Structuri de Date
Tema 1

David Andreea Raluca – grupa 10LF221

1. Se consideră doi vectori v_1 cu nr_1 elemente şi v_2 cu nr_2 elemente. Vectorul v_1 este sortat crescător şi vectorul v_2 este sortat descrescător. Să se obţină un al treilea vector v_3 care conţine atât elementele lui v_1 cât şi elementele lui v_2 şi care este sortat crescător. Folosiţi o metodă eficientă. (1p)

```
#include <iostream>

// Complexitatea este O(nr1+nr2)

void AlocareDinamica(int*& vector,int dimensiune)
{
	vector = new int[dimensiune];
}

void DealocareDinamica(int*& vector)
{
	delete[]vector;
}

void CitireDateProblema(int&nr1,int*&v1,int&nr2,int*&v2)
{
	std::cout << "Introduceti dimensiunea primului vector:" << std::endl;
	std::cin >> nr1;
	AlocareDinamica(v1, nr1);
	std::cout << "Introduceti elementele primului vector:" << std::endl;
	for (int index = 0; index < nr1; index++)
		std::cin >> v1[index];
	std::cout << "Introduceti dimensiunea celui de al doilea vector:" << std::endl;
	std::cin >> nr2;
	AlocareDinamica(v2, nr2);
	std::cout << "Introduceti elementele celui de al doilea vector:" << std::endl;
	for (int index = 0; index < nr2; index++)
		std::cin >> v2[index];
}

void AfisareVector(int* vector, int dimensiune)
{
	for (int index = 0; index < dimensiune; index++)
		std::cout << vector[index] << ' ';
}

void Interclasare(int nr1,int*v1,int nr2,int*v2)
```

```
{
	int* v3;
	AlocareDinamica(v3, nr1 + nr2);
	int index1 = 0, index2 = nr2-1, loc = 0;
	while ((nr1 != index1) && (index2 != 0))
	{
		if (v1[index1] < v2[index2])
		{
			v3[loc] = v1[index1];
			loc++;
			index1++;
		}
		else if (v2[index2] < v1[index1])
		{
			v3[loc] = v2[index2];
			loc++;
			index2--;
		}
		else
		{
			v3[loc] = v1[index1];
			loc++;
			index1++;
			v3[loc] = v2[index2];
			loc++;
			index2--;
		}
	}
	while (index1 < nr1)
	{
		v3[loc] = v1[index1];
		loc++;
		index1++;
	}
	while (index2 >= 0)
	{
		v3[loc] = v2[index2];
		loc++;
		index2--;
	}
	AfisareVector(v3, nr1 + nr2);
	DealocareDinamica(v3);
}

int main()
{
	int nr1, nr2;
	int*v1, *v2;
	CitireDateProblema(nr1, v1,nr2,v2);
	Interclasare(nr1, v1, nr2, v2);
	DealocareDinamica(v1);
```

```
	DealocareDinamica(v2);
	return 0;
}
```

2. Se consideră un vector conţinând nr numere naturale. Scrieţi o funcţie care are ca parametru vectorul şi dimensiunea acestuia şi care returnează cel mai mare număr natural care se poate forma cu toate cifrele pare ale numerelor existente în vector. Folosiţi un algoritm eficient. (1p)
Exemplu: pentru $v = \{369, 113, 2, 0, 33, 1354, 42\}$ funcţia va return 644220.

```
#include <iostream>

//Complexitatea de timp este O(n)

void CitireDateProblema(int&nr,int*&vector)
{
	std::cout << "Introduceti dimensiunea vectorului:" << std::endl;
	std::cin >> nr;
	vector = new int[nr];
	std::cout << "Introduceti elementele vectorului:" << std::endl;
	for (int index = 0; index < nr; index++)
		std::cin >> vector[index];
}

void AflareFrecventa(int numar, int frecventaCifrePare[])
{
	if (numar == 0)
		frecventaCifrePare[0]++;
	else
	{
		while (numar > 0)
		{
			frecventaCifrePare[numar % 10]++;
			numar = numar / 10;
		}
	}
}

int AflareCelMaiMareNr(int nr,int*vector)
{
	int frecventaCifrePare[9];
	int numar = 0;
	for (int index = 0; index <= 8; index++)
		frecventaCifrePare[index] = 0;
	for (int index = 0; index < nr; index++)
	{
```

```
                AflareFrecventa(vector[index], frecventaCifrePare);
        }
        for (int index = 8; index >= 0; index=index-2)
        {
                while (frecventaCifrePare[index] != 0)
                {
                        numar = numar * 10 + index;
                        frecventaCifrePare[index]--;
                }
        }
        return numar;
}

int main()
{
        int nr;
        int* vector;
        CitireDateProblema(nr,vector);
        std::cout << "Cel mai mare numar format din cifrele pare este " << AflareCelMaiMareNr(nr,vector);
        delete[] vector;
        return 0;
}
```

