Tipuri de date structurate

Problema 1

Să se citească un șir *A* de la tastatură, citirea șirului se termina la introducerea valorii 0. Să se construiască și să se tipărească șirul *B* de perechi (*element*, *frecvență*) care memorează frecvența de apariție a fiecărui element din șirul *A* (*element* reprezintă elementul din șirul *A*, iar *frecvență* reprezintă frecvența de apariție a acestuia). Să se elimine din șirul A cele mai frecvente 2 numere prime (în cazul în care mai mult de două numere au aceeași frecvență, se vor elimina aleator două dintre ele), apoi să se tipărească șirul rezultat pe ecran.

Exemple

- Pentru șirul A={1,2,3,4,5,6,1,2,3,1,2,3,4}
  Se va tipări B={(2,3),(3,3),(5,1)}
  Pe ecran se va tipări șirul A modificat: 1,4,5,6,1,1,4
- Pentru șirul A={1,2,4,6,8,10}
  Se va tipări B={(2,1)}
  Pe ecran se va tipări: 1,4,6,8,10

Se vor scrie subprograme pentru:

- citirea șirului *A*;
- determinare frecvență de apariție în șir pentru un număr dat;
- construirea șirului de frecvențe pentru toate numerele prime din șirul *A*;
- sortarea șirului *B* descrescător după frecvențe;
- verificarea dacă un număr este sau nu prim;
- eliminarea din șirul *A* a tuturor aparițiilor unei valori *v* date;
- afișarea șirului *B*;
- afișarea șirului *A*

Rezolvare

```
#include <iostream>
using namespace std;
const int MAX_ELEMENTE = 100;

//Sirul de numere
typedef struct {
    int elems[MAX_ELEMENTE];
    int lg;
} Sir;

//pereche numar, numar de aparitii
typedef struct {
    int nr;
    int nrAparitii;
} Frecventa;

//sirul de frecvente
typedef struct {
    Frecventa elems[MAX_ELEMENTE];
    int lg;
} SirFrecvente;
```

```
/**
 * Citeste un sir de numere de la tastatura
 * returneaza sirul citit
 */
Sir citesteNumere() {
    Sir rez;
    rez.lg = 0;
    int n;
    do {
        cout << "Numarul:";
        cin >> n;
        if (n) {
            rez.elems[rez.lg] = n;
            rez.lg++;
        }
    } while (n);
    return rez;
}

/**
 * Tipareste sirul in consola
 * nrs - sir de numere
 */
void tiparesteSir(Sir nrs) {
    for (int i = 0; i < nrs.lg; i++) {
        cout << nrs.elems[i] << ",";
    }
    cout << "\n";
}

/**
 * Tipareste sirul cu frecvente
 * nrs - sir de numere
 */
void tiparesteFrecvente(SirFrecvente nrs) {
    for (int i = 0; i < nrs.lg; i++)
        cout << "(" << nrs.elems[i].nr << "," << nrs.elems[i].nrAparitii
                << "),";
    cout << "\n";
}

/**
 *  calculeaza numarul de aparitii in sir pentru numarul dat
 *  returneaza perechea numar, numar aparitii
 */
Frecventa calculeazaFrecventa(Sir nrs, int nr) {
    Frecventa rez;
    rez.nr = nr;
    rez.nrAparitii = 0;
    for (int i = 0; i < nrs.lg; i++) {
        if (nrs.elems[i] == nr) {
            rez.nrAparitii++;
        }
    }
    return rez;
}
```

```
/**
 * Varifica daca un numar e prim
 * returneaza true daca numarul e prim
 */
bool ePrim(int nr) {
    if (nr <= 1) {
        return false;
    } else if (nr <= 3) {
        return true;
    } else if (nr % 2 == 0 || nr % 3 == 0) {
        return false;
    }
    int i = 5;
    while (i * i <= nr) {
        if (nr % i == 0 || nr % (i + 2) == 0) {
            return 0;
        }
        i += 6;
    }
    return 1;
}
/**
 * Sorteaza sirul descrescator dupa numarul de aparitii
 * modifica sirul primit ca parametru (sirul devine sortat)
 */
void sorteazaDupaFrecvente(SirFrecvente& frecv) {
    for (int i = 0; i < frecv.lg - 1; i++)
        for (int j = i + 1; j < frecv.lg; j++)
            if (frecv.elems[i].nrAparitii < frecv.elems[j].nrAparitii) {
                Frecventa temp = frecv.elems[i];
                frecv.elems[i] = frecv.elems[j];
                frecv.elems[j] = temp;
            }
}

/**
 * Verifica daca pentru numarul dat avem deja frecventa in sirul fvs
 * return true daca in sirul fvs avem deja un element pentru numarl nr
 */
bool apare(SirFrecvente fvs, int nr) {
    for (int i = 0; i < fvs.lg; i++) {
        if (fvs.elems[i].nr == nr) {
            return true;
        }
    }
    return false;
}
```

```
/**
 * Sirul de frecente pentru numerele prime
 * returneaza sir de frecvente
 */
SirFrecvente determinaSirFrecventePrime(Sir nrs) {
    SirFrecvente rez;
    rez.lg = 0;
    for (int i = 0; i < nrs.lg; i++) {
        if (!apare(rez, nrs.elems[i]) && ePrim(nrs.elems[i])) {
            rez.elems[rez.lg] = calculeazaFrecventa(nrs, nrs.elems[i]);
            rez.lg++;
        }
    }
    return rez;
}
/**
 * Sterge elementul de pe pozitie data
 * Modifica sirul, elimina elementul de pe pozitia poz
 */
void stergeDePePozitie(Sir& nrs, int poz) {
    for (int i = poz; i < nrs.lg - 1; i++) {
        nrs.elems[i] = nrs.elems[i + 1];
    }
    nrs.lg--;
}
/**
 * Sterge din sir toate aparitiile numarului dat
 * Modifica sirul, elimina toate aparitiile lui nr
 */
void stergeToateAparitiile(Sir& nrs, int nr) {
    int poz = 0;
    while (poz < nrs.lg) {
        if (nrs.elems[poz] == nr) {
            stergeDePePozitie(nrs, poz);
        } else {
            poz++;
        }
    }
}

int main() {
    Sir nrs = citesteNumere();
    tiparesteSir(nrs);

    SirFrecvente frecvente = determinaSirFrecventePrime(nrs);
    sorteazaDupaFrecvente(frecvente);
    tiparesteFrecvente(frecvente);

    stergeToateAparitiile(nrs, frecvente.elems[0].nr);
    if (frecvente.lg > 1) {
         //ma asigur ca am avut cel putin 2 elemente distincte in sir
        stergeToateAparitiile(nrs, frecvente.elems[1].nr);
    }
    tiparesteSir(nrs);
    return 0;
}
```

