Beispiel: Funktion für Matrixvektormultiplikation (STL-Vektoren)

```
#include <iostream>
#include <iomanip>
#include <vector>

using namespace std;

vector<double> matvekmult(vector<vector<double>> a, vector<double> x)
// Liefert A*x als Funktionswert
{
  int m=a.size(),n=x.size(); // keine Ueberpruefung auf Rechtecksgestalt
  vector<double> b(m);

  for (int i=0; i<m; ++i) {
    b[i] = 0;
    for (int j=0; j<n; ++j) b[i] += a[i][j]*x[j];
  }
  return b;
}


int main()
{
  int m,n;
  cout << "m n: "; cin >> m >> n;

  vector<double> x(n),b;

  // Matixvereinbarung
  vector<vector<double>> a(m,vector<double>(n));

  // Matrixeingabe (a)
  cout << endl << "a:" << endl;
  for (int i=0; i<m; ++i)
    for (int j=0; j<n; ++j)
      cin >> a[i][j];

  // Vektoreingabe (x)
  cout << endl << "x:" << endl;
  for (int j=0; j<n; ++j) cin >> x[j];

  // Matrixvektormultiplikation (b=a*x)
  b = matvekmult(a,x);

  // Vektorausgabe (b)
  cout << endl << "b:" << endl;
  for (int i=0; i<m; ++i)
    cout << fixed << setprecision(2) << setw(6) << b[i] << endl;
  return 0;
}
```

Zeichen

Datentypen für Zeichen

`char`, `unsigned char`, `signed char` (Größe 1 Byte=`CHAR_BIT` Bits, definiert in `climits`)

Sie zählen (wie `bool`) zu den ganzzahligen Datentypen und unterliegen in arithmetischen Ausdrücken der Integererweiterung. Die Regeln für implizite Typumwandlungen gelten analog.

Es ist implementierungsabhängig, ob `char` dieselbe Darstellung wie `signed char` oder `unsigned char` besitzt. (g++-7.5, Ubuntu Linux 18.04: `char`$\hat{=}$`signed char`).

Bsp. für Zeichenliterale: `'A'`, `'5'`, `'*'`, `'\n'`, `' '`, `'\0'`, `'\40'`, `'\x20'`, `'\''`, `'\\'`.

Der arithmetische Wert eines Zeichens ist positiv für die druckbaren Zeichen des Minimalzeichensatzes.

Ersatzdarstellung für einzelne Zeichen

Zeichen	*Bedeutung*
`\a`	akustisches Signal
`\b`	Rückwärtspositionierung um 1 Zeichen
`\f`	Seitenvorschub
`\n`	Zeilenvorschub
`\r`	Wagenrücklauf
`\t`	horizontaler Tabulator
`\v`	vertikaler Tabulator
`\\`	rückwärts geneigter Schrägstrich
`\?`	Fragezeichen
`\'`	Apostrophzeichen
`\"`	Doppelapostrophzeichen
`\`*ooo*	Oktaldarstellung (ein bis drei Oktalziffern)
`\x`*hh*	Hexadezimaldarstellung (ein oder zwei Hexadezimalziffern)

Testen und Umwandeln von Zeichen (`cctype`)

Funktion	*Bedeutung*
`int islower(int c)`	Kleinbuchstabe
`int isupper(int c)`	Großbuchstabe
`int isalpha(int c)`	Buchstabe
`int isdigit(int c)`	Dezimalziffer
`int isxdigit(int c)`	Hexadezimalziffer
`int isalnum(int c)`	Buchstabe oder Ziffer
`int isgraph(int c)`	druckbar ohne Leerzeichen
`int isprint(int c)`	druckbar einschließlich Leerzeichen
`int ispunct(int c)`	druckbar mit Ausnahme von Leerzeichen, Buchstaben und Ziffern
`int isspace(int c)`	Zwischenraumzeichen
`int iscntrl(int c)`	Steuerzeichen
`int tolower(int c)`	Umwandlung in Kleinbuchstaben
`int toupper(int c)`	Umwandlung in Großbuchstaben

c muss eine ganze Zahl vom Datentyp `int` aus dem Wertebereich von `unsigned char` und `EOF` sein. Ebenso liefern `toupper` und `tolower` ganze Zahlen aus diesem Wertebereich.

C-Zeichenketten

C-Zeichenketten sind C-Vektoren aus `char`, die mit `'\0'` enden.

C-Zeichenkettenkonstante: `"`$z_0 \ldots z_{k-1}$`"` $\hat{=}$ `{'`z_0`',...,'`z_{k-1}`','\0'}`

Bsp.: Initialisierung von C-Zeichenkettenvariablen

```
char name[5] = "text";  /* {'t','e','x','t','\0'} */
char name[4] = "text";  /* {'t','e','x','t'}  keine C-Zeichenkette! */
char name[]  = "text";  /* {'t','e','x','t','\0'} */
char name[6] = "text";  /* {'t','e','x','t','\0','\0'} */
char name[]  = "";      /* {'\0'} */
```

Funktion (<cstring>)	*Bedeutung*
`size_t strlen(const char s[])`	Länge der Zeichenkette s (ohne `'\0'`)

STL-Zeichenketten in C++ (`string`)

Im Unterschied zu C-Zeichenketten werden C++-Zeichenketten nicht mit `\0` abgeschlossen, sondern es wird die aktuelle Länge mit abgespeichert.
Im folgenden sind einige wichtige Operatoren und Funktionen für den Datentyp `string` aufgeführt. Dabei stehen s und t für C++-Zeichenketten, ct für eine `\0`-terminierte C-Zeichenkette, c für ein Zeichen vom Typ `char` und i und n für ganze Zahlen vom Typ `string::size_type` (g++-7.5 unter Ubuntu Linux 18.04 amd64: `unsigned long`).

