0.1 “ Vector Matrix”

0.1.1

vect ( ) matr ( ) C++ . , , , , , (, ,), , . main.

0.1.2

- **vect ( ):**
  - .
  - private :
    - dim: ( ).
    - v: double, .
    - num: ( ).
    - static count: .
  - public :
    - :
      - vect(): ( ).
      - vect(int n): , n, .
      - vect(int n, double* x): , n x.
      - vect(const vect& x): ( ).
    - ~vect(): ( ).
    - :
      - vect operator+(const vect& r) const: .
      - vect& operator=(const vect& r): .
      - vect& operator-(): ( ).
      - double operator*(const vect& r) const: .
      - double& operator[](int i): ( ).
      - double operator[](int i) const: ( ).
    - :
      - int getDim() const: .
      - double getElement(int i) const: i.
      - void setElement(int i, double val): val i.
      - static int getCount(): .

        - `void print() const`: .
        - :
            - `friend vect operator-(const vect& l, const vect& r)`: ( ).
            - `friend vect operator*(double k, const vect& r)`: (,).
- **`matr` ( ):**
    - .
    - `private` :
        - `dim`: ( ).
        - `a`: `double`, ( ).
    - `public` :
        - :
            - `matr()`: ( ).
            - `matr(int n)`: , `n x n`, .
            - `matr(int n, double* x)`: , `n x n` `x`.
            - `matr(const matr& x)`: ( ).
        - `~matr()`: ( ).
        - :
            - `matr operator+(const matr& r) const`: .
            - `matr operator-(const matr& r) const`: .
            - `matr operator-() const`: ( ).
            - `matr operator*(const matr& r) const`: .
            - `vect operator*(const vect& r) const`: .
            - `matr& operator=(const matr& r)`: .
        - :
            - `int getDim() const`: .
            - `double getElement(int i, int j) const`: `i` `j`.
            - `void setElement(int i, int j, double val)`: `val` `i` `j`.
        - `int ind(int i, int j) const`: `a` `i` `j` (1).
        - `void print() const`: .
        - :
            - `friend matr operator*(double k, const matr& r)`: (,).

0.1.3

vect matr :

1. : vect matr, . private public . count vect .
2. : vect matr (, , , , -). . . .
3. : . , , , . (, , ,). std::out_of_range .
4. : getDim(), getElement(), setElement() .
5. : ( ) , vect matr, private .
6. **main:** main vect matr, .

:

- : , . , .
- : , . , , .
- : .

0.1.4

vect matr, . , .

( ):

```
vect()     1
vect(int n)     2
vect(int n)     3
```

```
    v1: (1, 2, 3)
    vect(int n)     4
    vect(int n)     5
    vect(int n)     6
    v2: (4, 5, 6)
    vect(int n)     7
v1 + v2 =         7: (5, 7, 9)
    vect(int n)     8
    vect(int n)     9
2.0 * v1 =         9: (2, 4, 6)
             v1   v2: 32
    matr(int n)
    matr(int n)
     m1:
    1     2
    3     4
    matr(int n)
    matr(int n)
     m2:
    5     6
    7     8

    matr(int n)
m1 + m2:
    6     8
   10    12

    matr(int n)
m1 * m2:
   19    22
   43    50

    vect(int n)
m1 * v1:
   !                 .
       1: (1, 2, 3)
```

, , ,
. .

0.1.5

vect matr,
. ,
.
:

- .
- .
- .
- .

:

- .
- .
- .
- - .

, :

- (, , ,).
- ( ).
- (, SIMD-).
- , , .

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

class vect {
private:
    int dim; //
    double* v; //
    int num; //
    static int count; //

public:
    vect(); //
    vect(int n); //
    vect(int n, double* x); //
    vect(const vect& x); //

    ~vect(); //

    vect operator+(const vect& r) const; //
    vect& operator=(const vect& r); //
    vect& operator-(); //
    double operator*(const vect& r) const; //
    double& operator[](int i); //            (       )
    double operator[](int i) const; //              (      )
    void print() const; //
    int getDim() const; //
    double getElement(int i) const; //
    void setElement(int i, double val);
    static int getCount(); //

    friend vect operator-(const vect& l, const vect& r); //
    friend vect operator*(double k, const vect& r); //
};

class matr {
private:
    int dim; //
    double* a; //

public:
    matr(); //
    matr(int n); //
    matr(int n, double* x); //
    matr(const matr& x); //

    ~matr(); //

    int ind(int i, int j) const; //
    matr operator+(const matr& r) const; //
    matr operator-(const matr& r) const; //
    matr operator-() const; //
    matr operator*(const matr& r) const; //
    matr& operator=(const matr& r); //
    vect operator*(const vect& r) const; //
    void print() const; //

    int getDim() const; //
    double getElement(int i, int j) const; //
    void setElement(int i, int j, double val); //

    friend matr operator*(double k, const matr& r); //
};

