Nicoleta.Radu.T6

C++

# Matrici bidimensionale

Aplicatia 1. Matricea bidimensionala prin utilizarea supraincarcarii operatorilor aritmetici si de I/O.

#include&lt;iostream&gt;

#include&lt;iomanip&gt;

using namespace std;

class matrice

{

int\*\* a;

int m, n; //m - linii, n - coloane

string s;

public:

matrice(int, int, const char\*); //constructorul

friend istream&amp; operator&gt;&gt;(istream&amp;, matrice&amp;); //citirea unei matrice

friend ostream&amp; operator&lt;&lt;(ostream&amp;, matrice); //afisarea unei matrice

matrice operator+(matrice); //adunarea a doua matrice

matrice operator-(matrice); //diferenta intre doua matrice

matrice operator\*(matrice); //inmultirea

matrice operator~(); //transpusa

int&amp; linii(); //numarul de linii

int&amp; coloane(); //numarul de coloane

private:

matrice operator-(); //opusa unei matrice

};

inline matrice::matrice(int m\_, int n\_, const char\* s\_) :m(m\_), n(n\_), s(s\_)

{

a = new int\* [m];

for (int i = 0; i &lt; m; i++)

a[i] = new int[n];

}

istream&amp; operator&gt;&gt;(istream&amp; x, matrice&amp; mat)

{

cout &lt;&lt; &quot;Citim matricea &quot; &lt;&lt; mat.s &lt;&lt; &#39;\n&#39;;

for (int i = 0; i &lt; mat.m; i++)

for (int j = 0; j &lt; mat.n; j++)

{

cout &lt;&lt; &quot;\telementul [&quot; &lt;&lt; i + 1 &lt;&lt; &#39;,&#39; &lt;&lt; j + 1 &lt;&lt; &quot;]= &quot;;

x &gt;&gt; mat.a[i][j];

}

return x;

}

ostream&amp; operator&lt;&lt;(ostream&amp; x, matrice mat)

{

x &lt;&lt; &quot;Matricea &quot; &lt;&lt; mat.s &lt;&lt; &quot; este:\n&quot;;

for (int i = 0; i &lt; mat.m; i++)

{

for (int j = 0; j &lt; mat.n; j++)

x &lt;&lt; setw(5) &lt;&lt; mat.a[i][j];

cout &lt;&lt; &#39;\n&#39;;

}

return x;

}

matrice matrice::operator+(matrice mat)

{

matrice rez(m, n, &quot;&quot;);

for (int i = 0; i &lt; rez.m; i++)

for (int j = 0; j &lt; rez.n; j++)

rez.a[i][j] = a[i][j] + mat.a[i][j];

rez.s = s + &#39;+&#39; + mat.s;

return rez;

}

matrice matrice::operator-()

{

matrice rez(m, n, &quot;&quot;);

for (int i = 0; i &lt; rez.m; i++)

for (int j = 0; j &lt; rez.n; j++)

rez.a[i][j] = -a[i][j];

return rez;

}

matrice matrice::operator-(matrice mat)

{

matrice rez(m, n, &quot;&quot;);

rez = (\*this) + (-mat);

rez.s = s + &#39;-&#39; + mat.s;

return rez;

}

matrice matrice::operator\*(matrice mat)

{

matrice rez(m, mat.n, &quot;&quot;);

for (int i = 0; i &lt; m; i++)

for (int j = 0; j &lt; n; j++)

{

rez.a[i][j] = 0;

for (int x = 0; x &lt; rez.n; x++)

rez.a[i][j] += a[i][x] \* mat.a[x][j];

}

rez.s = s + &#39;\*&#39; + mat.s;

return rez;

}

matrice matrice::operator~()

{

matrice rez(n, m, &quot;&quot;);

for (int i = 0; i &lt; rez.m; i++)

for (int j = 0; j &lt; rez.n; j++)

rez.a[i][j] = a[j][i];

rez.s = s+&quot; Transpus&quot;;

return rez;

}

int&amp; matrice::linii()

{

return m;

}

int&amp; matrice::coloane()

{

return n;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include&quot;Header.h&quot;

int main()

{

matrice a(2, 2, &quot;A&quot;);

matrice b(2, 2, &quot;B&quot;);

cin &gt;&gt; a;

cin &gt;&gt; b;

cin.get();

system(&quot;cls&quot;);

cout &lt;&lt; a &lt;&lt; b;

//suma si diferenta matricelor

if ((a.linii() == b.linii()) and (a.coloane() == b.coloane()))

{

cout &lt;&lt; a + b;

cout &lt;&lt; a - b;

cout &lt;&lt; b - a;

}

//inmultirea matricelor

if (a.coloane() == b.linii())

cout &lt;&lt; a \* b;

