2 Grundlegende Begriffe, Teil 1

2.1 Teilsprachen

2.2 Ein einfaches C++-Programm

2.3 Grundlegende Datentypen

Logischer Datentyp

Ganzzahlen

Gleitpunktzahlen

Zeichen

Zeiger

C-Arrays

C++ <array>

Referenzen

Aufzählungen

Komplexe Zahlen

In C++ kann man fünf Teilsprachen, die bezüglich Syntax und Semantik differieren, unterscheiden.

Es sind:

- **C-Sprache**
- **Objektorientierte Erweiterungen von C**
- **Templates**
- **Standardbibliothek**
- **Präprozessorsprache**

Bemerkungen:

(i) **Das C++-Komitee versucht, die Fähigkeiten der Präprozessorsprache von C weitgehend durch C++-Konstrukte zu ersetzen.**

(ii) **Im Dokument N3242 sind der Kernsprache etwa 35,5% des Textes gewidmet und den Bibliotheken etwa 64,5%.**

```
// Ein einfaches Programm mit Ein- und Ausgabe,
// fehlerträchtige Version bez. der Eingabe.

#include<iostream>    // cin,  cout,  endl
using namespace std;

int  main() {
    int  a;
    int  b;

    // Lies Zahlen.
    cout << "Bitte zwei Ganzzahlen: ";
    cin >> a  >> b;

    // Berechne Summe.
    int s = a + b;
    // Gib Summe aus.
    cout << "Summe = " << s << endl;

   // Gib Produkt aus.
    cout << "Produkt = " << a * b << endl;
}//main


/* Zwei Läufe:

Zwei Ganzzahlen:   123  456
Summe = 579
Produkt = 56088

Zwei Ganzzahlen: 12,34
Summe = 4316887              //???
Produkt = 51802500           //???
*/
```

```
// Ein einfaches Programm mit Ein- und Ausgabe.
#include<iostream>    // cin,  cout,  endl
using namespace std;

int  main() {
   int  a;
   int  b;

   //  verbesserte Version,
   //  Abbruch bei fehlerhafter Eingabe
   std::cin.exceptions (std::ios_base::iostate (-1));

   // Lies Zahlen.
   cout << "Bitte zwei Ganzzahlen: ";
   cin >> a  >> b;

   // Berechne Summe.
   int s = a + b;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;
   // Gib Summe aus.
   cout << "Summe = " << s << endl;

  // Gib Produkt aus.
   cout << "Produkt = " << a * b << endl;
}//main

/* Lauf:  VS 2008
Zwei Ganzzahlen: 32 , 39

This application has requested the Runtime to terminate it
in an unusual way.  Please contact the application's
support team for more information.
*/
```

Logischer Datentyp

Werte: **true, false**

Operationen: **!, &&, ||, ==, !=, =**

Bemerkung: **Es finden automatische Typanpassungen zwischen booleschen Werten und Ganzzahlen statt.**

```
//  Beispiel zum booleschen Datentyp
#include <iostream>
using namespace std;

int  main () {
   bool  a  {true};
   bool  b  {};
   bool  c  {3 != 7};

   bool  d  =  21;
   //  warning C4305: 'Initialisierung': Verkürzung von
   // 'int'  in 'bool'
   int    i  =  a + b + 4*c;

   cout << "Ausgabe von Wahrheitswerten: "  << endl
        << "    a =  " << a << endl
        << "    b =  " << b << endl
        << "    c =  " << c << endl
        << "    d =  " << d << endl << endl;

cout << "Wert von  i =  " << i << endl << endl;
```

```
   cout << "Ausgabe von Wahrheitswerten (Textform): "
        << endl
        << boolalpha
        << "     a =  " << a << endl
        << "     b =  " << b << endl
        << "     c =  " << c << endl
        << "     d =  " << d << endl << endl;

   cout << "Ausdruck:  (true != false == true)  =  "
        <<  (true != false == true)  << endl << endl;

}//main



/* Ausgabe

Ausgabe von Wahrheitswerten:
    a =  1
    b =  0
    c =  1
    d =  1

Wert von  i = 5

Ausgabe von Wahrheitswerten (Textform):
    a =  true
    b =  false
    c =  true
    d =  true

Ausdruck:  (true != false == true)  =  true

*/
```

Ganzzahlen

C++ kennt zwei Arten von Ganzzahlen: vorzeichenbehaftete und vorzeichenlose Ganzzahlen.

vorzeichenbehaftete Ganzzahlen:

signed char,
short int,
int,
long int,
long long int.

vorzeichenlose Ganzzahlen:

unsigned char,
unsigned short int,
unsigned int,
unsigned long int,
unsigned long long int.

Bemerkungen:

(i) **Die Zahlbereiche für char, short, int, long und long long werden durch die Implementation festgelegt.**

(ii) **Die vorzeichenbafteten und vorzeichenlosen Varianten benötigen jeweils den gleichen Speicherplatz.**

(iii) **Für die Zahlbereiche gilt die folgende Inklusionskette: signed char $\subseteq$ short $\subseteq$ int $\subseteq$ long $\subseteq$ long long.**