3. Se citesc dintr-un fişier un număr de elevi. Fiecare elev are un nume, un prenume şi 3 note, numere naturale. Se va folosi pentru un elev un **tuple** cu câmpurile nume şi prenume de tip **string** (căutati pe net documentaţie) şi cu trei câmpuri de note de tip **int**. Elevii vor fi memoraţi într-un obiect de tip **std::vector<std::tuple<std::string, std::string, int, int, int> >**. Să se sorteze vectorul de elevi descrescător după medie şi să se afişeze frumos, punând în evidenţă elevii cu media mai mică decât 5. (2p)

```
#include <iostream>
#include <fstream>
#include <tuple>
#include <vector>
#include <algorithm>

//Complexitatea de timp este O(nlogn)

void CitireDate(std::vector<std::tuple<std::string,std::string,int,int,int>>& listaElevi,int& nrPersoane)
{
        std::ifstream fin("Fisier.in");
        std::tuple<std::string, std::string, int, int, int > elev;
        while (!fin.eof())
        {
                fin >> std::get<0>(elev) >> std::get<1>(elev) >> std::get<2>(elev) >> std::get<3>(elev) >>
std::get<4>(elev);
                listaElevi.push_back(elev);
                nrPersoane++;
```

```
	}
	fin.close();
}

struct Comparator
{
	bool operator()(std::tuple<std::string,std::string, int, int, int> Elev1, std::tuple<std::string,std::string, int, int, int> Elev2)
	{
		if (((std::get<2>(Elev1) + std::get<3>(Elev1) + std::get<4>(Elev1)) / 3.0) > ((std::get<2>(Elev2) + std::get<3>(Elev2) + std::get<4>(Elev2)) / 3.0))
			return 1;
		return 0;
	}
};

void SortareDescrescator(std::vector<std::tuple<std::string, std::string, int, int, int>>& listaElevi, int nrPersoane)
{
	std::sort(listaElevi.begin(), listaElevi.end(), Comparator());
}

void AfisareVector(std::vector<std::tuple<std::string,std::string,int,int,int>> listaElevi,int nrPersoane)
{
	int ok = 0;
	for (int index = 0; index < nrPersoane; index++)
	{
		if (((std::get<2>(listaElevi[index]) + std::get<3>(listaElevi[index]) + std::get<4>(listaElevi[index])) / 3.0 < 5) && (ok == 0))
		{
			ok = 1;
			std::cout << std::endl;
			std::cout << "Lista elevilor cu mediile < 5:" << std::endl;
		}
		std::cout << std::get<0>(listaElevi[index]) << ' ';
		std::cout << std::get<1>(listaElevi[index]) << ' ';
		std::cout << std::get<2>(listaElevi[index]) << ' ';
		std::cout << std::get<3>(listaElevi[index]) << ' ';
		std::cout << std::get<4>(listaElevi[index]) <<"   ";
		std::cout << (std::get<2>(listaElevi[index]) + std::get<3>(listaElevi[index]) + std::get<4>(listaElevi[index])) / 3.0;
		std::cout << std::endl;
	}
}

int main()
{
	std::vector<std::tuple<std::string, std::string, int, int, int>> listaElevi;
	int nrPersoane = 0;
	CitireDate(listaElevi,nrPersoane);
	SortareDescrescator(listaElevi, nrPersoane);
	AfisareVector(listaElevi, nrPersoane);
```

```
	return 0;
}
```

