

Numerische Vektoren in STL (valarray)

- ▶ Datentyp `valarray<T>` für *stark eingeschränktes T*:
eingebaute Zahlentypen (`float`, `double`, ...), Zeiger, `complex` (STL)
- ▶ Verschachtelte `valarrays` ebenfalls möglich
- ▶ Standardkonstruktor für Länge 0, Konstruktor mit Länge, Konstruktor mit Wert und Länge (umgekehrte Reihenfolge wie bei `vector`), Initialisierungsliste mit Elementen (wie bei `vector`)
- ▶ Arithmetische operatoren für `valarrays` *selber Größe* und mit Skalar, Überladungen für `exp`, `log`, `pow`, `sqrt`, `sin`, `cos`, ...
Jeweils *punktweise*
- ▶ Methode `apply` wendet Parameter-Funktion (Zeiger, nicht Funktionsobjekt) punktweise an, liefert neues `valarray`
- ▶ Methoden `sum`, `min`, `max`, und `cshift` (zirkulärer shift)
- ▶ Methoden `begin` und `end` für range-based for loops; keine ordentlichen Iteratoren

Beispiel: Skalarprodukt mit valarray

```
valarray_scalar.cpp

#include <iostream>
#include <valarray>

using namespace std;

int main() {
    valarray<double> a(5), b{1.0, 2.0, 3.0, 4.0, 5.0};

    cout << "a: ";
    for (double& x: a) cin >> x;

    cout << "skalar(a, b) = " << (a*b).sum() << endl;
}
```

```
a: 5 4 3 2 1
skalar(a, b) = 35
```

Indexmengen

- ▶ `valarray` ist selbst kein Matrixtyp, effiziente Implementierung von Matrixtyp damit jedoch möglich
- ▶ Hierfür: Klassen `slice`, `gslice` und `valarray<bool>`, `valarray<size_t>` beschreiben *Indexmengen*
- ▶ Jeweils subscript-Operator `[]` überladen mit Indexmenge als Parameter, liefert Teilvektor mit Referenz-Semantik
- ▶ `valarray` hat Konvertier-Konstruktoren für Teilvektoren, jedoch performance Auswirkungen wegen Kopie
- ▶ `valarray<bool>` ist bitmaske, `valarray<size_t>` ist Vektor von Indizes

`slice`

`slice(i_0 , n , h)` liefert $(i_0 + kh)_{k=0, \dots, n-1}$

Für Matrix $(a_{ij})_{\substack{i=0, \dots, m-1 \\ j=0, \dots, n-1}}$ gespeichert als `a[$i \cdot n + j$]`:

i -te Zeile Index-Folge $(i \cdot n + j)_{j=0, \dots, n-1}$ entspricht `slice($i \cdot n$, n , 1)`

j -te Spalte Index-Folge $(i \cdot n + j)_{i=0, \dots, m-1}$ entspricht `slice(j , m , n)`

Beispiel: slice

```

slice.cpp

#include <iostream>
#include <iomanip>
#include <valarray>

using namespace std;

slice zeile(size_t i, size_t m, size_t n)
{ return slice(i * n, n, 1); }
slice spalte(size_t j, size_t m, size_t n)
{ return slice(j, m, n); }

void ausgabe_vektor(const valarray<double>& a) {
    for (double x: a)
        cout << setw(3) << x;
}

void ausgabe_matrix(const valarray<double>& a, size_t n) {
    for (size_t k = 0; k < a.size(); k++) {
        cout << setw(3) << a[k];
        if ((k+1) % n == 0)
            cout << endl;
    }
}

int main() {
    size_t i, j, l, m, n;
    cout << "m n: "; cin >> m >> n;
    cout << "i j l: "; cin >> i >> j >> l;

    valarray<double> a(m*n);

    for (size_t k = 0; k < a.size(); k++) a[k] = k;

    cout << "Zeile " << i << ": ";
    ausgabe_vektor(a[zeile(i, m, n)]); cout << endl;
    cout << "Spalte " << j << ": ";
    ausgabe_vektor(a[spalte(j, m, n)]); cout << endl;

    a[zeile(i, m, n)] += a[zeile(l, m, n)];

    cout << "a:" << endl;
    ausgabe_matrix(a, n); cout << endl;
}

