insert (posi,"K");
cout <<"ystr:" << ystr << endl;

ystr = ystr.replace(ystr.find("keine Frau"),10,"ein
Mann");
cout <<"ystr:" << ystr << endl;

c = ystr [3];
cout <<"ystr [3]:" << c<< endl;

cout <<"ord(ystr[3]):" << (int) c << endl;

cout <<"chr (65):" << (char) 65 << endl;

cout <<"pred ystr[3] succ :";

cout << (char) (c-1) << c<< (char) (c+1)<<endl;

Return EXIT_SUCCESS

} // main

```

Der vorher gefundene Teilstring wird gelöscht;
xstr ist eine Variable aus der Klasse string
erase ist eine von vielen Methoden dieser Klasse.

Länge eines strings; length ist eine weitere Methode.

Hier wird ein Teilstring in den Zwischenspeicher ystr kopiert;
laen-5 ist die Anzahl der Buchstaben, die herauskopiert werden.

Im Beispiel hat posi die Hausnummer 10

Die Methode insert fügt an der Stelle posi ein "k" ein.

replace sucht nach string "keine Frau", beginnt, wo
"keine Frau" anfängt, löscht 10 Zeichen und fügt "ein Mann"
ein.

replace hat 3 Parameter:
- wo die Ersetzung anfängt
- wieviele Zeichen ersetzt werden sollen
- den ErsatzText

Der char- Variablen c wird das 4. Zeichen des string
zugeordnet. Die Zählung beginnt stets bei 0

Die Ordnungszahl des Zeichens wird gesucht:
(int) erzwingt eine Typenkonversion von char nach int.

(char) erzwingt eine Typenkonversion von int nach char.

Vorgänger und Nachfolger werden ausgegeben

Klasse : string

```
string sss;

cin >> sss;
cout << sss;
sss = "Dies ist eine Zeichenkette";

sss.length();
sss.find("eine");
sss.c_str();           // sss --> ccc , nur für streams

sss = ccc;             // ccc --> sss
sss.assign(ccc);
```

Umwandlung : string ind char-Feld

```
string   wort = "Mist";
char     ccc[16];
int      min;

min = ( wort.length()<16 ? wort.length() : 16 );

for ( int i=0; i<min; i++ ) ccc[i] = wort[i];
```

Feld : char

```
char ccc[256];
char ccc[] = "Heute ist Montag";

cin >> ccc; // Überlaufgefahr, besser getline
cout << ccc;
// Zuweisungen sind nicht möglich !

strlen(ccc); // die alten C-Funktionen :strcpy, strcat, strcmp

for (int i=0; i<sss.length(); i++) ccc[i] = sss[i];

cin.getline(ccc,255,'\n');
```

Die folgenden Steuerzeichen gelten als ein Zeichen:

\n	Neue Zeile statt endl
\t	Tabulator
\“	Quotes
\\	
\0	nul

Umwandlung : char-Feld in string

```
char     feld[16];
string   sss;

sss = feld; // oder ss.assign(feld);

sss.c_str();// Zugriff auf den c-String eines Strings
```

Codierung

Steffi M.

Unter Codierung versteht man die systematische Veränderung einer Zeichenfolge. Sie dient der Unkenntlichmachung eines Textes und kann zum Beispiel durch Verschieben von Zeichen nach einem bestimmten System erreicht werden.

In einer Funktion oder einem Unterprogramm kann man ohne weiteres einzelne Zeichen durch andere ersetzen.

```
char verschiebe ( char buchstabe, int zahl)
{
    return (char) ( ( (int)buchstabe-64+zahl)%26+64 );
}
```

Wenn die Funktion `verschiebe` mit einem Buchstaben und einer Zahl als Parametern aufgerufen wird, wird durch `(int)` der `buchstabe` vom Typ `char` zunächst in eine Zahl umgewandelt. Durch den modulo-Operator „%“ werden nur die 26 Zeichen des Alphabets erzeugt. Die nun berechnete Zahl wird wiederum durch `(char)` in einen Buchstaben konvertiert und mit `return` ans Hauptprogramm zurückgegeben.

Alternativ kann man einen Text verschlüsseln, indem man sämtliche Vokale durch andere Zeichen (Sonderzeichen, Bildzeichen, etc.) ersetzt.

```
bool istvok(char c)
{
    string menge „AEIOUaeiou“;
    return (menge.find(c)<=menge.length());
}
```

Hierzu wird in einer `bool`- Funktion geprüft, ob der definierte String `menge` einen Vokal enthält. Der Originalstring wird im Hauptprogramm nun an jeder Stelle daraufhin überprüft ob er ein Element des anderen Strings enthält. Ist das der Fall, wird ein Wahrheitswert (`true` oder `false`) ans Hauptprogramm zurückgegeben, welches diesen dann entsprechend verwertet.

```
int main(void)
{
    string zeile, ohne, codiert;
    cin >> zeile; ohne=codiert=zeile;
    for(i=0;i<zeile.length();i++) {
        if(istvok(zeile[i])) ohne[i]="🍏";
        else ohne[i]=zeile[i];
        codiert[i] = verschiebe( zeile[i], 7);
    } // for
    cout << ohne << endl << codiert;
}
```

```
Testlauf: Heute ist Dienstag
          H🍏🍏🍏 st D🍏🍏🍏st🍏🍏
```

In diesem Fall wurde nun jeder Vokal (da er sich im lokal vereinbarten String „menge“ der Funktion `istvok` befand) durch ein 🍏 ersetzt.

Prinzipiell können auch andere „Mengen“ durch beliebige andere Zeichen ersetzt werden.

Klassen

```
using namespace std; // ----- Klassen

class Zeit
{
private: // ----- lokal
    int stunden;
    int minuten;
    int erhoehe_mod60(int& mm, int aa);

// ----- öffentlich
public:
    int std() { return stunden; }
    int min() { return minuten; }
    void setzen (int s, int m) // direkte Deklaration
    {
        stunden = s % 24;
        minuten = m % 60;
    }
    Zeit() // Default- Konstruktor
    {
        time_t timer = time(0);
        struct tm* tblock = localtime(&timer);
        stunden = tblock->tm_hour;
        minuten = tblock->tm_min;
    }
    Zeit(int s, int m) // Konstruktor mit 2 Param.
    {
        stunden = abs(s) % 24;
        minuten = abs(m) % 60;
    }
    ~Zeit() {} // Destruktor
    void tickTack(); // Prototyp
    void tickTack(int anzahl); // Prototyp überladen

    friend ostream& operator <<
    (ostream& s, const Zeit& z); // Prototyp Operator
}; // class Zeit; Semikolon beachten

// Es folgen nun die ausserhalb der Klasse definierten
// Funktionen, Prozeduren und Operatoren
```

Kommentar

Sven M.

Die Klassen sind eine Erweiterung des Verbund- Datentyps, welcher mit dem Schlüsselwort struct beginnt.

Geöffnet wird eine Klasse durch class , gefolgt vom Namen der Klasse und „}“.
Es folgt die Vereinbarung der Attribute(Variablen) und Methoden (Konstruktoren, Destuktoren, Funktionen, Prozeduren, Operatoren).

Man unterteilt sie in zwei Gruppen , den lokalen Teil, welcher mit private: eingeleitet wird und dem öffentlichen Teil, welcher mit public: beginnt.

Die Variablen stunden, minuten und die Funktion erhoehe_mod60 des lokalen Bereichs in nebenstehendem Beispiel können nur innerhalb der Klasse und nicht außerhalb etwa im main -Programm aufgerufen werden.

Attribute und Methoden, die außerhalb der Klassendefinition benötigt werden, müssen als public deklariert werden. Die Deklaration kann innerhalb wie bei setzen aber auch ausserhalb wie bei tickTack erfolgen.

Eine spezielle Art der Deklaration ist der Konstruktor einer Klassenvariablen, etwa Zeit(), der sicherheitshalber durch einen Destruktor ~Zeit() ergänzt werden sollte.

Mit „Überladen“ bezeichnet man den Vorgang, daß eine bereits vorhandene Funktion, Prozedur oder Operator erweitert wird. Speziell der „<<“ - Operator, der bereits über using namespace std bei cout verwendet wird, muß als friend deklariert werden, um überladen werden zu können.

