**Cerinta laborator 3 – Functii friend si supradefinirea operatorilor**

Fie tipul de date fractie cu atribute a si b – numarator si, respectiv, numitor si metodele si functiile prietene definite mai jos.

Intr-un proiect nou, adaugati fisierele fractie.h si fractie.cpp, pe langa fisierul generat automat main.cpp.

In fractie.h se gaseste interfata clasei (vezi mai jos). In fractie.cpp se vor implementa toate functiile specificate in fisierul header.

In main.cpp se vor declara obiecte de tip fractie si se vor testa toate metodele implementate.

Veti incarca pe Moodle cele 3 fisiere (fractie.h, fractie.cpp, main.cpp).

//fractie.h

class fractie{

int a; //numarator

int b; //numitor

public:

fractie(int =0,int =0); // constructor cu parametri – valori default

fractie(const fractie&); // constructor de copiere

fractie & operator=(const fractie &); /\*se intoarce referinta la obiectul modificat pt a putea face operatii de genul : int a,b,c,d ; a=(b=(c=(d=4))) ; asociativitate la dreapta \*/

~fractie(); /\*OBS: constr de copiere, op= si destr se genereaza automat si functioneaza corect; implementarea lor va fi facuta doar in scop didactic \*/

double getValoare(); //cat face a/b

fractie getInv(); //returnez fractia b/a

void setData(int,int); /\*modifica valoarile lui a si b cu ce se gaseste in parametrii\*/

float getA(); //returneaza numaratorul

float getB(); //returneaza numitorul

void afisare() const;

friend const fractie operator +(const fractie &, const fractie&); /\* supradefinire operator adunare; operanzii sunt declarati const ca sa nu poata fi modificati; se returneaza un obiect constant ca sa nu pot sa fac operatii de tipul (a+b) = fractie(1,2); caz in care - atribuirea ar putea sa aiba loc, dar am pierde rezultatul \*/

friend const fractie operator -( const fractie&, const fractie& ); // a-b

friend const fractie operator -(const fractie&); //ex d=- e; minus forma unara

friend const fractie operator \*( const fractie&, const fractie&);

friend const fractie operator /( const fractie&, const fractie&);

fractie& operator +=( const fractie& ); /\* supradefinire operator incrementare cu o valoare; se intoarce referinta la fractie pt a putea face operatii ca : int m; (m+=5)+=3 ;\*/

// implementarea poate sa fie \*this=\*this+a; return \*this;

fractie& operator -=( const fractie&);

fractie& operator \*=( const fractie&);

fractie& operator /=( const fractie&);

friend bool operator ==(const fractie &, const fractie &);/\*supradefinire operator de egalitate \*/

friend bool operator !=(const fractie& x, const fractie &);/\* supradefinire operator diferit; pot sa folosesc in implementare operatorul == implementat anterior {return(!(y==x));} \*/

friend bool operator >(const fractie& x, const fractie &);

friend bool operator <(const fractie& x, const fractie &);

friend bool operator >=(const fractie& x, const fractie &);

friend bool operator <=(const fractie& x, const fractie &);

};