4. Utilizând tipul de date **std::pair<float, float>** să se citesca coordonatele a doua puncte p_1 şi p_2 din sistemul catezian XOY şi să rezolve următoarele cerinte:

   a) Să se verifice dacă cele două puncte se află în acelaşi cadran (0.5 p)

   b) Să se calculeze distanţa euclidiană dintre cele două puncte (0.5 p)

   c) Să se calculeze unghiul pe care îl face dreapta ce conţine segmentul p_1p_2 cu axa Ox (1 p)

   Folosiţi funcţii pentru a rezolva cerinţele.

```
#include <iostream>
#include <math.h>

//Complexitatea de timp este de O(1)

void CitireDate(std::pair<float,float>&p1,std::pair<float,float>&p2)
{
	std::cin >> p1.first >> p1.second;
	std::cin >> p2.first >> p2.second;
}

bool VerificareAcelasiCadran(std::pair<float,float>p1,std::pair<float,float>p2)
{
	if ((p1.first > 0) && (p2.first > 0))
	{
		if ((p1.second > 0) && (p2.second > 0))
			return 1;
		else if ((p1.second < 0) && (p2.second < 0))
			return 1;
	}
	else if ((p1.first < 0) && (p2.first < 0))
	{
		if ((p1.second > 0) && (p2.second > 0))
			return 1;
		else if ((p1.second < 0) && (p2.second < 0))
			return 1;
	}
	return 0;
}

float DistantaEuclidiana(std::pair<float, float>p1, std::pair<float, float>p2)
{
	return std::sqrt((p2.first-p1.first)*(p2.first-p1.first)+(p2.second-p1.second)*(p2.second-p1.second));
}
```

```
double AflareUnghi(std::pair<float,float>p1,std::pair<float,float>p2)
{
	return atan((double)((p2.first - p1.first) / (p2.second - p1.second)));
}

int main()
{
	std::pair<float,float>p1,p2;
	CitireDate(p1, p2);
	if (VerificareAcelasiCadran(p1, p2))
		std::cout << "Punctele sunt in acelasi cadran!" << std::endl;
	else
		std::cout << "Punctele nu sunt in acelasi cadran!" << std::endl;
	std::cout << "Distanta euclidiana este " << DistantaEuclidiana(p1, p2) << std::endl;
	std::cout << "Unghiul format dintre dreapta ce contine segmentul p1p2 si axa Ox este de " <<
AflareUnghi(p1, p2) << std::endl;
	return 0;
}
```