```

```

m n: 3 4
i j l: 1 2 0
Zeile 1:  4  5  6  7
Spalte 2:  2  6 10
a:
 0  1  2  3
 4  6  8 10
 8  9 10 11

```

gslice

Mehrdimensionale Verallgemeinerung von slice

start gleich, aber zwei beliebig (aber gleich) lange valarrays für size und stride

`gslice(i_0 , { n }, { h })` liefert: `slice(i_0 , n , h)`

`gslice(i_0 , { n_0 , n_1 , ...}, { h_0 , h_1 , ...})` liefert:

`gslice($i_0 + 0 \cdot h_0$, { n_1 , ...}, { h_1 , ...}), gslice($i_0 + 1 \cdot h_0$, { n_1 , ...}, { h_1 , ...}),
..., gslice($i_0 + (n_0 - 1) \cdot h_0$, { n_1 , ...}, { h_1 , ...})`

Beispiel: gslice

```
gslice.cpp

#include <iostream>
#include <iomanip>
#include <valarray>

using namespace std;

gslice submatrix(size_t mp, size_t np, size_t n)
{ return gslice(0, {mp, np}, {n, 1}); }

void ausgabe_matrix(const valarray<double>& a, size_t n) {
    for (size_t k = 0; k < a.size(); k++) {
        cout << setw(3) << a[k];
        if ((k+1) % n == 0)
            cout << endl;
    }
}

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

    valarray<double> a(m*n);
    for (size_t k = 0; k < a.size(); k++) a[k] = k;

    cout << "a:" << endl;
    ausgabe_matrix(a, n); cout << endl;

    cout << "sub a:" << endl;
}
```

```
    ausgabe_matrix(a[submatrix(mp, np, n)], np); cout << endl;
}
```

```
m n: 3 4
mp np: 2 3
a:
0 1 2 3
4 5 6 7
8 9 10 11
```

```
sub a:
0 1 2
4 5 6
```

Beispiel: Weitere Indextmengen

```
misc_slices.cpp

#include <iostream>
#include <iomanip>
#include <valarray>
#include <algorithm>

using namespace std;

void ausgabe_vektor(const valarray<double>& a) {
    for (double x: a)
        cout << setw(3) << x;
}

void ausgabe_matrix(const valarray<double>& a, size_t n) {
    for (size_t k = 0; k < a.size(); k++) {
        cout << setw(3) << a[k];
        if ((k+1) % n == 0)
            cout << endl;
    }
}

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

    valarray<double> a(m*n);

    for (size_t k = 0; k < a.size(); k++) a[k] = k;

    valarray<bool> bv(m*n);
    bv = false;
}
```

```
bv[0] = bv[n - 1] = bv[n * (m-1)] = bv[m*n - 1] = true;
a[bv] = 0;
cout << "a:" << endl;
ausgabe_matrix(a, n); cout << endl;
```

```
valarray<size_t> iv(min(m, n));
for (size_t i = 0; i < min(m, n); i++)
    iv[i] = i*(n + 1);
cout << "diag(a):";
ausgabe_vektor(a[iv]); cout << endl;
}
```

```
m n: 4 3
a:
0 1 0
3 4 5
6 7 8
0 10 0

diag(a): 0 4 8
```

Beispiel: Matrix-Datentyp, Deklarationen

```

matrix.h

#pragma once

#include <valarray>

template<class T> class matrix {
private:
    std::valarray<T> v;
    std::size_t m, n;

public:
    matrix(std::valarray<T>& v, std::size_t m, std::size_t n);
    matrix(std::size_t m_ = 0, std::size_t n_ = 0, T val_ = T());

    size_t rows() const; size_t columns() const;

    class matrix_slice : public std::slice {
    friend class matrix;
    private:
        std::valarray<T>& v;

        matrix_slice(std::valarray<T>& v_, std::size_t start,
            ↪ std::size_t size, std::size_t stride);

    public:
        T& operator[] (std::size_t i);
        T operator[] (std::size_t i) const;
        operator std::valarray<T>() const;
    };
};