(iv) **Die vorzeichenlosen Zahlen kennen nicht das Problem des Überlaufs.**

(v) **Für Ganzzahlen sind die üblichen Operationen definiert.**

```
//  Zahlbereiche
#include <iostream>
#include <limits>              //  Bereichsangaben
using namespace std;

int main() {
cout << "signed char umfasst die Zahlen " << endl
       << "von  "
      << (int) numeric_limits<signed char>::min()
      << endl
      << "bis   "
      << (int) numeric_limits<signed char>::max()
      << endl << endl;

cout << "short int umfasst die Zahlen " << endl
      << "von  " << numeric_limits<short>::min()
      << endl
      << "bis   "
      << numeric_limits<short>::max()
      << endl << endl;

cout << "int umfasst die Zahlen " << endl
      << "von  " << numeric_limits<int>::min()
      << endl
      << "bis   "
      << numeric_limits<int>::max()
      << endl << endl;

cout << "long int umfasst die Zahlen " << endl
      << "von  "
      << numeric_limits<signed long int>::min() << endl
      << "bis   "
      << numeric_limits<long>::max()
      << endl << endl;
```

```
cout << "long long int umfasst die Zahlen " << endl
     << "von  " << numeric_limits<long long>::min()
     << endl
     << "bis   "
     << numeric_limits<long long>::max()
     << endl << endl;

}//main


/*  Ausgabe unter VS2010

signed char umfasst die Zahlen
von   - 128
bis      127

short int umfasst die Zahlen
von   - 32768
bis      32767

int umfasst die Zahlen
von   - 2147483648
bis      2147483647

long int umfasst die Zahlen
von   - 2147483648
bis      2147483647

long long int umfasst die Zahlen
von  - 92233720368547758O8
bis     9223372036854775807
*/
```

```
//  Schreibweise von Ganzzahlen
//  in Programmtexten
//  Suffixe:  u, U, l, L, ll,  LL, ul, UL, ull, ULL
#include <iostream>
using namespace std;

int main () {

    int a = 123;
    int b = 0123;
    int c = 0x123;
    long la = 123456789l;
    long lb = 1234567892L;

    long long lla = 12345678901234ll;
    long long llb = 123456789012345 6LL;

    unsigned ua = 0xfabcdefau;
    unsigned ub = -1U;
    unsigned uc = 4294967395LL;     // keine Warnung !!!

    cout << "a  =   " << a << endl;
    cout << "b  =   " << b << endl;
    cout << "c  =   " << c << endl;
    cout << "la  = " << la << endl;
    cout << "lb  = " << lb << endl;
    cout << "lla  = " << lla << endl;
    cout << "llb  = " << llb << endl;

    cout << "ua  =  " << ua << endl;
    cout << "ub  =  " << ub << endl;
    cout << "uc  =  " << uc << endl;
}//main
```

/* Ausgabe

a = 123
b = 83
c = 291
la = 123456789
lb = 1234567892
lla = 12345678901234
llb = 1234567890123456
ua = 4206681850
ub = 4294967295
uc = 99 // Man beachte die
// Verkürzung von uc !!

***/**

```
//  Implizite Typanpassungen,  Integral promotion
#include <iostream>
#include <typeinfo>
using namespace std;

int  main () {
signed char sca = 102,  scb = 112,  scc = 122;
cout << "sca  =  "  <<  (int) sca  << "    ";
cout << "scb  =  "  <<  (int) scb  << "    ";
cout << "scc  =  "  <<  (int) scc  << endl;

cout << "sca + scb + scc = "  << sca + scb + scc << endl;
cout << "sca * scb * scc  = "  << sca * scb *scc << endl;
cout << "sca * scb * scc * sca * scb * scc = "
       << sca * scb *scc * sca * scb * scc << endl;
cout << "sca * scb * scc * sca * scb * scc * scc = "
       << sca * scb *scc * sca * scb * scc * scc << endl;
cout << "Typ des vorhergehenden Ausdrucks ist " <<
   typeid (sca*scb*scc*sca scb*scc*scc).name() << endl;

unsigned char uca = 102,  ucb = 112,  ucc = 122;
cout << "uca + ucb + ucc = "  << uca + ucb + ucc << endl;
cout << "uca * ucb * ucc  = "  << uca * ucb *ucc << endl;
cout << "uca * ucb * ucc * uca * ucb * ucc = "
       << uca * ucb *ucc * uca * ucb * ucc << endl;
cout << "uca * ucb * ucc * uca * ucb * ucc * ucc = "
       <<  uca * ucb * ucc * uca * ucb * ucc * ucc<< endl;
cout << "Typ des vorhergehenden Ausdrucks ist " <<
   typeid (uca*ucb*ucc*uca*ucb*ucc*ucc).name() << endl;

int ia  =  2000111222,  ib = 2111000333;
cout << "ia  =  "  << ia  << endl;
cout << "ib  =  "  << ib  <<  endl;
cout << "ia * ib  = "  <<  ia * ib  << endl;
```

```
long long lla  =  ia;
cout << "lla * ib  = "  <<  lla * ib  <<  endl;

unsigned ua = 2;
cout << "ua = "  <<  ua  <<  endl;
cout << "ua * -2  =  "  << ua * -2  << endl;
cout << "Typ von ua*lla ist "
       << typeid (ua*lla).name () << endl;
}//main


/*  Ausgabe:

sca  =  102    scb  =  112    scc  =  122
sca + scb + scc = 336
sca * scb * scc  = 1393728
sca * scb * scc * sca * scb * scc = 1152520192
sca * scb * scc * sca * scb * scc * scc = -1126457344
Typ des vorhergehenden Ausdrucks ist int
uca + ucb + ucc = 336
uca * ucb * ucc  = 1393728
uca * ucb * ucc * uca * ucb * ucc = 1152520192
uca * ucb * ucc * uca * ucb * ucc * ucc = -1126457344
Typ des vorhergehenden Ausdrucks ist int
ia  =  2000111222
ib  =  2111000333
ia * ib  = -842072578
lla * ib  = 42222354556790369 26
ua = 2
ua * -2  =  4294967292
Typ von ua*lla ist __int64
*/
```

Bitoperatoren

Man kennt die folgenden Bitoperatoren:

~	Bitweise Negation
<<	Linksschieben
>>	Rechtsschieben
&	Bitweises UND
^	Bitweises XOR
|	Bitweises ODER
<<=	Zuweisendes Linksschieben
>>=	Zuweisendes Rechtsschieben
&=	Zuweisendes bitweises UND
^=	Zuweisendes bitweises XOR
|=	Zuweisendes bitweises ODER

```
#include <iostream>
int main () {
   std:: cout << "Groß- und Kleinbuchstaben in der "
     "ASCII-Codierung haben einen Abstand von 32."
                << std::endl;

   for (int  i = 65;  i < 91; +  +i) {
      char ch = i;
      char chX32 = ch^32;
      std::cout << i << ":\t" << ch << "\t\t" << int(chX32)
              << ":\t" << chX32 << "\t\t" << int(chX32^32)
             << ":\t" << char(chX32^32) << std::endl;
   }
}//main
```

/*

Groß- und Kleinbuchstaben in der ASCII-Codierung haben einen Abstand von 32.

65:	A	97:	a	65:	A
66:	B	98:	b	66:	B
67:	C	99:	c	67:	C
68:	D	100:	d	68:	D
69:	E	101:	e	69:	E
70:	F	102:	f	70:	F
71:	G	103:	g	71:	G
72:	H	104:	h	72:	H
73:	I	105:	i	73:	I
74:	J	106:	j	74:	J
75:	K	107:	k	75:	K
76:	L	108:	l	76:	L
77:	M	109:	m	77:	M
78:	N	110:	n	78:	N
79:	O	111:	o	79:	O
80:	P	112:	p	80:	P
81:	Q	113:	q	81:	Q
82:	R	114:	r	82:	R
83:	S	115:	s	83:	S
84:	T	116:	t	84:	T
85:	U	117:	u	85:	U
86:	V	118:	v	86:	V
87:	W	119:	w	87:	W
88:	X	120:	x	88:	X
89:	Y	121:	y	89:	Y
90:	Z	122:	z	90:	Z

*/

Gleitpunktzahlen

C++ kennt drei Genauigkeitsstufen für Gleitpunktzahlen: float, double und long double. Die Genauigkeiten der einzelnen Gleitpunkttypen sind implementations-abhängig.

Bemerkung: Für maschinelle Gleitpunktzahlen gelten die algebraischen Gesetze der Arithmetik nur eingeschränkt. Im Header <limits> findet man einige implementationsabhängige Konstanten, um eine sinnvolle Fehlerabschätzung zu initiieren.

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

int main() {
    float a = 1.234567E-7F,
          b = 1.000000F,
          c = -b;
    float s1 = a + b;
    s1 += c;
    float s2 = a;
    s2 += b + c;
    cout << "Beispiel zur Nichtassoziativität der "
            "Gleitpunktoperationen" << endl << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
```

```
    cout << "c = " << c << endl;
    cout << "Summe (a+b)+c = " << s1 << endl;   // 1.192e-7
    cout << "Summe a+(b+c) = " << s2 << endl << endl;
                                          // 1.234e-7

    // Bestimmumg der Zahlengenauigkeit.
    //  Die Zahlengenauigkeit ist implementationsabhängig.
    cout << "Genauigkeit von float:         "
         << std::numeric_limits<float>::digits10 <<
                                  " Dezimalziffern"<< endl;
    cout << "Genauigkeit von double:       "
         << std::numeric_limits<double>::digits10 <<
                                   " Dezimalziffern"<< endl;
    cout << "Genauigkeit von long double:  " <<
       std::numeric_limits<long double>::digits10 <<
                                   " Dezimalziffern"<< endl;

}//main


/*  Ausgabe:  Rechnung auf Sparc unter Solaris
Beispiel zur Nichtassoziativität der Gleitpunktoperationen
a = 1.23457e-07
b = 1
c = -1
Summe (a+b)+c = 1.19209e-07
Summe a+(b+c) = 1.23457e-07

Genauigkeit von float:              6 Dezimalziffern
Genauigkeit von double:            15 Dezimalziffern
Genauigkeit von long double:  33 Dezimalziffern

*/
```

Zeichen

Aus dem Standard

The fundamental storage unit in the C++ memory model is the byte. A byte is at least large enough to contain any member of the basic execution character set and is composed of a contiguous sequence of bits, the number of which is implementation-defined. The least significant bit is called the low-order bit; the most significant bit is called the high-order bit. The memory available to a C++ program consists of one or more sequences of contiguous bytes. Every byte has a unique address.

Für das Schreiben von C++-Programmen gibt es den "basic source character set", er entspricht in etwa den druckbaren Zeichen des 7-Bit-ASCII-Codes und einigen Kontrollzeichen. Neben diesem Grundzeichensatz existieren: "basic execution character set", "basic execution wide-character set", "execution character set", "execution wide-character set". Sie sind alle implementations-definiert.

Die 91 graphischen Zeichen des basic source character set:

```
a b c d e f g h i j k l m n o p q r s t u
v w x y z
A B C D E F G H I J K L M N O P Q R S T U
V W X Y Z
0 1 2 3 4 5 6 7 8 9
_ { } [ ] # ( ) < > % : ; . ? * + - / ^ &
| ~ ! = , \ " '
```

Man kennt drei Zeichenarten, char, signed char, unsigned char. Die Mitglieder dieser drei Arten sind typverschieden. Weiterhin gibt es noch die Zeichenarten wchar_t, char16_t und char32_t.

Beispiele für Zeichen:

'a', '\141', '\x61', 'A', '\101', '\x41'

Sonderformen zur Zeichendarstellung:

newline	**\n**
horizontal tab	**\t**
vertical tab	**\v**
backspace	**\b**
carriage return	**\r**
formfeed	**\f**
alert	**\a**
backslash	****
question mark	**\?**
single quote	**\'**
double quote	**\"**

Daneben existieren für die Angabe eines Zeichens die Oktaldarstellung, die Sedezimaldarstellung und zwei universelle Darstellungen, \ooo, \xhhhh, \uhhhh und \Uhhhhhhhh.