5. Se citeşte un număr natural nr şi nr medii (numere reale cu două zecimale cu valori cuprinse între 1 şi 10). Notele se stochează într-un container de tip **std::vector<float> note**. Să se afişeze o statistică a mediilor astfel: numărul de medii cuprinse în intervalul $[1, 5)$, numărul de medii din intervalul $[5, 5.50)$, numărul de medii cuprinse în intervalul $[5.50, 6.50)$, ..., numărul de medii cuprinse în intervalul $[9.50, 10]$. (1.5 p)

```
#include<iostream>
#include <vector>
#include <math.h>

//Complexitatea de timp este O(n)

void CitireDate(std::vector<float>& note,int&nr)
{
	std::cin >> nr;
	float nota;
	for (int index = 0; index < nr; index++)
	{
		std::cin >> nota;
		note.push_back(nota);
	}
}

void AflareStatistica(std::vector<float> note, int nr)
{
  int statistica[11];
```

```
	for (int index = 4; index <= 10; index++)
		statistica[index] = 0;
	for (int index = 0; index < nr; index++)
	{
		if (note[index] < 5)
			statistica[4]++;
		else
		{
			statistica[(int)std::round(note[index])]++;
		}
	}
	for (int index = 4; index <= 10; index++)
	{
		if (index == 4)
			std::cout << "Numarul de medii cuprinse in intervalul [1,5) este " << statistica[index]
<< std::endl;
		else if (index==5)
			std::cout << "Numarul de medii cuprinse in intervalul [5,5.5) este " << statistica[index]
<< std::endl;
		else if (index == 10)
			std::cout << "Numarul de medii cuprince in intervalul [9.50,10] este " <<
statistica[index] << std::endl;
		else
			std::cout << "Numarul de medii cuprinse in intervalul [" << index+0.50 << "," <<
index + 1.50 << ") este " << statistica[index] << std::endl;
	}
}

int main()
{
	std::vector<float> note;
	int nr;
	CitireDate(note, nr);
	AflareStatistica(note,nr);
	return 0;
}
```

6. Cristina şi George vor pleca într-o excursie şi trebuie să cumpere alimente. Fiecare a alcătuit o lista de cumpărături pe care sunt numele de produse şi cantitatea din fiecare produs, pe care o doresc. George foloseşte o funcţie, care preia cele două liste şi returnează o a treia, ce conţine toate produsele din ambele liste, împreună cu cantităţile corespunzătoare. Produsele care apar pe ambele liste vor apărea o singură dată în lista finală, cu cea mai mare cantitate dintre cele de pe cele două liste. Scrieţi această funcţie şi afişaţi frumos lista finală. **Observaţie**: folosiţi vectori pentru memorarea listelor de cumpărături. Produsele vor avea nume de tip **std::string**.(2p)

```
#include <iostream>
```

```cpp
#include <string.h>
#include <algorithm>

//Complexitatea de timp este O(nlogn)

struct listaDeCumparaturi
{
	std::string numeProdus;
	int cantitateProdus;
};

void CitireLista(listaDeCumparaturi*& lista, int& nrProduse)
{
	std::cin >> nrProduse;
	lista = new listaDeCumparaturi[nrProduse];
	for (int index = 0; index < nrProduse; index++)
	{