```

int Zeit::erhoehe_mod60(int& mm, int aa)
{   int tt = mm +aa;
    mm = tt % 60;
    return tt / 60;
} // erhoehe_mod60

void Zeit::tickTack()
{   int uebertrag = erhoehe_mod60(minuten,1);
    stunden = (stunden + uebertrag) % 24;
} // tickTack

void Zeit::tickTack(int anzahl)
{   int uebertrag = erhoehe_mod60(minuten, anzahl);
    stunden = (stunden + uebertrag) % 24;
} // tickTack

ostream& operator<< (ostream& s, const Zeit& z)
{   s.width(2); s.fill('0'); s << z.stunden << ":" ;
    s.width(2); s.fill('0'); s << z.minuten << "\n";
    return s;
} // operator <<

int main(void) // ----- Zeitrechnung
{   Zeit anf, aktuell, ende(15,25);

    anf.setzen(13,58);
    cout << "Anfangszeit -----> " ;
    cout << anf.std() << ":" << anf.min() << endl ;

    for ( int i=0; i<7; i++) {
        anf.tickTack(1); cout << anf;
    }
    cout << endl;
    for ( int i=0; i<6; i++) {
        anf.tickTack(5); cout << anf;
    }
    cout << "Aktuelle TagesZeit -----> " << aktuell;
    cout << "Feierabend -----> " << ende;

    return EXIT_SUCCESS;
} // main

```

Um eine Klasse abzuschließen, werden wie beim Ende eines Verbundes Klammer und Semikolon „};“ benötigt

Eine Deklaration außerhalb der Klasse benötigt die Kennzeichnung durch den Namen der Klasse und zwei „::“, auch allgemein in der Form „namespaceName::“ zum Zugriff auf einen anderen Namensbereich möglich

tickTack(int anzahl) überläd tickTack() dadurch, dass eine beliebige Anzahl von Takten zur Minutenanzahl addiert wird.

Der << Operator wird durch die formatierte Ausgabe von Zeiten überladen. setfill('0') bzw. fill('0') bewirkt, dass bei einstelligen Zahlen von vorne mit einer Null aufgefüllt wird.

Ausgabeprotokoll des nebenstehenden Beispiels:

```

Anfangszeit -----> 13:58
13:59
14:00
14:01
14:02
14:03
14:04
14:05

14:10
14:15
14:20
14:25
14:30
14:35
Aktuelle TagesZeit -----> 14:39
Feierabend -----> 15:25

```

Matrizen

vollständiges Programmab Seite 45

```
// Definition der Klasse----- Matrix -----
using namespace std;

const unsigned long N = 3;

class Matrix {
private:
    set<unsigned long> menge;// später für DET
    vector< vector<int> > matrix;
public:
    Matrix(); // ----- Konstruktoren
    Matrix(int x);
    Matrix(int min, int max);
    ~Matrix(); // ----- Destruktor

    void ausgabe(); // ----- Methoden
    void summe(Matrix a, Matrix b);
    void produkt(Matrix a, Matrix b);

}; // class Matrix

void trenner ( string ); // ----- Prototyp

Matrix::Matrix()
{
    vector <int> zeile(N, 0);
    for (unsigned long z = 1; z <= N; z++)
        matrix.push_back(zeile);
}

Matrix::Matrix(int x)
{
    vector <int> zeile(N, 0);
    for (unsigned long z = 1; z <= N; z++)
        matrix.push_back(zeile);
    for (unsigned long i=0; i<N; i++)
        matrix.at(i).at(i) = x;
}
```

Kommentar

Sandra R.

Definition der konstanten Variable N = 3 als unsigned long.
Der Einfachheit halber werden nur Matrizen mit gleicher Zeilen- und Spaltenzahl vereinbart.

private: Vereinbarung der Variablen, die nur innerhalb der Klasse gebraucht werden: menge, matrix.

Vereinbarung der menge als set von unsigned long und Vereinbarung der matrix als vector vom vector.

public: Vereinbarung von Variablen und Methoden (Funktionen bzw. Prozeduren), die außerhalb der Klasse, etwa in main, gebraucht werden: Konstruktor, ausgabe, summe, produkt.

Es werden im nebenstehenden Beispiel 3 Konstruktoren der Klasse Matix definiert: bei keinem Parameter die Nullmatrix, bei einem Parameter eine Matrix mit konstanter Hauptdiagonale und bei zwei Parametern eine Zufallsmatrix.

Die Methoden dienen der ausgabe, der summe bzw. dem produkt von Matrix a und b.

Erstellen der Nullmatrix

Jedem Element, welches durch seinen Zeilenindex (row, waage-recht) und seinen Spaltenindex (column, senkrecht) wird der Wert 0 zugeordnet.

Beispiel für eine Nullmatrix
mit 3 Zeilen und
3 Spalten:

0	0	0
0	0	0
0	0	0

Trickreich wird der Konstruktor der C++-Klasse vector verwendet, der eine ganze Zeile mit Nullen füllt.

```

Matrix::Matrix(int min, int max)
{
    vector <int> zeile(N, 0);
    for (unsigned long z = 1; z <= N; z++)
        matrix.push_back(zeile);
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++)
            matrix.at(z).at(s) = min+abs(rand())%(max-min+1);
    }
}

Matrix::~Matrix() { }

void Matrix::ausgabe()
{
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++)
            cout << setw(5) << matrix.at(z).at(s);
        cout << endl;
    }
    cout << endl;
}

void Matrix::summe(Matrix a, Matrix b)
{
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++)
            matrix.at(z).at(s) =
                a.matrix.at(z).at(s) + b.matrix.at(z).at(s);
    }
}

void Matrix::produkt(Matrix a, Matrix b)
{
    int summe;
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++) {
            summe = 0;
            for (unsigned long i=0; i<N; i++)
                summe = summe +
                    a.matrix.at(z).at(i) * b.matrix.at(i).at(s);
            matrix.at(z).at(s) = summe;
        }
    }
}

```

Erstellen der Hauptdiagonalen x 0 0
 Beispiel für eine Matrix : 0 x 0
 mit konstanter Hauptdiagonale 0 0 x

Erstellen der Zufallsmatrix
 Der Nullmatrix werden Zufallszahlen 6 4 1
 zwischen zwei Werten min, max 3 9 5
 zugeordnet, etwa $1 \leq \text{wert} \leq 9$ 6 8 2

Zeilenweise Ausgabe der Matrizenelemente, formatiert auf höchstens 5 Stellen.

Summe zweier Matrixen a und b:
 8 9 3 7 4 3 15 13 6
 6 4 1 + 6 3 1 = 12 7 2
 1 2 7 1 2 1 2 4 8

Die Zahlen der einzelnen Stellen von Matrix a werden zu den Zahlen auf den entsprechenden Stellen von Matrix b addiert.
 Bsp : $8+7 = 15$; $9+4=13$; $3+3=9$ etc.

Produkt zweier Matrizen a und b :
 8 9 3 7 1 2 113 41 28
 2 4 1 * 6 3 1 = 39 16 9
 1 2 3 1 2 1 22 13 7

Die Zeilen von Matrix a werden jeweils mit den Spalten von Matrix b multipliziert. So wird die erste Zeile (8 9 3) wird mit der ersten Spalte (7 6 3) nach folgender Vorschrift multipliziert : $(8*7+9*6+3*1 = 113)$, wodurch man den ersten Wert der Ergebnismatrix erhält. Multipliziert man die erste Zeile von a mit der zweiten und dritten Spalte von b , so erhält man die erste Zeile der Ergebnismatrix.