```
//  Beispiel zur Zeichencodierung
#include <iostream>
using namespace std;

int main () {
   cout << "Ausgabe einiger Zeichen: "  <<  endl;
   cout << 'a' << ' ' << '\141' << ' ' << '\x61' << ' '
        << 'A' << ' ' << '\101' << ' ' << '\x41' << endl << endl;

   cout << "Ausgabe von Trigraphen:" << endl;

   cout << "\?\?=   =  ??=   \?\?(  =   ??(\n"
           "\?\?<   =  ??<   \?\?)  =   ??)\n"
           "\?\?>   =  ??>   \?\?'  =   ??'\n"
           "\?\?!   =  ??!   \?\?-  =   ??-\n"
           "\?\?/'  =  ??/'   "  << endl << endl;

}//main



/* Ausgabe:

Ausgabe einiger Zeichen:
a a a A A A

Ausgabe von Trigraphen:
??=   =  #    ??(  =   [
??<   =  {    ??)  =   ]
??>   =  }    ??'  =   ^
??!   =  |    ??-  =   ~
??/'  =  '

*/
```

```
//  Zweites Beispiel zur Zeichencodierung
#include <iostream>
using namespace std;

int main () {
  cout << "Benutzung universeller Bezeichnungen:"
          << endl;

  cout  << "\u015A  \u015B  \u015C  \u015D" << endl
          << endl;
}//main


/*
Ausgabe:

Benutzung universeller Bezeichnungen:
Ś ś Ŝ ŝ            //  Ausgabe unter g++ unter Ubuntu
? ? ? ?            //  Ausgabe unter Windows codepage 1252

*/
```

Bezeichner:

identifier:
identifier-nondigit
identifier identifier-nondigit
identifier digit

identifier-nondigit:
nondigit
universal-character-name
other implementation defined character

nondigit: one of
a b c d e f g h i j k l m
n o p q r s t u v w x y z
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z _

digit: one of
0 1 2 3 4 5 6 7 8 9

Bemerkungen:

(i) Ein Bezeichner beginnt mit einem Unterstrich oder einem Buchstaben, gefolgt von Buchstaben, Ziffern oder Unterstrich.

(ii) Welche Unicode-Zeichen als Buchstaben gelten, ist im Anhang E des Standards geregelt.

(iii) Manche Bezeichner sind für die Sprache C++ und ihre Bibliotheken reserviert. Dazu gehören die Schlüsselwörter und Bezeichner, die zwei aufeinanderfolgende Unterstriche enthalten.

(iv) Bezeichner unterliegen keiner Längenbeschränkung.

```
//  Umlaute in Bezeichnern
//  Beispiel unter Windows
#include <iostream>
using namespace std;

int main () {
   int größe = 10;
   int grün  = 20;
   cout << "grün + größe = " << grün + größe << endl;
}

/* Ausgabe:
grün + größe = 30
*/


//  Obiges Programm unter g++ von Ubuntu
//  Nicht-ASCII-Zeichen in Bezeichnern
//  Universal Character Names
//  Uebersetzt mit -fextended-identifiers
#include <iostream>
using namespace std;

int main () {
   int gr\u00F6\u00DFe = 10;
   int gr\u00FCn = 20;
   cout << "gr\u00FCn + gr\u00F6\u00DFe = "
        << gr\u00FCn + gr\u00F6\u00DFe  << endl;
}

/*  Ausgabe:
grün + größe = 30
*/
```

```
//  Ein Programm zur Demonstration von wide-character


#include <cwchar>
#include <cwctype>
#include <cstdio>


wchar_t*  tolower (wchar_t* str) {
   wchar_t*  start  =  str;
   // wandeln in Kleinschreibung
   for ( ;  *str;  str++) {
      if  (iswupper (*str))
      *str  =  towlower (*str);
   }
return start;
}


int main() {

   printf ("sizeof (wchar_t) = %d\n", sizeof (wchar_t));

   wchar_t  wca = L'a';
   wchar_t  wcb = L'\377';
   wchar_t  wcc = L'\x4567';
   printf ("wca = %x\nwcb = %x\nwcc = %x\n",
               wca,  wcb,  wcc);

   wprintf (L"%lc\n", wca);
   wprintf (L"%lc\n", wcb);
   wprintf (L"%lc\n", wcc);
```

```
    wchar_t str [] = L"TEST\300\301\302normal";

    wchar_t  buf [14];
    wcscpy  (buf,  str);
    (void) tolower (buf);
    //  teilweise werden vertraute Glyphen ausgegeben
    wprintf (L"%ls\n", buf);
    for (int i = 0; i < 13; ++i)
        wprintf (L"%lc ", buf [i]);
    printf ("\n");
}//main




/* Ausgabe
// Lauf unter VS2008

sizeof (wchar_t) = 2
wca = 61
wcb = ff
wcc = 4567
a
ÿ
?                              //  unbekanntes Zeichen
testÀÁÂnormal
t e s t À Á Â n o r m a l

*/
```

Zeiger

Zeiger zeigen auf Objekte oder Funktionen.