		std::cin >> lista[index].numeProdus >> lista[index].cantitateProdus;
	}
}

struct Comparator
{
	bool operator()(listaDeCumparaturi produs1, listaDeCumparaturi produs2)
	{
		if (produs1.numeProdus.compare(produs2.numeProdus) >= 0)
			return 0;
		return 1;
	}
};

void InterclasareListe(listaDeCumparaturi* lista1, int nrProduse1, listaDeCumparaturi* lista2, int
nrProduse2,listaDeCumparaturi*& lista, int& nrProduse3)
{
	int index1 = 0, index2 = 0;
	while ((index1 < nrProduse1) && (index2 < nrProduse2))
	{
		if (lista1[index1].numeProdus.compare(lista2[index2].numeProdus) == 0)
		{
			if (lista1[index1].cantitateProdus >= lista2[index2].cantitateProdus)
			{
				lista[nrProduse3].numeProdus = lista1[index1].numeProdus;
				lista[nrProduse3].cantitateProdus = lista1[index1].cantitateProdus;
			}
			else
			{
				lista[nrProduse3].numeProdus = lista2[index2].numeProdus;
				lista[nrProduse3].cantitateProdus = lista2[index2].cantitateProdus;
			}
			index1++, index2++, nrProduse3++;
		}
```

```
		else if (lista1[index1].numeProdus.compare(lista2[index2].numeProdus) < 0)
		{
			lista[nrProduse3].numeProdus = lista1[index1].numeProdus;
			lista[nrProduse3].cantitateProdus = lista1[index2].cantitateProdus;
			index1++, nrProduse3++;
		}
		else
		{
			lista[nrProduse3].numeProdus = lista2[index2].numeProdus;
			lista[nrProduse3].cantitateProdus = lista2[index2].cantitateProdus;
			index2++, nrProduse3++;
		}
	}
	while (index1 < nrProduse1)
	{
		lista[nrProduse3].numeProdus = lista1[index1].numeProdus;
		lista[nrProduse3].cantitateProdus = lista1[index2].cantitateProdus;
		index1++, nrProduse3++;
	}
	while (index2 < nrProduse2)
	{
		lista[nrProduse3].numeProdus = lista2[index2].numeProdus;
		lista[nrProduse3].cantitateProdus = lista2[index2].cantitateProdus;
		index2++, nrProduse3++;
	}
}

listaDeCumparaturi* AflareLista3(listaDeCumparaturi* lista1, int nrProduse1, listaDeCumparaturi* lista2,int
nrProduse2,int& nrProduse3)
{
	listaDeCumparaturi* lista;
	std::sort(lista1, lista1 + nrProduse1, Comparator());
	std::sort(lista2, lista2 + nrProduse2, Comparator());
	lista = new listaDeCumparaturi[nrProduse1 + nrProduse2];
	InterclasareListe(lista1, nrProduse1, lista2, nrProduse2, lista, nrProduse3);
	return lista;
}

void AfisareLista(listaDeCumparaturi*lista, int nrProduse)
{
	std::cout << std::endl;
	for (int index = 0; index < nrProduse; index++)
	{
		std::cout << lista[index].numeProdus << " " << lista[index].cantitateProdus;
		std::cout << std::endl;
	}
}

int main()
{
	listaDeCumparaturi* lista1, * lista2,*lista3;
```

```
	int nrProduse1, nrProduse2,nrProduse3=0;
	CitireLista(lista1, nrProduse1);
	CitireLista(lista2, nrProduse2);
	lista3=AflareLista3(lista1, nrProduse1, lista2, nrProduse2,nrProduse3);
	AfisareLista(lista3, nrProduse3);
	delete[] lista1;
	delete[] lista2;
	delete[] lista3;
	return 0;
}
```