```

trenner ( string titel) // ----- Trennlinie mit Titel
{
    string linie=" ";
    for (int i=0; i<5*N; i++)
        linie = linie + "-"; linie = linie + " ";
    cout << linie << titel << endl;
} // trenner

```

```

int main(void)
{
    try {
        srand( time(NULL));
        Matrix m0(0, 9);
        Matrix m1(0, 9);
        Matrix m3, m4;

        trenner("Matrix M0 ");
        m0.ausgabe();

        trenner("Matrix M1 ");
        m1.ausgabe();

        trenner("Summe M0 + M1 prozedural");
        m3.summe(m0, m1);
        m3.ausgabe();

        trenner("Produkt M0 + M1 prozedural");
        m4.produkt(m0, m1);
        m4.ausgabe();

        return EXIT_SUCCESS;
    }

    catch (...) {cerr << "FAIL!";}
    return EXIT_FAILURE;
}

```

Definition einer Trennlinie zur besseren Unterscheidung der einzelnen Matrixen bei der Ausgabe.

Hauptprogramm :

Die Matrixen m0 und m1 werden mit dem Zufalls-Konstruktor erzeugt; `srand(time(NULL))` startet den Zufallsgenerator. Die Matrixen m3 und m4 werden mit dem Nullmatrix-Konstruktor erzeugt, dabei wird m3 die Summe und m4 das Produkt der Matrixen m0 und m1 zugeordnet.

Die vier Matrixen werden jeweils mit einem Trenner und einer Beschriftung zur Kennzeichnung der Funktion der jeweiligen Matrix ausgegeben.

Ausgabeprotokoll:

```

----- Matrix M0
  9   6   0
  6   7   5
  0   7   6

----- Matrix M1
  4   9   7
  1   1   5
  5   3   9

----- Summe M0 + M1 prozedural
 13  15   7
  7   8  10
  5  10  15

----- Produkt M0 + M1 prozedural
 42  87  93
 56  76 122
 37  25  89

```

Man beachte das trickreiche Abfangen von Laufzeitfehlern im Hauptprogramm.

Mehrzeilige Texte

Bedingte Zuweisungen

```
Using namespace std;

string liesStr(char bis) {
    const STRINGLAENGE = 256;
    char c[STRINGLAENGE];
    char endzeichen = (bis==0 ? '\n' : bis );
    string sss;

    cin.getline ( c, STRINGLAENGE-1, endzeichen );
    return sss.assign ( c );
} // liesStr

int main(void) { // Texteingabe
    string zeile, seite;

    zeile = liesStr ( 0 );
    seite = liesStr ( '#' );

    cout << zeile << endl;
    cout << seite << endl;

    return EXIT_SUCCESS;
} // Texteingabe
```

Kommentar

Mario S.

Mit dem Standardoperator cin

```
string zeile, seite;
cin >> zeile;
```

wird nur eine Zeichenkette bis zum nächsten Leerzeichen oder bis zum RETURN erfasst. Will man jedoch Texte mit mehreren Wörtern erfassen, welche durch Leerzeichen (Blank) getrennt sind, muss man ein Textende definieren, im Beispiel durch Eingabe von RETURN oder durch Eingabe eines #.

In der Funktion liesStr wird die bedingte Zuweisung

```
endzeichen = (bis==0 ? '\n' : bis );
```

verwendet, welche allgemein folgende Syntax hat:

Variable = (*Bedingung* ? *true-Zuweisung* : *false-Zuweisung*);

Dabei trennt das

„?“ die Bedingung (Frage) von den beiden Zuweisungen und „:“ trennt die Zuweisung für den ja-Fall von der Alternative.

```
zeile = liesStr ( 0 );
```

erfasst den Text bis zur Eingabe von RETURN.

```
seite = liesStr ( '#' );
```

erfasst den Text bis zur Eingabe von #.

alternativ: `getline(cin, zeile, '\n')`
`getline(cin, seite, '#')`

Beispiel: Folgedatum

```
using namespace std;

struct    tempus {
        int tag, monat, jahr; };
bool janein = false;

bool schaltjahr(int) ;//prototypen
tempus nachfolger (tempus);
tempus lies();
void schreib(tempus);

tempus lies()
{
    tempus dd; bool wdh;
    do {
        wdh=false; cout << endl;
        cout << "Tag      : "; cin >> dd.tag  ;
        cout << "Monat   : "; cin >> dd.monat;
        cout << "Jahr    : "; cin >> dd.jahr ;
        switch(dd.monat){
            case 1: case 3: case 5: case 7:
            case 8: case 10: case 12:
                if (dd.tag>31) wdh=true;break;
            case 4: case 6: case 9: case 11:
                if (dd.tag>30) wdh=true;break;
            case 2:
                if (dd.tag>29) wdh=true;
                if (!schaltjahr(dd.jahr) and (dd.tag>28) )
                    wdh=true;break;
        }
        if ( wdh) cout << "Können Sie kein richtiges
        Datum eingeben ?"<<endl;
    } while (wdh);

    return dd;
}

// lies
```

Kommenta

Martin H.

Der Verbund tempus definiert die ganzzahligen Komponenten tag, monat und jahr;

Definition von Prototypen für die Unterprogramme schaltjahr, nachfolger, lies und schreib.

Unterprogramm zum Einlesen von Tag , Monat und Jahr wdh = false um falsche Eingabe zu verhindern

Kommentierte Eingabe

Switch zum Abfragen der Monatslänge

Falls wdh im switch aufgetreten ist , Aufforderung zur Neueingabe vom Datum
Solange bis ein falsches Datum auftaucht

```

void schreib (tempus d)
{
    cout << "-----> ";
    cout << d.tag << ".";
    cout << d.monat << ".";
    cout << d.jahr << endl;
} // schreib

bool schaltjahr(int n)
{
    if(n % 4 == 0) {
        if(n % 100 == 0) {
            if(n % 400 == 0) return true;
            else return false;
        }else return true;
    }else return false;
} //schaltjahr

tempus nachfolger (tempus d)
{
    tempus dd;
    string laenge;
    dd=d;
    janein = true;
    switch(d.monat){
        case 1: case 3: case 5:
        case 7: case 8: case 10: case 12:
            laenge = "lang";break;
        case 4: case 6: case 9: case 11:
            laenge = "kurz";break;
        case 2:
            laenge = "februar";break;
        default: janein = false;
        cout << "Können Sie kein richtiges Datum
        eingeben ?";
    }
}

```

Unterprogramm zum schreiben eines Datums

Ausgabe von tag, monat und jahr

Unterprogramm zur Schaltjahrabfrage

Der Kern des gesamten Beispiels ist das
Unterprogramm zur Bestimmung des Nachfolgedatums:

Fallunterscheidung zur Einteilung der eingegebenen Monate in

lang (31 Tage),
kurz (30 Tage),
februar(28 bzw 29 Tage)

Wenn janein = false, wird zur erneuten Eingabe vom Datum
mit korrektem Monat aufgefordert

Die drei Fälle, lang, kurz bzw februar werden nun der Reihe nach
behandelt. Dabei muß stets geprüft werden, ob ein Tag
mitten im Monat (tag++)
am Monatsende (tag=1; monat++)
am Jahresende (tag=1, monat=1, jahr++) liegt.