```

```

matrix_slice operator[] (std::size_t i);
matrix_slice row(std::size_t i);
matrix_slice column(std::size_t j);

operator std::valarray<T>() const;

T* begin();
T* end();
};

```

Beispiel: Matrix-Datentyp, Implementierung

```

matrix.cpp

#include <valarray>
#include <stdexcept>

#include "matrix.h"

using namespace std;

template<class T>
matrix<T>::matrix(std::valarray<T>& v_, std::size_t m_,
    ↪ std::size_t n_)
    : v(v_), m(m_), n(n_) {
    if (v.size() != m * n)
        throw invalid_argument("valarray of wrong size");
}

template<class T>
matrix<T>::matrix(size_t m_, size_t n_, T val_)
    : v(val_, m_ * n_), m(m_), n(n_) {}

template<class T>
size_t matrix<T>::rows() const { return m; }

template<class T>
size_t matrix<T>::columns() const { return n; }

template<class T>
matrix<T>::matrix_slice::matrix_slice
    (valarray<T>& v_, size_t start, size_t size, size_t stride)
    : slice(start, size, stride), v(v_) {}

template<class T>

```

```

T& matrix<T>::matrix_slice::operator[] (size_t i)
{ return v[start() + i*stride()]; }

template<class T>
T matrix<T>::matrix_slice::operator[] (size_t i) const
{ return v[start() + i*stride()]; }

template<class T>
matrix<T>::matrix_slice::operator valarray<T>() const {
    valarray<T> v(size());
    for (size_t i = 0; i < size(); i++)
        v[i] = (*this)[i];
    return v;
}

template<class T>
typename matrix<T>::matrix_slice { return row(i); }

template<class T>
typename matrix<T>::matrix_slice matrix<T>::row(size_t i)
{ return matrix<T>::matrix_slice{v, i*n, n, 1}; }

template<class T>
typename matrix<T>::matrix_slice matrix<T>::column(size_t j)
{ return matrix<T>::matrix_slice{v, j, m, n}; }

template<class T>
matrix<T>::operator valarray<T>() const {
    return v;
}

template<class T>
T* matrix<T>::begin()
{ return std::begin(v); }

```

Beispiel: Matrix-Datentyp, Demonstration

```
matrix_basic.cpp

#include <iostream>

#include "matrix.h"

using namespace std;

int main() {
    size_t m, n, i, j;
    cout << "m n: "; cin >> m >> n;
    cout << "i j: "; cin >> i >> j;
    matrix<double> a{m, n};
    cout << "Matrix a: " << endl;
    for (double& x: a) cin >> x;
    cout << (a[i][j] = 13) << endl;
}
```

```
m n: 3 4
i j: 1 2
Matrix a:
1 2 3 4
5 6 7 8
9 10 11 12
13
```

Beispiel: Matrix-Multiplikation

```
matrixmul.cpp

#include <iostream>
#include <iomanip>

#include "matrix.h"

using namespace std;

int main() {
    size_t m, n;
    cout << "m n: "; cin >> m >> n;
    matrix<double> a{m, n};
    cout << "Matrix a: " << endl;
    for (double& x: a) cin >> x;
    cout << endl;

    size_t k, l;
    cout << "k l: "; cin >> k >> l;
    matrix<double> b{k, l};
    cout << "Matrix b: " << endl;
    for (double& x: b) cin >> x;
    cout << endl;

    matrix<double> c{m, l};
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < l; j++) {
            valarray<double> as = a.row(i), bs = b.column(j);
            c[i][j] = (as * bs).sum();
        }
    }
}
```

```
for (size_t i = 0; i < m; i++) {
    for (size_t j = 0; j < l; j++) {
        cout << setw(3) << c[i][j];
        cout << endl;
    }
}
```

```
m n: 4 4
Matrix a:
3 2 1 4
1 0 2 3
3 2 1 2
3 2 1 4

k l: 4 4
Matrix b:
1 2 1 4
0 1 0 3
4 0 4 2
1 2 1 4

11 16 11 36
12 8 12 20
9 12 9 28
11 16 11 36
```