7. Alex are un joc cu blocuri din lemn de forma unor paralelelipipede dreptunghice cu baza pătrată de dimensiune fixă. Blocurile au înălţimi diferite. Alex aranjază aceste blocuri în pătrate concentrice.

   a) Să se verifice dacă un astfel de aranjament, reprezentat printr-o matrice, are formă piramidală, adică: toate blocurile dintr-un pătrat sunt mai mici decât cele aflate în regiunea din interiorul acestui pătrat (1p).

   b) Dacă pe baza fiecărui pătrat este gravat un număr pozitiv (ce reprezintă înălţimea blocului în cm), să se scrie o funcţie care calculează pentru un pătrat k înălţimea medie a cuburilor de pe acel pătrat. (1p).

```
#include <iostream>
#include <fstream>

// Complexitatea de timp este O(n^2/2)

void AlocareMatrice(float**& matrice, int dim)
{
	matrice = new float*[dim];
	for (int index = 0; index < dim; index++)
		matrice[index] = new float[dim];
}

void CitireDate(float**& matrice, int& dim)
{
	std::ifstream fin("Fisier.in");
	fin >> dim;
	AlocareMatrice(matrice, dim);
	for (int index = 0; index < dim; index++)
		for (int jndex = 0; jndex < dim; jndex++)
			fin >> matrice[index][jndex];
	fin.close();
}

bool VerificarePiramida(float** matrice, int dim)
{
	float maxim = matrice[0][0];
```

```
	int index = 0,maxPatrat;
	while (index <=dim)
	{
		maxPatrat = matrice[index][index];
		for (int coloana = index; coloana < dim; coloana++)
		{
			if (matrice[index][coloana] > maxPatrat)
				maxPatrat = matrice[index][coloana];
			if (matrice[dim- 1][coloana] > maxPatrat)
				maxPatrat = matrice[dim- 1][coloana];
		}
		for (int linie = index + 1; linie < dim-1; linie++)
		{
			if (matrice[linie][index] > maxPatrat)
				maxPatrat = matrice[linie][index];
			if (matrice[linie][dim - 1] > maxPatrat)
				maxPatrat = matrice[linie][dim - 1];
		}
		if (maxPatrat > maxim)
			maxim = maxPatrat;
		else
			return false;
		index++;
		dim--;
	}
	return true;
}

float AflareInaltimeMedie(float** matrice,int dim,int k)
{
	float suma = 0;
	int nr = 0;
	for (int coloana = k-1; coloana <= dim - k; coloana++)
	{
		if (matrice[k-1][coloana] > 0)
		{
			suma = suma + matrice[k-1][coloana];
			nr++;
		}
		if (matrice[dim-k][coloana] > 0)
		{
			suma = suma + matrice[dim-k][coloana];
			nr++;
		}
	}
	for (int linie = k; linie <= dim - k-1; linie++)
	{
		if (matrice[linie][k-1] > 0)
		{
			suma = suma + matrice[linie][k-1];
			nr++;
```

```
                }
                if (matrice[linie][dim - k] > 0)
                {
                        suma = suma + matrice[linie][dim - k];
                        nr++;
                }
        }
        return suma / nr;
}

void DealocareDinamica(float**& matrice, int dim)
{
        for (int index = 0; index < dim; index++)
                delete[] matrice[index];
        delete[] matrice;
}

int main()
{
        float** matrice;
        int dim,k;
        CitireDate(matrice, dim);
        if (VerificarePiramida(matrice, dim))
                std::cout << "Constructie piramidala!" << std::endl;
        else
                std::cout << "Constructie care nu e piramidala!" << std::endl;
        std::cout << "Introduceti patratul: " << std::endl;
        std::cin >> k;
        std::cout << "Inaltimea medie din patratul " << k << " este " << AflareInaltimeMedie(matrice, dim, k);
        DealocareDinamica(matrice, dim);
        return 0;
}
```

8. Se consideră o matrice $matr$ cu $nrows$ linii şi $ncols$ coloane, cu $ncols < 10$, ale cărei elemente sunt numere naturale formate dintr-o singură cifră. Se consideră că fiecare coloană col reprezintă un număr în baza $col + 2$. Să se scrie o funcţie care plasează numerele transformate în baza zece într-un vector $numbers$ şi returnează **true**, dacă matricea a fost validă şi **false** altfel. În cazul în care matricea a fost validă să se afişeze acest vector de numere. Matricea este validă dacă toate elementele de pe coloana col sunt numere naturale din intervalul $[0, col + 2)$. (2p)

```
#include <iostream>
#include <fstream>

// Complexitatea de timp este O(nrows*ncols)

void AlocareDinamica(int**&matrice,int nrows,int ncols)
{
```

```
	matrice = new int* [nrows];
	for (int index = 0; index < nrows; index++)
		matrice[index] = new int[ncols];
}

void CitireDate(int**& matrice, int& nrows, int& ncols)
{
	std::ifstream fin("Fisier.in");
	fin >> nrows >> ncols;
	AlocareDinamica(matrice, nrows, ncols);
	for (int index = 0; index < nrows; index++)
		for (int jndex = 0; jndex < ncols; jndex++)
			fin >> matrice[index][jndex];
	fin.close();
}

bool VerificareMatrice(int** matrice,int nrows,int ncols)
{
	for(int index=0;index<nrows;index++)
	 for(int jndex=0;jndex<ncols;jndex++)
		if(!((0<=matrice[index][jndex])&&(matrice[index][jndex]<jndex+2)))
			return false;
	return true;
}

void AfisareVector(int*numbers, int dim)
{
	for (int index = 0; index < dim; index++)
		std::cout << numbers[index] << ' ';
}

void AflareNumbers(int** matrice,int nrows,int ncols)
{
	int* numbers = new int[ncols];
	int numar;
	for (int jndex = 0;jndex<ncols;jndex++)
	{
		numar = 0;
		for (int index = nrows - 1; index >= 0; index--)
			numar = numar + matrice[index][jndex]*pow(jndex + 2, nrows - index-1);
		numbers[jndex] = numar;
	}
	AfisareVector(numbers, ncols);
	delete[] numbers;
}

void DealocareDinamica(int**& matrice, int nrows)
{
	for (int index = 0; index < nrows; index++)
		delete[] matrice[index];
	delete[] matrice;
```

```
}

int main()
{
	int** matrice;
	int nrows, ncols;
	CitireDate(matrice, nrows, ncols);
	if (VerificareMatrice(matrice, nrows, ncols))
	{
		std::cout << "Matricea este valida!" << std::endl;
		AflareNumbers(matrice,nrows,ncols);
	}
	else
		std::cout << "Matricea nu este valida!" << std::endl;
	DealocareDinamica(matrice, nrows);
	return 0;
}
```