```

dd.tag++;
if ( (laenge == "lang") && (dd.tag == 32) ){
    dd.tag = 1;
    if (dd.monat == 12) {
        dd.jahr++;
        dd.monat=1;
    }
    else dd.monat++;
}

if ( (laenge == "kurz") and (dd.tag == 31) ) {
    dd.tag = 1;
    dd.monat++;
}

if ( (laenge == "februar") ) {
    if ( schaltjahr(dd.jahr) and (dd.tag == 30)
) {
        dd.tag = 1;
        dd.monat++;
    }
    if ( !schaltjahr(dd.jahr) and (dd.tag == 29)
) {
        dd.tag = 1;
        dd.monat++;
    }
}
return dd;
} // nachfolger

int main(void) {
    tempus datum;
    do {
        datum = lies();
        datum = nachfolger(datum);
        if (janein)
            schreib(datum);
    } while(datum.jahr > 1582);

return EXIT_SUCCESS;

} // main Folgedatum

```

Am Anfang wird jeder Tag um eins erhöht, schlimmstenfalls kann tag=32 werden.

lang: Sonderfall Dezember

kurz:

Februar: Sonderfall Schaltjahr

Rückgabe des berechneten FolgeDatums.

Hauptprogramm:

datum ist eine Variable vom Typ tempus

do-Schleife:

Lesen des Datums

Bestimmung des Nachfolgers

Wenn das Datum korrekt ist wird datum geschrieben solange wiederholt wie das Jahr größer als 1582 ist

Beispiel: Schiffe versenken

```
using namespace std ;

const      N=10;
char  feld[N][N];

void  schiffe_setzen(int);
void  spielStand_ausgeben();
int  zufall (int,int);

int  zufall (int a, int b) {
    return (a + rand() % (b-a+1));
} // zufall

void  spielStand_ausgeben() {
    cout << "          ";
    for (int s=0; s<N; s++)
        cout << (char) (s+65) << ' ';
    cout << endl<< endl;
    for (int z=0; z<N; z++) {
        cout << setw(4) << z+1 << "  ";
        for ( int s=0; s<N; s++)
            cout << feld[z][s] << ' ';
        cout << endl;
    }
} // spielStand_ausgeben
```

Kommentar

Timo T.

Festlegen der Feldgröße
Koordinaten werden festgelegt,dabei wurde N vorher bereits
definiert.

Definieren der Variablen

Zufallsgenerator

Spielfeld wird ausgegeben, wobei s(spalten) die Buchstaben
und z(zeilen) die Zahlen der Spielfeldumrandung erzeugen.

Ausgabe des Feldes

```

void schiffe_setzen(int nn)
{
    int x,x1,x2,y,y1,y2;
    bool fehler;
    do {
        fehler=false;
        if (zufall(0,1)==0) {
            x1=zufall(0,N-nn);
            x2=x1+nn-1;
            y1=y2= zufall(0,N-1);
        }
        else {
            x1=x2 = zufall(0,N-1);
            y1=zufall(0,N-nn);
            y2=y1+nn-1;
        }
        for (x=x1; x<=x2; x++)
            for (y=y1; y<=y2; y++)
                fehler=fehler or (feld[y][x]!='x');
        } while (fehler);
        for (x=x1; x<=x2; x++)
            for (y=y1; y<=y2; y++) feld[y][x]=(char)(48+nn);
    } // schiffe_setzen

int main (void) {

    int zeile, spalte;

    srand(time(NULL));

    for (zeile= 0; zeile <N; zeile++)
        for (spalte=0; spalte<N; spalte++)
            feld[zeile][spalte]='x';

    schiffe_setzen(5);
    schiffe_setzen(4);
    schiffe_setzen(3);
    schiffe_setzen(3);

    spielStand_ausgeben();

    return EXIT_SUCCESS;
}

```

Variablen der Schiffe

do-Schleife, um die Schiffe zufällig waagrecht oder senkrecht zu setzen, wobei zufällig die 0 für waagrecht und die 1 für senkrecht gewählt wird.

Die erste for-Schleife testet, ob Überschneidungen der Schiffe vorliegen. Kommt es zu Überschneidungen(fehler=true), so wird erneut versucht, die Schiffe zu setzen.

Die zweite for-Schleife setzt die Schiffe .

Hauptprogramm

Startet den Zufallsgenerator

Das gesamte Anzeigefeld wird mit ‚x‘ gefüllt:

Anzahl und Größe der Schiffe wird festgelegt

Ausgabe des Spielfeldes mit den zufällig gesetzten Schiffen

Aufgabe:

Vervollständigen Sie das Programm, so dass sie gegen den Computer spielen.

Anleitung:

Im main- Programm müssen die Variablen

```
char  anzeige[N][N];  
string eingabe;  
int x;  
int y;
```

vereinbart werden. Dabei soll etwa aus der Eingabe „C7“ die Koordinaten x=2 und y=6 berechnet werden.

In einer Schleife muß dann rückgemeldet werden, ob der Fall

Treffer (•)

Wasser (-)

Schiff versenkt (ziffer)

realisiert wurde.

In der Spielstansanzeige müssen geeignete Symbole für jeden Schuss eingetragen werden.

Neben dem `feld`, in dem die gesetzten Schiffe gespeichert sind, muss ein zweites Feld `anzeige` erzeugt werden, in dem der aktuelle Spielstand gespeichert ist.

Dateiverarbeitung

zeichenweise

```
void zeichenweise() {
    ifstream alt("probe.TXT");
    ofstream neu("probe_neu_char.TXT");
    char zeichen;
    while( !alt.eof() ) {
        alt.get( zeichen);
        cout << zeichen;
        neu << zeichen;
    }
    neu.close();
    alt.close();
} // zeichenweise
```

wortweise

```
void wortweise() {
    string wort, altName, neuName;
    altName="probe.TXT";
    neuName=altName+"_neu_word";
    ifstream alt(altName.c_str());
    ofstream neu(neuName.c_str());
    while( !alt.eof() ) {
        alt >> wort;
        cout << wort << " - ";
        neu << wort << " - ";
    }
    neu.close();
    alt.close();
} // wortweise
```

Kommentar

MSP

Auf der Festplatte liege unter

probe.TXT

ein Probetext, der auf drei Arten gelesen und jeweils in eine neue Datei geschrieben wird:

probe_neu_char.TXT
probe_neu_word.TXT
probe_neu_line.TXT

ifstream alt(datName);
öffnet die Datei zum Lesen.

ofstream alt(datName);
öffnet die Datei zum Schreiben.

datname muss ein c-String und nicht ein Element der C++Klasse string sein. Der Methodenaufruf .c_str() konvertiert einen C++string in einen c-string. Das Verketteten sowie die bekannten String-Operationen sind nur für C++Strings definiert.

char zeichen;
string wort;
char zeile[256];

Lesen und Schreiben erfolgt mit den Methoden

.get(zeichen) // zeichenweise

.get(zeile) // zeilenweise

Das wortweise Lesen und Schreiben erfolgt mit dem input-Stream, welcher automatisch Leerzeichen und Zeilenvorschub als Trennzeichen interpretiert. Im Beispielprogramm werden daher die Leerzeichen durch „-“ ersetzt

zeilenweise

```
void zeilenweise() {
    ifstream alt("probe");
    ofstream neu("probe_neu_line.TXT");
    char cFeld[256];
    int nr=0;
    while( !alt.eof() ) {
        nr++;
        alt.getline(cFeld,256);
        cout << setw(3) << nr << " " << cFeld << endl;
        neu << setw(3) << nr << " " << cFeld << endl;
    }
    neu.close();
    alt.close();
} // zeilenweise
```

Hilfsroutinen zur Konvertierung

```
string num2str(int num) {
    ostringstream ostr;
    ostr << num;
    return ostr.str();
} // num2str

int str2num(string sss) {
    int num;
    istringstream iStr;
    iStr.str(sss);
    iStr >> num;
    return num;
} // str2num
```

Die Zeilen werden im Beispielprogramm gelesen und mit Zeilennummer versehen auf den Bildschirm und in die Datei probe_neu_line.TXT geschrieben.

