Programabilni uređaji i objektno orjentisano programiranje

Računske vježbe 9

1. Realizovati klasu **kompleks** koja predstavlja kompleksne brojeve i koja kao podatke članove sadrži realni i imaginarni dio kompleksnog broja (realni brojevi). Klasa kompleks treba da sadrži statičku promjenljivu, koja će voditi računa o najmanjoj apsolutnoj vrijednosti kompleksnog broja, koja se ikada desila tokom izvršavanju programa.

   Nakon toga realizovati klasu **kompleksN**, koja kao podatke članove sadrži niz kompleksnih brojeva (objekata klase kompleks) i njegovu dužinu (cio broj). Kod klase kompleksN Izvršiti preklapanje operatora +, += kao i operatora za prefiksno i postfiksno inkrementiranje. Pri inkrementiranju, potrebno je realni dio svakog kompleksnog broja u nizu uvećati za 1. Potrebno je relizovati i operator() koji kao argumente ima dva indeksa (dva cijela broja) a kao rezultat treba da vrati dio niza koji se nalazi između proslijeđena dva indeksa. Pri tome, potrebno je izvršiti provjeru da li su proslijeđeni indeksi ispravno zadati i da li se nalaze u granicama niza.

   Na kraju, realizovati glavni dio programa u kojem ćete kreirati probne objekte klasa kompleks i kompleksN pa zatim na njima testirati sve realizovane funkcije.

```
#include <iostream>
#include <stdlib.h> // treba nam za exit() funkciju
#include <math.h> // treba nam za sqrt() i fabs() funkcije
#include <float.h> // treba nam za konstantu FLT_MAX

using namespace std;


class kompleks
{
    private:
        float real;
        float imag;

    public:
        static float najmanji;
        kompleks(){}
        kompleks(float a, float b):real(a), imag(b)
        {
            if(vratiApsVr()<najmanji)
                najmanji = vratiApsVr();
        }
        // za konverziju float u kompleks
        kompleks(float a):real(a), imag(0)
        {
            if(vratiApsVr()<najmanji)
                najmanji = vratiApsVr();
        }
        ~kompleks(){}
        friend kompleks operator+(kompleks, kompleks);
        float vratiReal() const {return real;}
        float vratiImag() const {return imag;}
        float vratiApsVr(){return sqrt(real*real + imag*imag);}
        void stampaj() const { cout<<real<<(imag > 0 ? " + " : " - ")<<fabs(imag)<<endl;}
};

float kompleks::najmanji = FLT_MAX;

kompleks operator+(kompleks a, kompleks b)
{
    return kompleks(a.real+b.real, a.imag+b.imag);
}
```

```
class kompleksN
{
    private:
        kompleks *niz;
        int brElem;

    public:
        kompleksN(){niz = 0;}
        kompleksN(int a):brElem(a) {niz = new kompleks[brElem];}
        kompleksN(kompleks *, int);
        kompleksN(const kompleksN &);
        ~kompleksN(){delete [] niz; niz = 0;}

        kompleksN & operator=(const kompleksN &);
        friend kompleksN operator+(kompleksN, kompleksN);
        kompleksN & operator+=(const kompleksN &);
        kompleksN & operator++();
        kompleksN operator++(int);
        kompleks & operator[](int i){return niz[i];} // poziv: a.operator[](i)
        kompleksN operator()(int, int); //daje dio niza izmedju zadatih indeksa

        void stampaj() const
        {
            // pozivamo funkciju stampaj klase kompleks
            for(int i=0; i<brElem; i++)
                niz[i].stampaj();
            cout<<endl;
        }
};

kompleksN::kompleksN(kompleks *nizArg, int brElemArg):brElem(brElemArg)
{
    niz = new kompleks[brElem];
    // pozivamo ugradjeni operator dodjele klase kompleks
    // u njoj nema pokazivaza, pa sve ispravno funkcionise
    for(int i=0; i<brElem; i++)
        niz[i] = nizArg[i];
}

kompleksN::kompleksN(const kompleksN &a):brElem(a.brElem)
{
    niz = new kompleks[brElem];
    for(int i=0; i<brElem; i++)
        niz[i] = a.niz[i];
}

kompleksN & kompleksN::operator=(const kompleksN &a)
{
    // ako se razlikuju mem. adrese objekata this i a
    // ako nije u pitanju jedan isti objekat, mozemo obrisati podatke objekta kome dodjeljujemo
nove vrijednosti
    if(this != &a)
    {
        delete [] niz;  niz = new kompleks[a.brElem];