Beispiel: Strassen Algorithmus

```

strassen.cpp

#include <valarray>
#include <stdexcept>

#include "matrix.h"

using namespace std;

template<class T>
valarray<T>& matmul_v(valarray<T>& a, const valarray<T>& b,
    ↪ size_t n) {
    if (n == 1)
        return a = a * b;

    gslice
        s11(0, {n / 2, n / 2}, {n, 1}),
        s12(n / 2, {n / 2, n / 2}, {n, 1}),
        s21(n * n / 2, {n / 2, n / 2}, {n, 1}),
        s22(n * n / 2 + n / 2, {n / 2, n / 2}, {n, 1});

    valarray<T> h = b[s11]; h += b[s22];
    valarray<T> m1 = a[s11]; m1 += a[s22];
    m1 = matmul_v(m1, h, n / 2);
    h = b[s11];
    valarray<T> m2 = a[s21]; m2 += a[s22];
    m2 = matmul_v(m2, h, n / 2);
    h = b[s12]; h -= b[s22];
    valarray<T> m3 = a[s11];
    m3 = matmul_v(m3, h, n / 2);
    h = b[s21]; h -= b[s11];

```

```

    valarray<T> m4 = a[s22];
    m4 = matmul_v(m4, h, n / 2);
    h = b[s22];
    valarray<T> m5 = a[s11]; m5 += a[s12];
    m5 = matmul_v(m5, h, n / 2);
    h = b[s11]; h += b[s12];
    valarray<T> m6 = a[s21]; m6 -= a[s11];
    m6 = matmul_v(m6, h, n / 2);
    h = b[s21]; h += b[s22];
    valarray<T> m7 = a[s12]; m7 -= a[s22];
    m7 = matmul_v(m7, h, n / 2);

    a[s11] = m1 + m4 - m5 + m7;
    a[s12] = m3 + m5;
    a[s21] = m2 + m4;
    a[s22] = m1 - m2 + m3 + m6;
    return a;
}

```

matmul strassen

Beispiel: Strassen Algorithmus (Forts.)

```

matmul strassen

bool power_of_two(size_t i)
{ return i && !(i & (i - 1)); }

template<class T>
matrix<T> matmul(matrix<T> a, matrix<T> b) {
    if (a.rows() != a.columns() || a.rows() != b.rows() ||
    ↪ a.rows() != b.columns())
        throw invalid_argument("matrices not square or not of same
    ↪ dimension");
    if (!power_of_two(a.columns()))
        throw invalid_argument("matrix dimension not power of
    ↪ two");

    valarray<T> av = a;
    av = matmul_v<T>(av, b, a.rows());
    return matrix<T>(valarray<T>(av), a.rows(), a.columns());
}

template matrix<double> matmul<double>(matrix<double>,
    ↪ matrix<double>);

```

```

strassen.h

#pragma once

#include "matrix.h"

template<class T>
matrix<T> matmul(matrix<T> a, matrix<T> b);

```

Beispiel: Strassen Algorithmus (Forts.)

matrixmul_strassen.cpp

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

#include "strassen.h"
#include "matrix.h"

using namespace std;

int main() {
    size_t m, n;
    cout << "m n: "; cin >> m >> n;
    matrix<double> a{m, n};
    cout << "Matrix a: " << endl;
    for (double& x: a) cin >> x;
    cout << endl;

    size_t k, l;
    cout << "k l: "; cin >> k >> l;
    matrix<double> b{k, l};
    cout << "Matrix b: " << endl;
    for (double& x: b) cin >> x;
    cout << endl;

    matrix<double> c = matmul(a, b);
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < l; j++)
            cout << setw(3) << c[i][j];
        cout << endl;
    }
}
```

}

```
m n: 4 4
Matrix a:  3  2  1  4
           1  0  2  3
           3  2  1  2
           3  2  1  4

k l: 4 4
Matrix b:
           1  2  1  4
           0  1  0  3
           4  0  4  2
           1  2  1  4

11 16 11 36
12  8 12 20
 9 12  9 28
11 16 11 36
```