```
string zeile;
char cFeld[256];
```

Die Methode `getline(cFeld)` erfordert als Parameter ein `cFeld`, welches bei Bedarf mit `zeile.assign(cFeld)` in einen `C++String` konvertiert werden kann.

Die beiden Hilfsroutinen zeigen, wie man mit Hilfe der „streams“ `integer` in `strings` und umgekehrt verwandeln kann.

Will man beliebige Typen konvertieren, so geht man folgendermaßen vor:

```
template<class zahl> string num2str(zahl num) {
    ostringstream ostr;
    ostr << num;
    return ostr.str();
} // num2str

template<class zahl> zahl str2num(string sss) {
    zahl num;
    istringstream iStr;
    iStr.str(sss);
    iStr >> num;
    return num;
} // str2num
```

Beispiele aus der Klasse algorithm

```
using namespace std;

string zeile = "Momo mag das Waschmittel Omo und nascht
mittags Schokolade";

vector<string>    vorrat;
vector<int>      anzahl;

string num2str ( int );
void tupel_zaehlen( unsigned long );

string num2str ( int num ) {
    ostringstream ostr;
    ostr << num;
    return ostr.str();
} // num2str

void tupel_zaehlen( unsigned long n ) {
    string teil="";
    unsigned long i, nr;
    for (i=0; i<zeile.length(); i++) {
        if ( isalpha(zeile.at(i) ) ) {
            teil = teil + zeile.at(i);
            if ( teil.length() > n ) teil = teil.erase(0,1);
        } else teil = "";
        if ( teil.length() == n ) {
            vector<string>::const_iterator
            pos = find( vorrat.begin(), vorrat.end(), teil);
            if ( pos == vorrat.end() ) {
                vorrat.push_back(teil);
                anzahl.push_back(1);
            }
            else {
                nr = pos - vorrat.begin() ;
                anzahl.at( nr )++
            }
        } // if Länge = n
    } // for
} // tupel_zaehlen
```

Kommentar zum Suchen

MSP

Aufgabe:

Der nicht unbedingt sinnvolle Text

```
"Momo mag das Waschmittel Omo und
nascht mittags Schokolade"
```

soll statistisch auf Bigramme, Trigramme,... untersucht werden.

Hierzu wird zunächst das Unterprogramm `tupel_zaehlen` erzeugt, welches nur reine Buchstabenketten

```
if ( isalpha(zeile.at(i) ) ) ) .....
der Länge n im Vektor vorrat speichert und bei mehrmaligem
Vorkommen im Vektor anzahl zählt.
```

Im String `teil` werden die Buchstaben solange zusammengesetzt, bis die Länge n erreicht ist. Ist die Länge größer als n, wird der erste Buchstabe mit `teil.erase(0,1)` gelöscht.

Hat ein `teil`-String genau die Länge n, so wird mit `find` geprüft, ob sich `teil` bereits im `vorrat` befindet:
falls nein, wird die betreffende `anzahl` mit 1 initialisiert und `teil` mit `vorrat.push_back(teil)` im `vorrat` aufgenommen;
falls ja, wird die `anzahl` durch die Anweisung `anzahl.at(pos - vorrat.begin())++` erhöht.

Die Methode `find` (`anfang`, `ende`, `suchbegriff`) erfordert für `anfang` und `ende` Zeiger auf das erste und letzte Element des Suchfeldes. Der `suchbegriff` muss dem Typ der Elemente des Suchfeldes entsprechen. Im Beispiel werden die Zeiger auf `vorrat` durch die Methoden

```
anfang = vorrat.begin()
ende   = vorrat.end()
```

bereitgestellt.

```

int main(void) { // algorithm & vector

    unsigned long tupelLaenge = 3;
    string tw = "SUCHEN UND SORTIEREN";
    vorrat.push_back(tw);
    anzahl.push_back(0);

    for (unsigned long i=0; i<zeile.length(); i++)
        zeile.at(i) = toupper( zeile.at(i) );
    cout << zeile << endl << endl;

    tupel_zaehlen(tupelLaenge);

    cout << endl;

    for (unsigned long i=0; i<vorrat.size(); i++) {
        if (anzahl.at(i)>0) {
            vorrat.at(i) = vorrat.at(i) + " " +
                num2str(anzahl.at(i));
            cout << vorrat.at(i) << " mal" << endl;
        } else cout << vorrat.at(i) << endl;
    } // for

    cout << "----- sortiert" << endl;
    sort ( vorrat.begin(), vorrat.end() );

    for (unsigned long i=0; i<vorrat.size(); i++) {
        cout << vorrat.at(i);
        if (vorrat.at(i)==tw) cout << endl;
        else cout << " mal" << endl;
    }
    cout << "----- TestWort binäre Suche ";
    if (binary_search(vorrat.begin(),vorrat.end(),tw )) {
        cout << "an Position " <<
            find( vorrat.begin(),vorrat.end(),tw)-
vorrat.begin();
        cout << " gefunden !" << endl;
    } else cout << " nicht gefunden !" << endl;

    return EXIT_SUCCESS;

} // algorithm & vector

```

Kommentar zum Sortieren MSP

Im Hauptteil wird zunächst ein teilWort `tw` in den `vorrat` aufgenommen und alle Buchstaben der `zeile = "Momo mag das Waschmittel` in Großbuchstaben umgewandelt und zur Kontrolle ausgegeben.

`tupel_zaehlen(tupelLaenge);`
 ruft die oben beschriebene Routine (hier für Tripel) auf.

Vor der Ausgabe und der Weiterverarbeitung werden die Buchstabentripel aus dem `vorrat` und die zugehörige `anzahl` verkettet: so kann später der Gesamtstring aus Tripel und Anzahl einfach alphabetisch sortiert werden.

`sort (anfang, ende);`

Die Syntax des C++ Sortierverfahrens `sort` mit dem Aufwand $n \cdot \lg(n)$ ist sehr einfach, wobei, wie bereits oben notiert, für die beiden Zeiger gilt:

```

    anfang = vorrat.begin()
    ende   = vorrat.end()

```

Nun, da der Vektor `vorrat` bereits sortiert ist, kann im Gegensatz zum langsamen sequentiellen `find` - Suchen im Unterprogramm `tupel_zaehlen` sehr schnell binär mit dem Aufwand $\lg(n)$ nach dem TestWort `tw` gesucht werden mit der Funktion

```

    binary_search ( anfang ,ende, suchbegriff )

```

welche `true` zurückgibt für gefunden und `false` für nicht gefunden.

Möchte man jedoch wissen, an welcher Position der `suchbegriff` im Vektor `vorrat` steht, so muss wieder mit dem langsamen `find` mit dem (Aufwand $n/2$) gesucht werden, welches einen Zeiger auf die gefundene Position zurückgibt. Mit dem Trick, die Differenz zweier Zeiger(adressen) zu bilden, erhält man jedoch eine `int` - Zahl:

```

    position = find(anfang,ende,suchbegriff)- anfang;

```

```

#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <stack>
#include <set>
#include <iterator>

// ----- Matrizen und Vektoren

const unsigned long N = 4;
const string fixed="";

class Vektor {
private:
    vector<double> vektor;

public:
    Vektor(); // ----- Konstruktoren
    Vektor(int z0);
    Vektor(int min, int max);
    ~Vektor(); // ----- Destruktor

    double at(unsigned long z) const; // Methoden
    void put(unsigned long z, const double& wert);
    double betrag();
    Vektor& operator = ( const Vektor& m );// Operatoren

    friend Vektor operator + ( const Vektor& a, const
Vektor& b);
    friend Vektor operator - ( const Vektor& a, const
Vektor& b);
    friend double operator * ( const Vektor& a, const
Vektor& b);
    friend Vektor operator * ( const Vektor& a, const
double& b);

    friend istream& operator >> ( istream& ist, Vektor&
v);
    friend ostream& operator << ( ostream& ost, const
Vektor& v);
}; // class Vektor

```

```

class Matrix {
private:
    set<unsigned long> menge;
    vector< vector<double> > matrix;
    double determinante(unsigned long anz, unsigned long
z, unsigned long s);
    void unterDeterminante(unsigned long z, unsigned
long s_gestrichen, set<unsigned long> z_gestrichen);
    unsigned long pivot (int z0);
    void zeilentausch(int z1, int z2);
public:
    Matrix(); // ----- Konstruktoren
    Matrix(int x);
    Matrix(double d);
    Matrix(int min, int max);
    ~Matrix(); // ----- Der Destruktor
    void ausgabe(Matrix e);// ----- Methoden
    double at(unsigned long z, unsigned long s) const;
    double det();
    Matrix ersetze(int sp, const Vektor& v);
    Matrix inverse();
    void loesche(unsigned long z0);
    void transponiere();

    Matrix& operator = ( const Matrix& m );// Operatoren

    friend Matrix operator + ( const Matrix& a, const
Matrix& b);
    friend Matrix operator - ( const Matrix& a, const
Matrix& b);
    friend Matrix operator * ( const Matrix& a, const
double& b);
    friend Matrix operator * ( const Matrix& a, const
Matrix& b);
    friend Vektor operator * ( const Matrix& a, const
Vektor& b);

    friend istream& operator >> ( istream& ist, Matrix&
m);
    friend ostream& operator << ( ostream& ost, const
Matrix& m);
}; // class Matrix

```

```

// Deklarationen der Klasse Matrix

void Matrix::unterDeterminante(unsigned long z, unsigned
long s_gestrichen, set<unsigned long> z_gestrichen)
{
    set<unsigned long> hilf;
    hilf = z_gestrichen;
    hilf.insert(z);
    cout << endl;
    for (unsigned long z=0; z<N; z++) {
        if (hilf.count(z) == 0) {
            cout << "|";
            for (unsigned long s=0; s<N; s++)
                if (s>s_gestrichen) cout << setw(6) <<
setprecision(2) << fixed << matrix.at(z).at(s);
            cout <<" |" << endl;
        }
    }
    cout << endl;
} // Matrix::unterDeterminante

```

```

double Matrix::determinante(unsigned long rang, unsigned
long z, unsigned long s)
{ // Streichen der Zeile z, Entwickeln nach Spalte s,
1.Aufruf mit rang=N, z=N, s=0.
    double sum, buffer;
    int sign = -1;
    menge.insert(z);
    if (rang == 1) {
        for (unsigned long i=0; i<N; i++) {
            if (menge.count(i) == 0) buffer =
matrix.at(i).at(s);
        }
    } else {
        sum = 0;
        for (unsigned long i=0; i<N; i++) {
            if (menge.count(i) == 0) { // falls i nicht
Element der Menge
                sign = -sign;
                // unterDeterminante(i,s,menge);
                sum = sum + sign * matrix.at(i).at(s) *
this->determinante(rang-1, i, s+1);
            }
        }
        buffer = sum;
    }
    menge.erase(z);
    return buffer;
} // Matrix::determinante

Matrix::Matrix()
{
    vector <double> zeile(N, 0);
    for (unsigned long z = 1; z <= N; z++)
matrix.push_back(zeile);
}

```

```

Matrix::Matrix(int x)
{
    vector <double> zeile(N, 0);
    for (unsigned long z = 1; z <= N; z++)
matrix.push_back(zeile);
    for (unsigned long i=0; i<N; i++) matrix.at(i).at(i)
= static_cast<double>(x);
}

Matrix::Matrix(double d)
{
    vector <double> zeile(N, 0);
    for (unsigned long z = 1; z <= N; z++)
matrix.push_back(zeile);
    if (d==1.0*N)
        for (unsigned long i=0; i<N; i++)
matrix.at(i).at(i) = i+1;
    else
        for (unsigned long z=0; z<N; z++)
            for (unsigned long s=0; s<N; s++)
matrix.at(z).at(s) = (z+1)*10 + s+1;
}

Matrix::Matrix(int min, int max)
{
    vector <double> zeile(N, 0);
    for (unsigned long z = 1; z <= N; z++)
matrix.push_back(zeile);
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++)
matrix.at(z).at(s) = min + abs(rand())%(max-min+1);
    }
}

Matrix::~Matrix()
{
}

```

```

void Matrix::ausgabe(Matrix e)
{
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++) cout <<
setw(8) << setprecision(2) << fixed <<
matrix.at(z).at(s);
        cout << " | ";
        for (unsigned long s=0; s<N; s++) cout <<
setw(8) << setprecision(2) << fixed <<
e.matrix.at(z).at(s);
        cout << endl;
    }
    cout << endl;
} // Ausgabe zweier Matrizen zur Kontrolle der Inversion

double Matrix::at(unsigned long z, unsigned long s)
const
{
    return matrix.at(z).at(s);
} // at

double Matrix::det()
{
    menge.clear();
    return this->determinante(N, N, 0);
} // det()

Matrix Matrix::ersetze(int sp, const Vektor& v)
{
    for (unsigned long z=0; z<N; z++)
matrix.at(z).at(sp) = v.at(z);
    return *this;
} // ersetze in einer Matrix die Spalte sp durch den
Vektor ve

```

```

Matrix Matrix::inverse()
{
    unsigned long i, s, z, p;
    double faktor;
    Matrix einheits(1), original;
    vector< vector<double> > m=matrix,
e=einheits.matrix;
    original.matrix = m;
    for (i=0; i<N-1; i++) { // Nullen unterhalb der
Diagonalen erzeugen
        original.matrix = m;
        einheits.matrix = e;
        p = original.pivot(i);
        if ( p!=i ) {
            original.zeilentausch(i,p);
            einheits.zeilentausch(i,p);
            m = original.matrix;
            e = einheits.matrix;
        }
        for (z =i+1; z<N; z++) {
            faktor = m.at(z).at(i)/m.at(i).at(i);
            for (s = 0 ; s<N; s++) e.at(z).at(s) =
e.at(z).at(s) - e.at(i).at(s)*faktor;
            for (s = i ; s<N; s++) m.at(z).at(s) =
m.at(z).at(s) - m.at(i).at(s)*faktor;
        }
    }
    for (i=N-1; i>0; i--) { // Nullen oberhalb der
Diagonalen erzeugen
        for (int z =i-1; z>=0; z--) {
            faktor = m.at(z).at(i)/m.at(i).at(i);
            for (int s = N-1 ; s>=0; s--) e.at(z).at(s)
= e.at(z).at(s) - e.at(i).at(s)*faktor;
            for (int s = i ; s>=0; s--) m.at(z).at(s)
= m.at(z).at(s) - m.at(i).at(s)*faktor;
        }
    }
    for (z = 0; z<N; z++) { // Diagonalen normieren
        for (s = 0; s<N; s++) e.at(z).at(s) =
e.at(z).at(s) / m.at(z).at(z);
        for (s = 0; s<N; s++) m.at(z).at(s) =
m.at(z).at(s) / m.at(z).at(z);

```