//
int vect::count = 0;

vect::vect() {
    count++;
    num = count;
    cout << "    vect()     " << num << endl;
    dim = 0;
    v = nullptr;
}

vect::vect(int n) {
    count++;
    num = count;
    cout << "    vect(int n)     " << num << endl;
    dim = n;
    v = new double[dim];
    for (int i = 0; i < dim; ++i) {
        v[i] = 0.0;
    }
}

vect::vect(int n, double* x) {
    count++;
    num = count;
    cout << "    vect(int n, double* x)     " << num << endl;
    dim = n;
    v = new double[dim];
    for (int i = 0; i < dim; ++i) {
        v[i] = x[i];
    }
}

vect::vect(const vect& x) {
    count++;
    num = count;
    cout << "    vect(const vect& x)     " << num << endl;
    dim = x.dim;
    v = new double[dim];
    for (int i = 0; i < dim; ++i) {
        v[i] = x.v[i];
    }
}

vect::~vect() {
    cout << "    vect     " << num << endl;
    delete[] v;
}

vect vect::operator+(const vect& r) const {
    if (dim != r.dim) {
        cout << "   !               ." << endl;
        return *this;
    }

    vect result(dim);
    for (int i = 0; i < dim; ++i) {
        result.v[i] = v[i] + r.v[i];
    }
    return result;
}

vect& vect::operator=(const vect& r) {
    if (this == &r) {
        return *this;
    }

    if (dim != r.dim) {
        delete[] v;
        dim = r.dim;
        v = new double[dim];
    }

    for (int i = 0; i < dim; ++i) {
        v[i] = r.v[i];
    }
    return *this;
}

vect& vect::operator-() {
    cout << "        vect    " << num << endl;
    for (int i = 0; i < dim; ++i) {
        v[i] = -v[i];
    }
    return *this;
}

double vect::operator*(const vect& r) const {
    if (dim != r.dim) {
        cout << "   !               ." << endl;
        return 0.0;
    }

    double result = 0.0;
    for (int i = 0; i < dim; ++i) {
        result += v[i] * r.v[i];
    }
    return result;
}
```

```cpp
double& vect::operator[](int i) {
    if (i < 0 || i >= dim) {
        throw std::out_of_range("                    ");
    }
    return v[i];
}

double vect::operator[](int i) const {
    if (i < 0 || i >= dim) {
        throw std::out_of_range("                    ");
    }
    return v[i];
}

void vect::print() const {
    cout << "vect     " << num << ": (";
    for (int i = 0; i < dim; ++i) {
        cout << v[i];
        if (i < dim - 1) {
            cout << ", ";
        }
    }
    cout << ")" << endl;
}

int vect::getDim() const {
    return dim;
}

double vect::getElement(int i) const {
    if (i < 0 || i >= dim) {
        cout << "    !                     ." << endl;
        return 0.0;
    }
    return v[i];
}

void vect::setElement(int i, double val) {
    if (i < 0 || i >= dim) {
        cout << "    !                     ." << endl;
        return;
    }
    v[i] = val;
}
```

```
int vect::getCount() {
    return count;
}

vect operator-(const vect& l, const vect& r) {
    if (l.dim != r.dim) {
        cout << "   !               ." << endl;
        return l;
    }

    vect result(l.dim);
    for (int i = 0; i < l.dim; ++i) {
        result.v[i] = l.v[i] - r.v[i];
    }
    return result;
}

vect operator*(double k, const vect& r) {
    vect result(r.dim);
    for (int i = 0; i < r.dim; ++i) {
        result.v[i] = k * r.v[i];
    }
    return result;
}

matr::matr() {
    cout << "    matr()" << endl;
    dim = 0;
    a = nullptr;
}

matr::matr(int n) {
    cout << "    matr(int n)" << endl;
    dim = n;
    a = new double[dim * dim];
    for (int i = 0; i < dim * dim; ++i) {
        a[i] = 0.0;
    }
}

matr::matr(int n, double* x) {
    cout << "    matr(int n, double* x)" << endl;
    dim = n;
    a = new double[dim * dim];
    for (int i = 0; i < dim * dim; ++i) {
        a[i] = x[i];
    }
}

matr::matr(const matr& x) {
    cout << "    matr(const matr& x)" << endl;
    dim = x.dim;
    a = new double[dim * dim];
    for (int i = 0; i < dim * dim; ++i) {
        a[i] = x.a[i];
    }
}

matr::~matr() {
    cout << "    matr" << endl;
    delete[] a;
}

int matr::ind(int i, int j) const {
    return dim * (i - 1) + (j - 1);
}

void matr::print() const {
    cout << "     " << dim << "x" << dim << endl;
    for (int i = 1; i <= dim; ++i) {
        for (int j = 1; j <= dim; ++j) {
            cout.width(5);
            cout << a[ind(i, j)] << " ";
        }
        cout << endl;
    }
}