```
//  Nutzung von Zeigern
#include <iostream>
using namespace std;
int main ()  {
   int   wert = 1024;
   int*  pi = &wert;      // pi zeigt auf Ganzzahl
   int** ppi = &pi;       // ppi zeigt auf einen Zeiger, der auf
                          // eine Ganzzahl zeigt
   cout << "Zugriff auf Wert:" << endl
        << "direkt:          " << wert << endl
        << "indirekt:        " << *pi << endl
        << "zweifach indirekt: " << **ppi << endl;
   int  i = 4;
   int  j = 10;
   int* p1 = &i;
   int* p2 = &j;
   *p2 = *p1 * *p2;
   *p1 *= *p1;
   cout << "j  = " << j << endl;
   cout << "i  = " << i << endl;
}//main

/*  Ausgabe:
Zugriff auf Wert:
direkt:                1024
indirekt:              1024
zweifach indirekt:     1024
j  = 40
i  = 16
*/
```

```
//  Zur Zeigerarithmetik
//  Zeiger +/- Ganzzahl,  Zeiger - Zeiger
#include <iostream>
using namespace std;

int main () {
    const int arr_sz = 5;
    int arr [arr_sz] = {0, 1, 2, 3, 4};
    // pbegin zeigt auf erstes Element,
    // pend auf Element unmittelbar hinter dem letzten
    for (int* pbegin = arr, *pend = arr + arr_sz;
          pbegin != pend;   ++pbegin)
          cout << *pbegin << ' ';
    cout << endl;
    cout << "Nochmalige Ausgabe des Arrays arr:"
          << endl;
    int* barr = &arr [0];
    cout << *(arr + 0) << ' '
        << *(1 + arr) << ' '
        << 2 [arr] << ' '
        << *(barr + 3) << ' '
        << *(4 + barr) << endl;
    cout << endl;
}


/* Ausgabe:

0 1 2 3 4
Nochmalige Ausgabe des Arrays arr:
0 1 2 3 4

*/
```

```
//  Arrays

/*  Beispiele eindimensionaler Arrays  */

#include <iostream>
#include <iomanip>
#include <typeinfo>
using namespace std;


int main () {
      //  Definition eines neuen Arrays
      int   arr01 [] = {-21, -43, -65, 76, 87, 91};
      cout << "Ausgabe des Arrays arr01:"
            << endl;
      for  (int i = 0;  i < sizeof arr01 / sizeof arr01 [0];  ++i) {
            cout << "   Element "  <<  i  << ":"
                  << setw (6)  << arr01 [i] << endl;
      }
   cout << endl;
   cout << "Typ von arr01 = " << typeid (arr01).name()
        << endl;
   cout << "Ausgabe von arr01: " << arr01 << endl;
   cout << "Typ von &arr01 [0]  = " <<
               typeid (&arr01[0]).name() << endl;

     //  Dynamische Erzeugung eines Arrays
      int*  ap01 = new int [6];
      //  Initialisierung
      for (int i = 0;   i < 6;  ++i)
            ap01 [i] = 10*i + i;
      //  Ausgabe
      cout << endl
```

```
            << "Erste Ausgabe des ap01-Arrays:"
            << endl;
       for  (int i = 0;   i < 6;  ++i)
            cout << setw (6)  << ap01 [i];
       cout << endl;
       cout << "Zweite Ausgabe des ap01-Arrays:"
            << endl;
       for  (int i = 0;   i < 6;  ++i)
            cout << setw (6) << *(ap01 + i);
       cout << endl;
   //  Speicher freigeben
   delete [] ap01;
}//main


/*  Ausgabe

Ausgabe des Arrays arr01:
  Element 0:   -21
  Element 1:   -43
  Element 2:   -65
  Element 3:    76
  Element 4:    87
  Element 5:    91

Typ von arr01 = int [6]
Ausgabe von arr01: 006BFDE0
Typ von &arr01 [0]  = int *

Erste Ausgabe des ap01-Arrays:
    0    11    22    33    44    55
Zweite Ausgabe des ap01-Arrays:
    0    11    22    33    44    55
*/
```

```
/*  Array als Parameter*/

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


int  arr01 [6] = {-21, -43, -65, 76, 87, 91};
int  arr02 [3] = {32, 42};

void f (int a []) {
   cout << "Typ von " << a << " = "
        << typeid(a).name() << endl;
}
#define g(par)  cout << "Typ von " << #par << " = " \
                << typeid(par).name() << endl

int main () {
      //  Aufruf von f
      f (arr01);
      f (arr02);

      //Makro g
      g (arr01);
      g (arr02);
      // Sonderbehandlung von char []
      char a []  = "Erste Zeichenkette";
      char* b    = "Zweite Zeichenkette";

      cout << a << endl;
      cout << b << endl;
}//main
```

/* Ausgabe:
Typ von 0021D000 = int *
Typ von 0021D018 = int *
Typ von arr01 = int [6]
Typ von arr02 = int [3]
Erste Zeichenkette
Zweite Zeichenkette
***/**

```
/*  Array: Typanpassung  */

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

int  arr01 [6] = {-21, -43, -65, 76, 87, 91};
int  arr02 [3] = {32, 42};

void f (int a []) { }
//  void f (int *) { }  Fehler: Neudefinition von f

typedef  int * ta;
typedef  int  tb [];

int main () {
        cout << typeid (ta).name() << endl;
        cout << typeid (tb).name() << endl;
        if (typeid (ta)  !=  typeid (tb))
            cout << typeid (ta).name() << " und " <<
             typeid (tb).name() << " sind verschieden." << endl;
        cout << arr01 << endl;
        cout << arr02 << endl;
}//main


/*  Ausgabe
int *
int [0]
int * und int [0] sind verschieden.
00425004
0042501C
*/
```

```
/* Typanpassung bei Templates  */

#include <iostream>
using namespace std;

int  arr01 [6] = {-21, -43, -65, 76, 87, 91};
int  arr02 [3] = {32, 42, 52};
int  arr03 []  = {101, 102, 103, 104};

template <typename T>
int f (T a, T b) {
   return  a [1] + b [1];
}


int main () {
       int x = f (arr01,  arr02);
       cout << "x = " << x  << endl;

       int y = f (arr01,  arr03);
       cout << "y = " << y  << endl;

       int z = f (arr03,  arr02);
       cout << "z = " << z  << endl;