```

    }
    einheits.matrix = e;
    original.matrix = m;
    return einheits;
} // inverse

unsigned long Matrix::pivot (int z0)
{
    unsigned long z, p=z0;
    double a, min = matrix.at(z0).at(z0);
    if (min<0) min = -min;
    for (z=z0+1; z<N; z++) {
        a = matrix.at(z).at(z0);
        if ( a < 0 ) a = -a;
        if ( (a>0) and (a<1) ) a = 1/a;
        if ( a<min ) { min = a; p = z; }
    }
    return p;
} // pivot-Element suchen

void Matrix::zeilentausch(int z1, int z2)
{
    double puffer;
    for (unsigned long s=0; s<N; s++) {
        puffer = matrix.at(z1).at(s);
        matrix.at(z1).at(s) = matrix.at(z2).at(s);
        matrix.at(z2).at(s) = puffer;
    }
} // zeilentausch

inline Matrix& Matrix::operator = ( const Matrix& m)
{
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++)
matrix.at(z).at(s) = m.matrix.at(z).at(s);
    }
    return *this;
} // operator =

```

```

inline Matrix operator + ( const Matrix& a, const
Matrix& b)
{   Matrix c;
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++)
            c.matrix.at(z).at(s) = a.matrix.at(z).at(s)
+ b.matrix.at(z).at(s);
    }
    return c;
} // operator +

inline Matrix operator - ( const Matrix& a, const
Matrix& b)
{   Matrix c;
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++)
            c.matrix.at(z).at(s) =
a.matrix.at(z).at(s) - b.matrix.at(z).at(s);
    }
    return c;
} // operator -

inline Matrix operator * ( const Matrix& a, const
Matrix& b)
{   double summe;
    Matrix c;
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++) {
            summe = 0;
            for (unsigned long i=0; i<N; i++)
                summe = summe + a.matrix.at(z).at(i) *
b.matrix.at(i).at(s);
            c.matrix.at(z).at(s) = summe;
        }
    }
    return c;
} // operator *

```

```

inline Vektor operator * ( const Matrix& a, const
Vektor& b)
{   double summe;
    Vektor c;
    for (unsigned long z=0; z<N; z++) {
        summe = 0;
        for (unsigned long i=0; i<N; i++) summe = summe
+ a.matrix.at(z).at(i) * b.at(i);
        c.put(z,summe);
    }
    return c;
} // operator *

inline Matrix operator * ( const Matrix& a, const
double& b)
{   Matrix c;
    for (unsigned long z=0; z<N; z++)
        for (unsigned long s=0; s<N; s++)
            c.matrix.at(z).at(s) = a.matrix.at(z).at(s) * b;
} // operator *

inline istream& operator >> ( istream& ist, Matrix& mmm
)
{
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++) cout << "a" <<
z+1 << s+1 << " ";
        cout << " bitte durch BLANKS getrennteingeben,
dann RETURN" << endl << endl;
        for (unsigned long s=0; s<N; s++) ist >>
mmm.matrix.at(z).at(s);
        cout << endl;
    }
    cout << endl;
    return ist;
} // operator >>

```

```

inline ostream& operator << ( ostream& ost, const
Matrix& mmm )
{   double wert;
    for (unsigned long z=0; z<N; z++) {
        for (unsigned long s=0; s<N; s++) {
            wert= mmm.matrix.at(z).at(s);
            if ( abs(static_cast<int>(10000*wert))<1 )
wert=0;
                ost << setw(8) << setprecision(2) << fixed
<< wert;
            }
        ost << "\n";
    }
    return ost;
} // operator <<

// Deklarationen der Klasse Vektor

Vektor::Vektor()
{
    for (unsigned long i=0; i<N; i++)
vektor.push_back(0);
}

Vektor::Vektor(int min, int max)
{
    for (unsigned long i=0; i<N; i++) vektor.push_back(
min + abs(rand())%(max-min+1) );
}

Vektor::~Vektor()
{
}

double Vektor::at(unsigned long z) const
{
    return vektor.at(z);
} // at

```

```

void Vektor::put(unsigned long z, const double& wert)
{
    vektor.at(z) = wert;
} // put

inline istream& operator >> ( istream& ist, Vektor& v)
{
    for (unsigned long z=0; z<N; z++) {
        cout << "z" << z+1 << " = ";
        ist >> v.vektor.at(z);
    } cout << endl;
    return ist;
} // operator >>

inline ostream& operator << ( ostream& ost, const
Vektor& v)
{
    for (unsigned long z=0; z<N; z++) {
        ost << setw(8) << setprecision(2) << fixed <<
v.vektor.at(z) << "\n";
    }
    return ost;
} // operator <<

inline Vektor& Vektor::operator = ( const Vektor& v)
{
    for (unsigned long z=0; z<N; z++) vektor.at(z) =
v.vektor.at(z);
    return *this;
} // operator =

inline Vektor operator + ( const Vektor& a, const
Vektor& b)
{   Vektor c;
    for (unsigned long z=0; z<N; z++) {
        c.vektor.at(z) = a.vektor.at(z) +
b.vektor.at(z);
    }
    return c;
} // operator +

```

```

inline Vektor operator - ( const Vektor& a, const
Vektor& b)
{   Vektor c;
    for (unsigned long z=0; z<N; z++) {
        c.vektor.at(z) = a.vektor.at(z) -
b.vektor.at(z);
    }
    return c;
} // operator -

inline double operator * ( const Vektor& a, const
Vektor& b)
{   double summe=0;
    for (unsigned long i=0; i<N; i++) summe = summe +
a.vektor.at(i) * b.vektor.at(i);
    return summe;
} // operator *

inline Vektor operator * ( const Vektor& a, const
double& b)
{   Vektor c;
    for (unsigned long z=0; z<N; z++)
        c.vektor.at(z) = a.vektor.at(z) * b;
    return c;
} // operator *

// sonstige Deklarationen

void trenner ( string );
void trenner ( string titel) // -----
-- Trennlinie mit Titel -----
{   string linie=" ";
    for (unsigned long i=0; i<8*N; i++) linie=linie + "-
"; linie = linie + " ";
    cout << endl << linie << titel << endl;
} // trenner

```

```

void Matrix::loesche(unsigned long z0)
{
    for (unsigned long s=0; s<N; s++)
matrix.at(z0).at(s) = 0;
} // Matrix::loesche(unsigned long z0)

void Matrix::transponiere()
{
    double hilf;
    for (unsigned long z=1; z<N; z++)
    for (unsigned long s=0; s<z; s++) {
        hilf = matrix.at(z).at(s);
        matrix.at(z).at(s) = matrix.at(s).at(z);
        matrix.at(s).at(z) = hilf;
    }
} // Matrix::transponiere()