Exemple

Date de intrare	Rezultat
Numarul:1 Numarul:2 Numarul:3 Numarul:4 Numarul:3 Numarul:0	1,2,3,4,3, (3,2),(2,1), 1,4,
Numarul:4 Numarul:6 Numarul:8 Numarul:10 Numarul:0	4,6,8,10, 4,6,8,10,
Numarul:11 Numarul:6 Numarul:8 Numarul:11 Numarul:10 Numarul:11 Numarul:0	11,6,8,11,10,11, (11,3), 6,8,10,
Numarul:11 Numarul:12 Numarul:13 Numarul:15 Numarul:23 Numarul:23 Numarul:23 Numarul:0	11,12,13,15,23,23,23, (23,3),(13,1),(11,1), 11,12,15,

Problema 2

Se citeşte o matrice $A(n,m)=(a_{ij})_{i=\overline{1,n},j=\overline{1,m}}$ de numere naturale nenule, unde $1 \le n,m \le 100$, $1 \le a_{ij} \le 30000$. Să se scrie un program care elimină din A coloanele j având proprietatea că minimul şi maximul de pe coloana respectivă sunt numere *apropiate*. Se va forma un şir R cu numerele distincte eliminate, în ordine descrescătoare a cifrei lor maxime (numerele cu aceeaşi cifră maximă vor apărea pe poziţii consecutive în şir, fără să conteze ordinea lor). Spunem ca două numere naturale sunt *apropiate* dacă scrierile celor două numere în baza 10 conţin cel puţin 2 cifre distincte comune (Ex.: 1313 şi 33112 SUNT *apropiate*, iar 1231 şi 6333 NU sunt *apropiate*). La sfârşit se cere tipărirea matricei A şi a şirului R. Şirul R se va construi direct ordonat, fără ordonarea ulterioară a acestuia.

Exemplu

Pentru n=4, m=4 şi matricea $A=\begin{pmatrix} 15 & 4 & 15658 \\ 13 & 18 & 24037 \\ 18 & 220 & 13169 \\ 13 & 183 & 3013 \end{pmatrix}$. Se va tipări $A=\begin{pmatrix} 4 & 58 \\ 18 & 37 \\ 20 & 196 \\ 33 & 133 \end{pmatrix}$ şi de ex. R =(182, 1316, 156, 15, 240, 13, 330)