9. Fiind dată o matrice pătratică de dimensiuni $height \times width$, să se verifice pentru care dintre cele 4 zone determinate de diagonala principală şi de cea secundară suma elementelor este maximă.

   Exemplu: În figura de mai jos este reprezentată o matrice de dimensiuni 5×5, împreună cu zonele delimitate de cele 2 diagonale. Se obesrvă că suma maximă se obţine în zona estică, marcată cu albastru. (1p)

```
#include <iostream>
#include <fstream>
#include <algorithm>

// Complexitatea de timp este O(height*width)

void AlocareDinamica(int**& matrice, int height, int width)
{
	matrice = new int* [height];
	for (int index = 0; index < height; index++)
		matrice[index] = new int[width];
}

void DealocareDinamica(int**& matrice,int height)
{
	for (int index = 0; index < height; index++)
		delete[] matrice[index];
	delete[] matrice;
}

void CitireDate(int**& matrice,int&height,int&width)
{
```

```
	std::ifstream fin("Fisier.in");
	fin >> height >> width;
	AlocareDinamica(matrice, height, width);
	for (int index = 0; index < height; index++)
		for (int jndex = 0; jndex < width; jndex++)
			fin >> matrice[index][jndex];
	fin.close();
}

void AflareMaximZone(int ZonaN,int ZonaE,int ZonaS,int ZonaV)
{
	int maxim = std::max(ZonaN, std::max(ZonaE, std::max(ZonaS, ZonaV)));
	if (maxim == ZonaN)
		std::cout << "Zona de nord are suma maxima." << std::endl;
	if (maxim == ZonaE)
		std::cout << "Zona de est are suma maxima." << std::endl;
	if (maxim == ZonaS)
		std::cout << "Zona de sud are suma maxima." << std::endl;
	if (maxim == ZonaV)
		std::cout << "Zona de vest are suma maxima." << std::endl;
}

void AflareSumeZone(int** matrice,int height,int width)
{
	int ZonaN = 0, ZonaE = 0, ZonaS = 0, ZonaV = 0;
	for(int index=0;index<height;index++)
		for (int jndex = 0; jndex < width; jndex++)
		{
			if ((index < jndex) && (index + jndex < width - 1))
				ZonaN = ZonaN + matrice[index][jndex];
			else if ((index < jndex) && (index + jndex > width - 1))
				ZonaE = ZonaE + matrice[index][jndex];
			else if ((index > jndex) && (index + jndex > width - 1))
				ZonaS = ZonaS + matrice[index][jndex];
			else if ((index > jndex) && (index + jndex < width - 1))
				ZonaV = ZonaV + matrice[index][jndex];
		  }
	AflareMaximZone(ZonaN,ZonaE,ZonaS,ZonaV);
}

int main()
{
	int** matrice;
	int height, width;
	CitireDate(matrice, height, width);
	AflareSumeZone(matrice,height,width);
	DealocareDinamica(matrice, height);
	return 0;
}
```