}//main

/*  Ausgabe:
   x = -1
   y = 59
   z = 144
*/
```

```
/*  Zweidimensionaler Array  */

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


int main () {

      double ar2 [3] [5];

      //  Initialisierung
      for (int i = 0;  i < 3;  ++i)
      for (int j = 0;  j < 5;  ++j)
            ar2 [i] [j] = 10.7*i + j;

      cout << "Adresse von ar2         = " << ar2 << endl;
      cout << "Adresse von ar2 [0]     = " << ar2 [0]
            << endl;
      cout << "Adresse von ar2 [0] [0] = " << & ar2 [0] [0]
            << endl;
      cout << "Adresse von ar2 [1]     = " << ar2 [1]
           << endl;
      cout << "Wert an ar2 [1] [0]     = " << ar2 [1] [0]
           << endl;
      cout << "Adresse von ar2 [2]     = " << ar2 [2]
           << endl;
      cout << "Adressdifferenz zu ar2 [1] = "
                                 << (ar2 [2] - ar2 [1]) << endl;
      cout << "Differenz in Byte          = "
            << setw (2)
            << ((char*) ar2 [2] - (char*) ar2 [1])  << endl;
      cout << "Wert an ar2 [2] [0]     = "
            << ar2 [2] [0] << endl << endl;
```

```
//  Nun das gleiche fuer dynamisch bereitgestellte
//  Arrays.


double*   datap = new double [3 * 5];
double**  dr2   = new double* [3];
for  (int  i0 = 0;  i0 < 3;  ++i0)
       dr2 [i0] = datap + i0 * 5;
for  (int i1 = 0;  i1 < 3;  ++i1)
for  (int j1 = 0;  j1 < 5;  ++j1)
       datap [i1*5 + j1] = 2.7*(i1+1) + j1;

//  Ausgabe des Arrays
cout << "Array dr2: "
        << endl;
for  (int i2 = 0;  i2 < 3;  ++i2)
for  (int j2 = 0;  j2 < 5;  ++j2)
       cout << datap [i2*5 + j2] << "   ";
cout << endl
        << endl;

cout << "Wert von datap          = " << datap << endl;
cout << "Wert von *datap         = " << *datap
       << endl;
cout << "Wert von datap [0]      = " << datap [0]
       << endl;
cout << "Wert von dr2            = " << dr2 << endl;
cout << "Inhalt von *dr2         = " << *dr2 << endl;
cout << "Inhalt von dr2 [0]      = " << dr2 [0] << endl;
cout << "Adresse von dr2 [0] [0]  = " << & dr2 [0] [0]
       << endl;
cout << "Inhalt von dr2 [1]      = " << dr2 [1] << endl;
```

```
    cout << "Wert an *dr2 [1]        = " << dr2 [1] [0]
         << endl;
    cout << "Inhalt von dr2 [2]      = " << dr2 [2] << endl;
    cout << "Differenz zu dr2 [1]    = "
         << (dr2 [2] - dr2 [1]) << endl;
    cout << "Differenz in Byte       = "
          << ((char*) dr2 [2] - (char*) dr2 [1])  << endl;
    cout << "Wert an dr2 [2] [0]     = " <<  dr2 [2] [0]
         << endl;
    cout << "Ausdruck des Arrays:" << endl;

    for  (int i3 = 0;  i3 < 3;  ++i3)
    for  (int j3 = 0;  j3 < 5;  ++j3)
         cout << dr2 [i3] [j3] << "   ";
    cout << endl;

    //  Rueckgabe des angeforderten Speichers
    delete [] dr2;
    delete [] datap;
}//main


/*   Ausgabe

Adresse von ar2              = 0012FF08
Adresse von ar2 [0]          = 0012FF08
Adresse von ar2 [0] [0]      = 0012FF08
Adresse von ar2 [1]          = 0012FF30
Wert an ar2 [1] [0]          = 10.7
Adresse von ar2 [2]          = 0012FF58
Adreßdifferenz zu ar2 [1]    = 5
Differenz in Byte            = 40
Wert an ar2 [2] [0]           = 21.4
```

Array dr2:
2.7 3.7 4.7 5.7 6.7
5.4 6.4 7.4 8.4 9.4
8.1 9.1 10.1 11.1 12.1

Wert von datap = 00481A60
Wert von *datap = 2.7
Wert von datap [0] = 2.7
Wert von dr2 = 00481CA0
Inhalt von *dr2 = 00481A60
Inhalt von dr2 [0] = 00481A60
Adresse von dr2 [0] [0] = 00481A60
Inhalt von dr2 [1] = 00481A88
Wert an *dr2 [1] = 5.4
Inhalt von dr2 [2] = 00481AB0
Differenz zu dr2 [1] = 5
Differenz in Byte = 40
Wert an dr2 [2] [0] = 8.1
Ausdruck des Arrays:
2.7 3.7 4.7 5.7 6.7
5.4 6.4 7.4 8.4 9.4
8.1 9.1 10.1 11.1 12.1

***/**

C++ Container array

Beispiel: array <int, 7> a = {17, 56, 25, 4}

Der C++-array stellt einen Behälter zur Aufnahme von typgleichen Elementen dar. Die Anzahl der Elemente wird zur Übersetzungszeit festgelegt und ist später nicht zu ändern. Der Behälter verhält sich platzsparend, außer den Elementen wird im Prinzip nichts gespeichert, auch nicht eine Größenangabe. Auf die einzelnen Elemente kann man über einen Index zugreifen, wobei der Index innerhalb der zulässigen Grenzen frei wählbar ist. Für den obigen array a sind dies die Grenzen 0 und 6. So speichert der array a die Ganzzahlen 17, 56, 25, 4, 0, 0, 0. Als Container verfügt der array über eine Reihe erwarteter Mitglieder, z. B. operator [], at, front, back, data, begin, end, size.

Allgemeine Form:
array <Typ, Ausdruck> bezeichner =
{Initialisierungsliste},
hierbei muß Ausdruck eine Konstante zur Übersetzungszeit liefern.