Se vor scrie subprograme pentru:

a). citirea unei matrice $A(n,m)$
b). verificarea dacă două numere sunt *apropiate*
c). verificarea dacă maximul şi minimul de pe o coloană j a matricei A sunt numere *apropiate*
d). eliminarea unei coloane j din matricea $A(n,m)$
e). inserarea unui număr în şirul R (conform cerinţelor problemei)
f). adăugarea elementelor de pe o coloană j din matricea A în şirul R
g). tipărirea unei matrice
h). tipărirea unui şir

```
#include <iostream>
using namespace std;
const int MAX_MATRICE = 100;
const int MAX_ELEMENTE = 100;

//Sirul
typedef struct {
	int elems[MAX_ELEMENTE];
	int lg;
} Sir;

/**
 * Calculeaza vectorul caracteristic al unui numar
 * */
void caracteristic(int a, bool apareCifra[10]) {
	for (int i = 0; i < 10; i++)
		apareCifra[i] = false; //nu apare cifra
	while (a > 0) {
		int ultimaCifra = a % 10;
		apareCifra[ultimaCifra] = true;
		a = a / 10;
	}
}
```

```
/**
 * Cifra maxima a unui numar
 */
int cifraMaxima(int nr) {
	bool apareCifra[10];
	caracteristic(nr, apareCifra);
	int max = 9;
	while (!apareCifra[max]) //ma opresc la prima cifra care apare
		max--;
	return max;
}
/**
 * Adauga elementul in sir
 * Sirul ramane ordonat descrescator dupa cifra maxima
 */
void adaugaSortat(Sir& l, int elem) {
	int cif = cifraMaxima(elem);
	//gasesc pozitia de inserare
	int i = 0;
	while (i < l.lg && cif <= cifraMaxima(l.elems[i])){
		i++;
	    if (elem == l.elems[i]){
	      return; // nu vrem duplicate
	    }
	}

	//inserez pe pozitia i
	for (int j = l.lg; j >= i + 1; j--)
		l.elems[j] = l.elems[j - 1];
	l.elems[i] = elem;
	l.lg++;
}

/**
 * Tipareste sir
 * l - sirul de numere
 */
void tiparesteSir(Sir l) {
	cout << "\n";
	for (int i = 0; i < l.lg; i++) {
		cout << l.elems[i] << ",";
	}
}

//Matrice
typedef struct {
	int elems[MAX_MATRICE][MAX_MATRICE];
	int linii;
	int coloane;
} Matrice;
// Citeste o matrice
// Out: a - matricea citita
Matrice citesteMat() {
	Matrice a;
	cout << "Nr. linii:";
	cin >> a.linii;
	cout << "Nr. coloane:";
	cin >> a.coloane;
	for (int i = 0; i < a.linii; i++) {
		for (int j = 0; j < a.coloane; j++) {
			cin >> a.elems[i][j];
		}
	}
	return a;
}
```

```
// Tipareste o matrice
// a - matricea
// se tipareste matricea la consola
void tiparesteMat(Matrice a) {
	cout << "\n";
	for (int i = 0; i < a.linii; i++) {
		for (int j = 0; j < a.coloane; j++) {
			cout << a.elems[i][j] << ",";
		}
		cout << "\n";
	}
}

/**
 * Verifica daca a, b sunt apropiate
 * a,b > 0 a,b<300000
 * return true daca a si b au cel putin doua cifre distincte in comun
 */
bool apropiate(int a, int b) {
	bool apareCifreA[10];
	caracteristic(a, apareCifreA);
	bool apareCifreB[10];
	caracteristic(b, apareCifreB);
	int comune = 0;
	for (int i = 0; i < 10; i++) {
		if (apareCifreA[i] && apareCifreB[i]) {
			comune++;
		}
	}
	return comune >= 2;
}

/**
 * Verifica daca minimul si maximul de pe coloana data sunt apropiate
 * return true daca minim si maxim sunt apropiate
 */
bool minMaxApropiate(Matrice a, int coloana) {
	int min = a.elems[0][coloana];
	int max = a.elems[0][coloana];
	for (int i = 1; i < a.linii; i++) {
		if (a.elems[i][coloana] < min) {
			min = a.elems[i][coloana];
		}
		if (a.elems[i][coloana] > max) {
			max = a.elems[i][coloana];
		}
	}
	return apropiate(min, max);
}
/*
 * Elimina din matrice coloana specificata
 * Numerele eliminate se adauga in sirul l
 */
void eliminaColoana(Matrice& a, int coloana, Sir& l) {
	for (int lin = 0; lin < a.linii; lin++) {
		//adauga elementul eliminat in rezultat
		adaugaSortat(l, a.elems[lin][coloana]);
		//mutam elementele spre stanga
		for (int col = coloana; col < a.coloane - 1; col++) {
			a.elems[lin][col] = a.elems[lin][col + 1];
		}
	}
	a.coloane = a.coloane - 1;
}
```

```
int main() {
	Matrice a = citesteMat();
	tiparesteMat(a);

	Sir r;
	r.lg = 0;
	//eliminam coloanele cu min/max apropiate
	for (int col = a.coloane - 1; col >= 0; col--) {
		if (minMaxApropiate(a, col)) {
			eliminaColoana(a, col, r);
		}
	}

	tiparesteMat(a);
	tiparesteSir(r);
	return 0;
}
```