Vektor::Vektor(int z0)
{
    for (unsigned long i=0; i<N; i++) vektor.push_back(
static_cast<double>(i+z0) );
} // Vektor(int z0)

double Vektor::betrag()
{
    double wert=0;
    for (unsigned long z=0; z<N; z++) wert = wert +
vektor.at(z)*vektor.at(z);
    return sqrt(wert);
} // Vektor::betrag()

```

```

// Hauptprogramm
int main(void)
{
    // srand( time(NULL));

    int fall = 3;

    try {
        Matrix m1, m2, m3, mat_1(1,9), mat_2(1,9),
mat_3, inv_3;
        Vektor v1, v2(10), v3(1,9), b, vek(3);
        double d, det_3;

        switch (fall) {

            case 1:    cout << "1. Fall" << endl;
                    cout << mat_1 + mat_2 << endl;
                    cin >> mat_3; det_3 = mat_3.det();
                    inv_3 = mat_3.inverse();
                    cout << mat_3*det_3 << endl;
                    cout << mat_3*inv_3 << endl;
                    cout << mat_3 << endl;
                    mat_3.loesche(1);          cout << mat_3
<< endl;
                    mat_3.transponiere();    cout << mat_3 <<
endl;
                    cout << vek << vek.betrag() << endl;
                    break;

            case 2:  cout << "2. Fall" << endl;
                    cin >> m1;
                    cin >> b;
                    trenner("Original");
                    cout << m1 << endl;
                    trenner("Determinante");
                    d = m1.det();
                    cout << d << endl;
                    trenner("Lösung mittels inverser Matrix");
                    cout << m1.inverse()*b;
                    trenner("Lösung mittels Cramerscher Regel");
                    for (int i=0; i<N; i++) {
                        m2=m1;
                        cout << "x" << i+1 << " = " <<
m2.ersetze(i,b).det() / d << endl;
                    }
                    break;

            case 3:  cout << "3. Fall" << endl;
                    m1 = mat_1.inverse();
                    trenner("ZufallsMatrix"); cout << mat_1 <<
endl;
                    trenner("InverseMatrix"); cout << m1 <<
endl;
                    trenner("ProduktMatrix"); cout << m1*mat_1
<< endl;
                    break;

        } // switch

        trenner("@ MSP");
        // return EXIT_SUCCESS;
    }
    catch (...) {cerr << "FAIL!";}

    return EXIT_FAILURE;
} // main

```

```

// Klasse zur Demo von Vererbung
// Beachte: Konstr. und Destrukt. werden nicht vererbt
using namespace std;

class Monster {
protected:
    string bezeichnung;
    string spezialitaet;
public:
    Monster(); // Default
    Monster(const string& name); // Konstruktor
    ~Monster(); // Destruktor
    void vorstellen(); // Methoden
    void nummerieren(); // werden vererbt
    void verabschieden(int n);
}; // class Monster

class Beisser : public Monster { // abgeleitet aus
    der class Monster // vererbt wurden:
public:
    vorstellen, nummerieren
    Beisser(const string& name);
    ~Beisser() ;
    void quadrieren(); // Methoden
    void wuenschen(string was); // werden vererbt
}; // class Beisser

class Vampir : public Beisser { // abgeleitet aus
    der class Beisser // vererbt wurden:
public:
    quadrieren, wuenschen
    Vampir(const string& name);
    ~Vampir() ;
    void randomieren(); // Methoden
    void beichten(int anz); // werden vererbt
}; // class Beisser

Monster::Monster() { // Konstruktor
    bezeichnung = "niemand";
    spezialitaet= "nichts";
    cout << "Monster " << bezeichnung << " ist da !" <<
endl; }

```

```

Monster::Monster(const string& name) { // Konstruktor
    bezeichnung = bezeichnung + name;
    spezialitaet= spezialitaet + "nichts ";
    cout << endl << "Monster " << bezeichnung << " ist da
!" <<endl;
}

Monster::~Monster() { // Destruktor
    cout << "Monster " << bezeichnung << " verschwindet
!" << endl<<endl;
}

inline void Monster::vorstellen() {
    cout << endl << "Ich bin " << bezeichnung << " und
kann " << spezialitaet << endl;
} // Monster::vorstellen

inline void Monster::nummerieren() {
    for (int i=1; i<10; i++) cout << endl << i << ".
Monster"; cout << endl;
} // Monster::nummerieren

inline void Monster::verabschieden(int n) {
    for (int i=0; i<n; i++)
        cout << bezeichnung << " verabschiedet sich zum "
<< i+1 << ". Mal"<< endl;
} // Monster::nummerieren

Beisser::Beisser(const string& name) : Monster(name) {
    // Konstruktor
    spezialitaet = "beissen";
    cout << "Beisser " << bezeichnung << " ist da !" <<
endl;
}

Beisser::~Beisser() { // Destruktor
    cout << "Beisser " << bezeichnung << " verschwindet
!" << endl;
}

```

```

inline void Beisser::quadrieren() {
    for (int i=10; i<21; i++) cout << endl << i*i << ".
Beisser"; cout << endl;
} // Beisser::quadrieren

inline void Beisser::wuenschen(string was) {
    cout << bezeichnung << " wünscht sich " << was <<
endl;
} // Beisser::wuenschen

Vampir::Vampir(const string& name) : Beisser(name) {
// Konstruktor
    spezialitaet = spezialitaet + " und saugen ";
    cout << "Vampir " << bezeichnung << " ist da !" <<
endl;
}

Vampir::~Vampir() {
    // Destruktor
    cout << "Vampir " << bezeichnung << " verschwindet
!" << endl;
}

inline void Vampir::randomieren() {
    for (int i=0; i<10; i++) cout << endl <<
400+rand()%600 << ". Vampir"; cout << endl;
} // Vampir::randomieren

inline void Vampir::beichten(int anz) {
    cout << bezeichnung << " hat heute Nacht " << anz <<
" Opfer vernascht" << endl;
} // Vampir::beichten

int main() // ----- Hauptprogramm -----
{
    Monster untier("Asino");
    Beisser bello("Bello");
    Vampir vamp("Cornus");

```

```

vamp.vorstellen();
vamp.wuenschen("fette Beute");
vamp.beichten(7);
vamp.verabschieden(5);

cout << endl << "..... E N D E main .....";
cout << endl<< endl;

return EXIT_SUCCESS;

} // main

Monster Asino ist da !

Monster Bello ist da !
Beisser Bello ist da !

Monster Cornus ist da !
Beisser Cornus ist da !
Vampir Cornus ist da !

Ich bin Cornus und kann beissen und saugen
Cornus wünscht sich fette Beute
Cornus hat heute Nacht 7 Opfer vernascht
Cornus verabschiedet sich zum 1. Mal
Cornus verabschiedet sich zum 2. Mal
Cornus verabschiedet sich zum 3. Mal
Cornus verabschiedet sich zum 4. Mal
Cornus verabschiedet sich zum 5. Mal

..... E N D E main .....

Vampir Cornus verschwindet !
Beisser Cornus verschwindet !
Monster Cornus verschwindet !

Beisser Bello verschwindet !
Monster Bello verschwindet !

Monster Asino verschwindet !

```

Literatur

- [Bas03] Peter Bastian. *Informatik I: Programmieren und Softwaretechnik*, 2003.
<http://hal.iwr.uni-heidelberg.de/lehre/inf1-ws02/index.html>
- [Lou01] Dirk Louis. *C / C++ New Reference*. Markt + Technik, 2001.
- [Lou02] Dirk Louis. *C / C++ Kompendium*. Markt + Technik, 2002.
- [Str00] Bjarne Stroustrup. *Die C++ Programmiersprache*. Addison-Wesley, 2000.