int matr::getDim() const {
    return dim;
}

double matr::getElement(int i, int j) const {
    return a[ind(i, j)];
}

void matr::setElement(int i, int j, double val) {
    a[ind(i, j)] = val;
}

matr matr::operator+(const matr& r) const {
    cout << "         " << endl;
    if (dim != r.dim) {
        cout << "   !               ." << endl;
        return *this;
    }

    matr result(dim);
    for (int i = 1; i <= dim; ++i) {
        for (int j = 1; j <= dim; ++j) {
            result.a[ind(i, j)] = a[ind(i, j)] + r.a[ind(i, j)];
        }
    }
    return result;
}

matr matr::operator-(const matr& r) const {
    cout << "         " << endl;
    if (dim != r.dim) {
        cout << "   !               ." << endl;
        return *this;
    }

    matr result(dim);
    for (int i = 1; i <= dim; ++i) {
        for (int j = 1; j <= dim; ++j) {
            result.a[ind(i, j)] = a[ind(i, j)] - r.a[ind(i, j)];
        }
    }
    return result;
}

matr matr::operator-() const {
    cout << "            " << endl;

    matr result(dim);
    for (int i = 1; i <= dim; ++i) {
        for (int j = 1; j <= dim; ++j) {
            result.a[ind(i, j)] = -a[ind(i, j)];
        }
    }
    return result;
}

matr matr::operator*(const matr& r) const {
    cout << "         " << endl;
    if (dim != r.dim) {
        cout << "   !               ." << endl;
        return *this;
    }

    matr result(dim);
    for (int i = 1; i <= dim; ++i) {
        for (int j = 1; j <= dim; ++j) {
            result.setElement(i, j, 0.0);
            for (int k = 1; k <= dim; ++k) {
                result.setElement(i, j, result.getElement(i, j) + getElement(i, k) * r.getEl
            }
        }
    }
    return result;
}

matr& matr::operator=(const matr& r) {
    cout << "           " << endl;
    if (this == &r) return *this;

    if (dim != r.dim) {
        delete[] a;
        dim = r.dim;
        a = new double[dim * dim];
    }

    for (int i = 0; i < dim * dim; ++i) {
        a[i] = r.a[i];
    }

    return *this;
}

vect matr::operator*(const vect& r) const {
    cout << "                " << endl;
    if (dim != r.getDim()) {
        cout << "   !               ." << endl;
        return r;
    }

    vect result(dim);
    for (int i = 1; i <= dim; ++i) {
        result.setElement(i - 1, 0.0);
        for (int j = 1; j <= dim; ++j) {
            result.setElement(i - 1, result.getElement(i - 1) + getElement(i, j) * r[j - 1]
        }
    }
    return result;
}

matr operator*(double k, const matr& r) {
    cout << "                 (   )" << endl;
    matr result(r.dim);
    for (int i = 1; i <= r.dim; ++i) {
        for (int j = 1; j <= r.dim; ++j) {
            result.setElement(i, j, k * r.getElement(i, j));
        }
    }
    return result;
}

int main() {
    setlocale(LC_ALL, "rus");

    // 1.
    cout << "\n          " << endl;
    vect v1(3);
    v1.setElement(0, 1.0);
    v1.setElement(1, 2.0);
    v1.setElement(2, 3.0);
    v1.print();

    vect v2(3);
    v2.setElement(0, 4.0);
    v2.setElement(1, 5.0);
    v2.setElement(2, 6.0);
    v2.print();

    // 2.
    vect v3 = v1 + v2;
    cout << "\nv1 + v2 = ";
    v3.print();

    // 3.
    vect v4 = 2.0 * v1;
    cout << "\n2.0 * v1 = ";
    v4.print();

    // 4.
    double scalarProduct = v1 * v2;
    cout << "\n             v1   v2: " << scalarProduct << endl;

    // 5.
    cout << "\n          " << endl;
    matr m1(2);
    m1.setElement(1, 1, 1.0);
    m1.setElement(1, 2, 2.0);
    m1.setElement(2, 1, 3.0);
    m1.setElement(2, 2, 4.0);
    cout << "\n    m1:" << endl;
    m1.print();

    matr m2(2);
    m2.setElement(1, 1, 5.0);
    m2.setElement(1, 2, 6.0);
    m2.setElement(2, 1, 7.0);
    m2.setElement(2, 2, 8.0);
    cout << "\n    m2:" << endl;
    m2.print();

    // 6.
    matr m3 = m1 + m2;
    cout << "\nm1 + m2:" << endl;
    m3.print();

    // 7.
    matr m4 = m1 * m2;
    cout << "\nm1 * m2:" << endl;
    m4.print();

    // 8.
    vect v5 = m1 * v1;
    cout << "\nm1 * v1:" << endl;
    v5.print();

    return 0;
}
```