//transpusa matricei

cout &lt;&lt; ~a;

cout &lt;&lt; ~b;

system(&quot;pause&quot;);

return 0;

# Vectorul de numere

#include <iostream>

using namespace std;

class Vector

{

public:

struct Extrem

{

int minimum, maximum;

};

private:

int arr[100]{};;

int sizeOfArray{0};

public:

Vector(int);

Vector(const Vector&);

friend istream& operator>>(istream&, Vector&);

friend ostream& operator<<(ostream&, Vector);

//overloaded operators methods

int& operator[](int);

int operator+();

int operator\*();

float operator-();

Vector operator\*(int);

friend Vector operator\*(int, Vector);

long long operator\*(Vector);

Extrem operator!();

//special operators

bool operator==(Vector);

Vector& operator=(Vector);

Vector operator,(Vector);

void operator()();

int operator()(int);

int operator()(Vector);

private:

bool isPrim(int);

void swap(int& a, int& b);

};

inline Vector::Vector(int dim) : sizeOfArray{dim}

{

}

inline Vector::Vector(const Vector& arrAux) : sizeOfArray{arrAux.sizeOfArray}

{

int i{0};

while (i < arrAux.sizeOfArray)

{

arr[i++] = arrAux.arr[i];

}

}

int& Vector::operator[](int i)

{

return arr[i];

}

istream& operator>>(istream& in, Vector& arrAux)

{

cout << "Introduceti valorile pentru array:" << endl;

for (int i = 0; i < arrAux.sizeOfArray; i++)

{

cout << "[" << i << "]= ";

in >> arrAux[i];

}

return in;

}

ostream& operator<<(ostream& out, Vector arrAux)

{

out << "Printing array" << endl;

cout << "[ ";

for (int i = 0; i < arrAux.sizeOfArray; i++)

{

out << arrAux[i] << " ";

}

cout << "]";

return out;

}

int Vector::operator+()

{

int rezultat{0};

for (int i = 0; i < sizeOfArray; i++)

{

rezultat = rezultat + arr[i];

}

return rezultat;

}

int Vector::operator\*()

{

int rezultat{ 1 };

for (int i = 0; i < sizeOfArray; i++)

{

rezultat = rezultat \* arr[i];

}

return rezultat;

}

float Vector::operator-()

{

return +(\*this) / (float)sizeOfArray;

}

Vector Vector::operator\*(int scalar)

{

Vector rezultat(sizeOfArray);

for (int i = 0; i < sizeOfArray; i++)

{

rezultat[i] = rezultat[i] \* scalar;

}

return rezultat;

}

long long Vector::operator\*(Vector auxArr)

{

long long rezultat{ 0 };

for (int i = 0; i < sizeOfArray; i++)

{

rezultat += static\_cast<long long>(arr[i]) \* auxArr[i];

}

return rezultat;

}

Vector operator\*(int scalar, Vector auxArr)

{

return auxArr \* scalar;

}

Vector Vector::operator,(Vector auxArr)

{

Vector rezultat(sizeOfArray);

for (size\_t i = 0; i < sizeOfArray; i++)

{

rezultat[i] = arr[i];

}

return rezultat;

}

void Vector::swap(int& a, int& b)

{

int\* aux{nullptr};

aux = &a;

a = b;

b = \*aux;

}

bool Vector::isPrim(int obiect)

{

bool prim = false;

if (obiect == 0)

{

return prim;

}

if (obiect == 1 || obiect == 2)

{

prim = true;

return prim;

}

for (int i = 3; i < obiect / 2; i = i + 2)

{

if (obiect % i == 0)

{

prim = false;

return prim;

}

}

return prim;

}

bool Vector::operator==(Vector auxArr)

{

bool isEqual = true;

for (int i = 0; i < sizeOfArray; i++)

{

if (arr[i] != auxArr.arr[i])

{

isEqual = false;

return isEqual;

}

}

return isEqual;

}

Vector& Vector::operator=(Vector auxArr)

{

for (int i = 0; i < sizeOfArray; i++)

{

auxArr.arr[i] = arr[i];

}

return auxArr;

}

void Vector::operator()()

{

for (int i = 0; i < sizeOfArray - 1; i++)

{

for (int j = i + 1; j < sizeOfArray; j++)

{

if (arr[j] < arr[i])

{

swap(arr[i], arr[j]);

}

}

}

}

int Vector::operator()(int obiect)

{

for (int i = 0; i < sizeOfArray; i++)

{

if (arr[i] == obiect)

{

return i;

}

}

return 0;

}

int Vector::operator()(Vector auxArr)

{

int nrPrime{ 0 };

for (int i = 0; i < auxArr.sizeOfArray; i++)

{

if (isPrim(auxArr[i]))

{

nrPrime++;

}

}

return nrPrime;

}