Operation	*Wirkung*
`string s`	vereinbart leere Zeichenkette
`string s(t)`	vereinbart Zeichenkette, kopiert Zeichenkette t auf s
`string s(t,i)`	vereinbart Zeichenkette, kopiert t auf s ab Index i
`string s(t,i,n)`	wie `string s(t,i)`, höchstens n Zeichen werden kopiert
`string s(ct)`	vereinbart Zeichenkette, kopiert C-Zeichenkette ct auf s
`string s(n,c)`	vereinbart Zeichenkette und initialisiert sie mit n Zeichen c
`s[i]`	Komponentenwert zum Index i
`s.at(i)`	Komponentenwert zum Index i mit Bereichsüberwachung
`s=t s=ct s=c`	weist Zeichenkette, C-Zeichenkette bzw. Zeichen zu
`s+t s+ct ct+s`	liefert Verkettung zweier Zeichenketten als neue Zeichenkette
`s+=t s+=ct s+=c`	hängt Zeichenkette, C-Zeichenkette bzw. Zeichen an s an
`s==t s!=t s>t s<t s>=t s<=t`	vergleicht s und t lexikographisch
`s==ct s!=ct s>ct s<ct s>=ct s<=ct`	vergleicht s und ct lexikographisch
`cs==t cs!=t cs>t cs<t cs>=t cs<=t`	vergleicht cs und t lexikographisch
`s.size() s.length()`	liefert Zeichenzahl (als `string::size_type`)
`s.max_size()`	liefert Maximalzahl von Zeichen in s
`s.substr(i) s.substr(i,n)`	liefert Teilzeichenkette ab Index i bzw. mit max. Länge n als neue Zeichenkette (falls $i \leq s.size()$)
`s.c_str()`	liefert s als konstante C-Zeichenkette (mit angehängtem `\0`)
`s.data()`	liefert s als konstanten C-Vektor aus `char` (ohne angehängtes `\0`)
`s.find(t) s.find(ct) s.find(c)`	sucht erstes t,ct,c in s, liefert Index oder `string::npos`
`s.find(t,i) s.find(ct,i)`	sucht erstes t,ct,c in s ab Index i, liefert Index
`s.find(c,i)`	oder `string::npos`

`s.rfind(t) s.rfind(ct) s.rfind(c)`	wie `s.find`, jedoch Suche nach letztem Auftreten
`s.rfind(t,i) s.rfind(ct,i)`	”
`s.rfind(c,i)`	”
`s.insert(i,t) s.insert(i,ct)`	fügt t bzw. ct in s ab Index i ein (falls $i \leq s.size()$)
`s.erase(i) s.erase(i,n)`	entfernt alle bzw. max. n Zeichen von s ab Index i (falls $i \leq s.size()$)
`s.replace(i,n,t) s.replace(i,n,ct)`	ersetzt max. n Zeichen in s ab Index i durch t bzw. ct (falls $i \leq s.size()$)
`s.copy(ct,n,i)`	kopiert max. n Zeichen ab Index i (Voreinst.: 0) von s nach ct (falls $i \leq s.size()$), `\0` wird *nicht* angehängt
`cout << s cin >> s`	Ausgabe, Eingabe (Länge der Zeichenkette s wird angepasst)
`getline(cin,s)`	liest Zeile von Eingabe und schreibt diese nach Entfernen von `\n` in s, liefert Zustand von `cin`

Beispiel: Zahldarstellung zur Basis B mit $2 \leq B \leq 36$

```
#include <iostream>
#include <string>

using namespace std;

string dec2base(unsigned int n, unsigned int base)
{
  string s; char c;
  int digit;

  // Abdividieren (liefert Ziffern von rechts nach links)
  do {
    digit = n%base;
    s += digit<10 ? '0'+digit : 'a'+digit-10;
    n /= base;
  } while (n!=0);

  // Ziffernreihenfolge umkehren
  for (string::size_type i=0; i<s.size()/2; ++i) {
    c = s[i];
    s[i] = s[s.size()-1-i];
    s[s.size()-1-i] = c;
  }

  return s;
}

int main()
{
  unsigned int n;
  cout << "n: "; cin >> n;

  cout << "Binaerdarstellung: " << dec2base(n,2) << endl;
  cout << "Oktaldarstellung:  " << dec2base(n,8) << endl;
  cout << "Hex.darstellung:   " << dec2base(n,16) << endl;

  return 0;
}
```