```
#include <array>
#include <iostream>

int main () {
   std::array<int, 6> a6  =  {1, 2, 3, 4, 5, 6};
   for (auto i : a6)
       std::cout << i << ' ';
   std::cout << std::endl;
}
/*1 2 3 4 5 6*/
```

```
// Zweites Beispiel
#include <iostream>
#include <algorithm>
#include <array>

int main () {

   // Erstellen mit Initialisierungsliste
   std::array<int, 9> a9 {1,-2, 3, -4, 5, -6, 7, -8};

   std::cout << "Ausgabe des Originalarrays:\n";

   for (auto j : a9)
        std::cout << j << ' ';
   std::cout << std::endl;

   // Sortierung des Inhalts ist Container-Operation.

   std::sort (a9.begin(), a9.end());

   std::cout << "Ausgabe nach Sortierung:\n";

  for (int j : a9)
        std::cout << j << ' ';
   std::cout << std::endl;
}//main

/*
Ausgabe des Originalarrays:
1 -2 3 -4 5 -6 7 -8 0
Ausgabe nach Sortierung:
-8 -6 -4 -2 0 1 3 5 7
*/
```

```
//  Beispiel zu zweidimensionalem Array

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

int main () {
    array<array<double, 3>,  5> d2 =
                              {17.1, 22.3, 79.7, 88.2, 11.1, 12.7,
                               31.6, 25.4, 91.7, 57.2, 18.6, 42.6,
                               11.1, 44.4, 78.9};

   cout << "Ausgabe mit expliziter Indexpruefung:"
        << endl;
   for (int i = 0;  i < 5;  ++i) {
      for (int j = 0;  j < 3;  ++j)
         cout << d2.at(i).at (j) << ' ';
      cout << endl;
   };

   cout << endl
        << "Ausgabe mittels Bereichsschleife:"  << endl;
   for (auto i : d2) {
      for (auto j : i)
         cout << j << ' ';
      cout << endl;
  }

  cout << endl << "Speicherplatz fuer d2:  "
       << sizeof (d2) << “Byte” << endl;

}//main
```

/*

Ausgabe mit expliziter Indexpruefung:
17.1 22.3 79.7
88.2 11.1 12.7
31.6 25.4 91.7
57.2 18.6 42.6
11.1 44.4 78.9

Ausgabe mittels Bereichsschleife:
17.1 22.3 79.7
88.2 11.1 12.7
31.6 25.4 91.7
57.2 18.6 42.6
11.1 44.4 78.9

Speicherplatz fuer d2: 120 Byte

***/**

```
//  Beispiel zu Referenzen
#include <iostream>
using namespace std;

int main () {
   int  i;
   int&  ri = i;     //  Referenzen müssen initialisiert werden!
   int&  rri = ri;
   i = 5;    ri = 10;    rri = 25;
   cout << "i = " << i << ",    ri = " << ri
        << ",   rri = " << rri << endl;

   long long a [] = {1234, 5678, 9012, 3456, 7654};
   long long& ra0 = a [0];
   long long& ra3 = a [3];

   cout << "a [0] = " << ra0 << endl;
   cout << "a [3] + 10 = " << ra3 + 10 << endl;
}// main

/*  Ausgabe:
i = 25,    ri = 25,   rri = 25
a [0] = 1234
a [3] + 10 = 3466
*/
```

Bemerkung: In C++98 gilt folgendes. Eine Referenz stellt einen Alias für ein Objekt dar. Es existieren keine Referenzen auf Referenzen, keine Arrays von Referenzen und keine Zeiger auf Referenzen. Es ist unspezifiziert, ob eine Referenz Speicherplatz benötigt.

Bemerkung: In C++11 wird der Begriff der Referenz erweitert, man kennt nun „Rvalue References“ für Typen T, dargestellt durch T&&. Der Grund hierfür liegt in dem Wunsch einer verbesserten Code-erzeugung in Sonderfällen.

Von Beginn an unterscheidet man in C bei einer Zuweisung x = 14 "lvalue" und "rvalue" für Links- und Rechtswert. In erster Näherung kann man sagen, ein lvalue beschreibt ein Stückchen Speicher und hat daher eine Adresse, ein rvalue beschreibt einen adreßlosen Wert.

```
//  Beispiel zu Rvalue Reference
#include <iostream>
using namespace std;

void fun (int& x) {
   cout << "In fun (&),  x = " << x << endl;}

void fun (int&& x) {
   cout << "In fun (&&),  x = " << x << endl;}

int main () {
  int z = 370;
  fun (z);
  fun (370);
}

/*
In fun (&),  x = 370
In fun (&&),  x = 370
*/
```

```
//  Weiteres Beispiel zu Referenzen
#include <cstdlib>
#include <iostream>
using namespace std;

int a [20];

void f (int& x) {
        const int z = 10;
         x += z;
}

int& g (int i) {
        return a [i];
}


int main () {
        int i = 10;
        f (i);  f (i);
        cout << "i = " << i <<  endl;
        g (4) = 40;       g (7) = 77;
        cout << "Ausgabe von a:" << endl;
        for (int i = 0;  i < 10;  i++)
                cout << a [i] << "   ";
        cout << endl;
}//main