        brElem = a.brElem;
        for(int i=0; i < brElem; i++)
            niz[i] = a.niz[i];
    }
    return *this;
}

kompleksN operator+(kompleksN a, kompleksN b)
{
    if(a.brElem != b.brElem)
    {
        cout<<"Nizovi moraju imati isti broj elemenata"<<endl;
        // prekid izvrsavanja programa
        exit(EXIT_FAILURE);
    }
```

```
    else
    {
        // zbog ovoga je napravljen konstruktor kojim se samo zauzima memorija
        //za prazan niz kompleksnih brojeva
        kompleksN rez(a.brElem);

        // mogli smo preklopiti operator [] tako da vraca referencu na i-ti element
        // u tom slucaju bismo pisali rez[i] = a[i] + b[i] (uraditi za vjezbu)
        for(int i=0; i<a.brElem; i++)
            rez.niz[i] = a.niz[i] + b.niz[i];

        return rez;
    }
}

kompleksN & kompleksN::operator+=(const kompleksN &a)
{
    // ovdje se pozivaju dva vec realizovana operatora klase kompleksN (operator+ i operator=)
    *this = *this + a;
    // kada vracamo promjenljivu (*this) koja vec postoji u main-u i prije poziva funkcije, mozemo
je vratiti preko reference!
    return *this;
}

kompleksN & kompleksN::operator++()
{
    // pravimo pomocni prazan niz kompleksnih brojeva klase kompleksN
    kompleksN pom(brElem);

    // pozvace se konstruktor klase kompleks koji pretvara cio (realan) broj u kompleksni, sa
imaginarnim dijelom 0
    for(int i=0; i<brElem; i++)
        pom.niz[i] = 1;

    return *this += pom;
};

kompleksN kompleksN::operator++(int)
{
    // cuvamo originalnu vrijednost objekta klase kompleksN
    // funkcionise kako treba samo ako smo realizovali konstruktor kopije
    kompleksN pom(*this);

    // imamo realizovan prefiksni operator++ za klasu kompleksN pa ga ovdje pozivamo
    ++(*this);

    // kada vracamo lokalnu promjenljivu funkcije kao rezultat, ne smijemo je vracati preko
reference!
    return pom;
}

kompleksN kompleksN::operator()(int poc, int kraj)
{
    int duzOps = kraj-poc+1;
    if( poc>kraj || duzOps>brElem )
    {
        cout<<"Prekoracili ste dimenzije niza"<<endl;
        exit(EXIT_FAILURE); // prekidamo izvrsavanje programa
    }

    // pozivamo konstruktor koji kreira prazan niz podatak clan objekta klase kompleksN
    kompleksN rez(duzOps);
    for(int i=0; i<duzOps; i++)
        // koristimo preklopljeni operator[] indeksiranja
        // rez[i] vraca referencu na rez.niz[i]
        rez[i] = niz[poc+i];

    return rez;
}
```

```
int main()
{
    kompleks a[] = {kompleks(2,3), kompleks(3,4), kompleks(2,-5)};
    kompleks b[] = {kompleks(-1,3), kompleks(2,1), kompleks(1,3)};
    kompleks c[] = {kompleks(-1,3), kompleks(2,1)};
    kompleks d[] = {kompleks(-1,3), kompleks(2,1), kompleks(1,3), kompleks(4,3), kompleks(-5,2)};

    kompleksN niz1 = kompleksN(a,3);
    cout<<"niz1 = "<<endl;
    niz1.stampaj();

    kompleksN niz2 = kompleksN(b,3);
    cout<<"niz2 = "<<endl;
    niz2.stampaj();

    kompleksN niz3;
    niz3 = niz1 + niz2;
    cout<<"niz1 + niz2 "<<endl;
    niz3.stampaj();

    niz1 += niz2;
    cout<<"niz1 += niz2 "<<endl;
    niz1.stampaj();

    kompleksN pom;
    pom = niz2++;
    cout<<"niz2++ "<<endl;
    pom.stampaj();
    cout<<"niz2"<<endl;
    niz2.stampaj();
    cout<<"++niz2"<<endl;
    // rezultat je referenca, pa se moze dalje koristiti kao bilo koji objekat klase kompleksN
    (++niz2).stampaj();

    cout << "Eksperimentisite malo i sami sa kombinacijama preklopljenih operatora" << endl;
    kompleksN nizMali(c,2);
    cout<<"nizMali = "<<endl;
    nizMali.stampaj();

    kompleksN nizC(c,2);
    nizC.stampaj();
    nizC[1] = kompleks(2,3);
    nizC.stampaj();

    kompleksN nizD(d,5);
    cout<<"Pocetni niz je:"<<endl;
    nizD.stampaj();

    kompleksN nizD1 = nizD(2,4);
    cout<<"Dio niza sa indeksima od 2 do 4 je:"<<endl;
    nizD1.stampaj();

    cout<<"Najmanja apsolutna vrijednost kompleksnog broja u programu je: "<<endl;
    cout<<kompleks::najmanji<<endl<<endl;

    cout<<"niz1 + nizMali se ne mogu sabirati"<<endl;
    niz3 = niz1 + nizMali;

    return 0;
}
```