/*  Ausgabe
i = 30
Ausgabe von a:
0  0  0  0  40  0  0  77  0  0
*/
```

Aufzählungen

Jede Enumeration führt einen neuen Typ ein. Die Enumerationskonstanten erhalten Integralwerte. Eine implizite Typanpassung erfolgt höchstens nur in Richtung zu Integralwerten. Eine Deklaration für Aufzählungen hat etwa folgende Form:

enum [class|struct] [Bezeichner] [:Trägertyp]{Enum-Liste}

Bemerkungen:

(i) **Man unterscheidet zwischen „scoped“ und „unscoped“ Aufzählungen, bei „scoped“ benutzt man das Schlüsselwort class oder struct. Beide Schlüsselwörter sind semantisch identisch.**

(ii) **Die Bezeichner in der Enum-Liste sind in der „scoped“ Form nur über den Bereichsoperator zugänglich, in der „unscoped“ Form werden sie automatisch in den direkt umschließenden Bereich exportiert.**

(iii) **Der Trägertyp muß ein integraler Typ sein, bei Nichtfestlegung ist es int.**

Beispiel aus Standard:

```
enum class color {red, yellow, green};

int x = color::red;          // Fehler, keine Typanpassung
color y = color::red;
if (y) {}                    // Fehler, keine Wandlung
                             // nach bool.
```

```
//  Programm zu Aufzählungen

#include <iostream>
using namespace std;

enum color  {red,  yellow,  green = 12,  blue,  violet = -6};

int main() {
   // enum ohne Bezeichner
   enum  {klein = -999,  gross = 999};
   cout  <<  "klein  =  "  <<  klein  << endl
         <<  "gross  =  "  <<  gross  << endl;

   cout << red << ' ' << yellow << ' ' << green << ' '
        << blue << ' ' << violet << endl;
   int i  =  green + violet;
   cout << "i = " << i << endl;
   // red  =  18;                      //  Fehler
   //  color violet = 10;              //  Fehler
   color violet  =  color (10);        //  Neue Variable violet
   cout << "violet = " << violet << endl;
   cout << "::violet = " << ::violet << endl;
   cout << "color::green = " << color::green << endl;

   //  "scoped"  enum
   enum class direction : char {left = 'l', right = 'r'};
   cout << "left = " << (char) direction::left << endl
        << "right = " << (char) direction::right  << endl;
   //  direction da = 'z';                  //  Fehler
   //  direction db = direction {120};    //  Fehler
   direction dc = static_cast<direction> (70);
   cout << "dc = " << (char) dc << endl;
}// main
```

/* Ausgabe:

klein = -999
gross = 999
0 1 12 13 -6
i = 6
violet = 10
::violet = -6
color::green = 1
left = l
right = r
dc = F

***/**

```
//  Beispiel zu komplexen Zahlen
#include <iostream>
#include <complex>          // Definition komplexer Zahlen
#include <cmath>
using namespace std;

// Beispiele für komplexe Zahlen
int main () {

   complex<double> a (5.4, 7.6);   // komplexe Zahl 5.4+7.6i
   complex<double> b (1.2, 3.4);   //                1.2+3.4i
   cout << "Komplexe Zahlen \na und b = "
        << a << "   "<< b << endl;

   cout << "a + b = " << a + b << endl;
   cout << "a - b = " << a - b << endl;
   cout << "a * b = " << a * b << endl;
   cout << "a / b = " << a / b << endl;
   complex<double> c = sqrt (a);
   complex<double> d = c*c;
   cout << "Quadratwurzel (" << a << ") = " << c << endl;
   cout << "Pruefung: c * c = a  " << c * c << endl;

   cout << "Komplexe Zahl,  zweiteilig =   ";
   double re = d.real();               // Realteil
   cout << re << " + ";
   cout << d.imag() << 'i' << endl;     // Imaginärteil

   complex<double> f {1.0, 2.0};                  // 1.0 + 2.0i
   d = conj (f);                        // Konjugierte: 1.0 - 2.0i

   // Bildung aus Polarkoordinaten
   const double pi = 4.0 * atan (1.0);
   double betrag = 10.0;
```

```
    double phase  = pi/4.0;
    complex<double> g = polar (betrag, phase);
    // Umrechnung in Polarkoordinaten
    double rho   = abs (g);   // Betrag
    double theta = arg (g);   // Phase
    double no = norm (g);     // Betrag^2

    cout << endl;
    cout << "Betrag  = " << betrag << endl;
    cout << "rho     = " << rho    << endl;
    cout << "Norm    = " << no     << endl;
    cout << "Phase   = " << phase  << endl;
    cout << "theta   = " << theta
         << " = " << theta/pi*180. << " Grad" << endl;
    cout << "Komplexe Zahl eingeben. \n"
            "Erlaubte Formate (Beispiel):"
            "\n (1.78, -98.2)\n (1.78)\n 1.78\n:";
    cin >> f;
    cout << "Komplexe Zahl = " << f << endl;

    complex<double> i = -1;
    i = sqrt (i);
    cout << "i = " << i << endl;

}//main
```

/* Ausgabe:

Komplexe Zahlen
a und b = (5.4,7.6) (1.2,3.4)
a + b = (6.6,11)
a - b = (4.2,4.2)
a * b = (-19.36,27.48)
a / b = (2.48615,-0.710769)
Quadratwurzel ((5.4,7.6)) = (2.71322,1.40055)
Pruefung: c * c = a (5.4,7.6)

Komplexe Zahl, zweiteilig = 5.4 + 7.6i

Betrag = 10
rho = 10
Norm = 100
Phase = 0.785398
theta = 0.785398 = 45 Grad

Komplexe Zahl eingeben.
Erlaubte Formate (Beispiel):
(1.78, -98.2)
(1.78)
1.78
:(0, 3.14)
Komplexe Zahl = (0,3.14)

i = (0,1)

***/**