C++

Napisati abstraktnu klasu **Stablo** koja se sastoji od:

- Polja: **carstvo** tipa DinString, **porodica** tipa DinString, **rod** tipa DinString, polje **lotBroj** tipa int, statičko polje **rbrSadnice** tipa int
- **Konstruktor bez parametara** koji setuje polja na proizvoljne vrednosti
- **Konstruktor sa parametrima** koji setuje polja na prosleđene vrednosti
- **Destruktor**
- **get** i **set** metode za polja
- Virtuelne metode **DinString getTipPloda() const**
- Virtuelne metode **double getPrinos() const**
- Virtuelne metode **DinString getTipStabla() const**
- **Preklopljen operator ispisa**

Napisati klasu **StabloJabuke** koja nasleđuje abstraktnu klasu **Stablo** i sadrži:

- Polje **tipStabla** tipa DinString, polje **starost** tipa int, polje **datumSadnje** tipa DinString, Polje **bojaJabuke** enumerativnog tipa **(CRVENA, ZELENA, ŽUTA)**, statičko polje **brojSadnice** tipa int koje modeluje koliko je do sada sadnica jabuke uzeto (ne I posađeno)
- **Prazan konstruktor** koji postavlja polja starost na 1, datumSadnje na “01.01.2022.”, I boju jabuke na ZELENA I uvećava brojSadnica za 1
- **Konstruktor sa parametrima** (uvećava brojSadnica za 1)
- **Konstruktor kopije** (uvećava brojSadnica za 1)
- **Destruktor** (smanjuje broj sadnica za 1)
- **get** I **set** metode za sva polja
- Metodu **DinString getTipPloda() const** koja vraća koji tip stabla je u pitanju kao I boju jabuke (**ZELENA JABUKA**, **ZUTA JABUKA**, …)
- Metodu **double getPrinos() const** koja vraća prosečan prinos po stablu. Prosečan prinos se računa kao **(starost * 0.8) * koeficijentPrinosaSorte**, koeficijent prinosa sorte se određuje na osnovu boje jabuke, dakle za CRVENE iznosi 24.5, ZELENE 22.9, dok za ŽUTE imaju koeficijent 31.5
- **Preklopljen operator ispisa**
- **Preklopljen operator dodele** koji sve osobine stabla prepisuje stablu kojem se dodeljuje

Napisati klasu **StabloSljive** koja nasleđuje abstraktnu klasu Stablo i sadrži:

- Polje **tipStabla** tipa DinString, polje **starost** tipa int, polje **datumSadnje** tipa DinString, **brojSadnice** tipa int koje modeluje koliko je do sada sadnica šljive uzeto (ne I posađeno), polje **zdravo** tipa bool koje moduluje da li je stablo zdravo ili mu je potreban hemijski tretman
- **Prazan konstruktor** koji postavlja polja starost na 1, datumSadnje na “01.01.2022.”, I postaviti polje **zdravo** na **true** I uvećava brojSadnica za 1
- **Konstruktor sa parametrima** (uvećava brojSadnica za 1)
- **Konstruktor kopije** (uvećava brojSadnica za 1)
- **Destruktor** (smanjuje broj sadnica za 1)
- **get** i **set** metode za sva polja

- Metodu **DinString getTipPloda() const** koja vraća koji tip stabla je u pitanju
- Metodu **double getPrinos() const** koja vraća prosečan prinos po stablu. Prosečan prinos se računa kao **(starost * 0.65) * koeficijentPrinosaSorte**, koeficijent prinosa sorte se određuje na osnovu generacije sadnice, npr prvu generaciju sadnica čine prvih uzetih 3 šljive i njihov koeficijent je 12.45, dok se za svaku sledeću generaciju koeficijent smanjuje za 3%
- **Preklopljen operator ispisa**
- Metodu **bool hemijskiTretman()** koja nad stablom vrši hemijski tretman, ako je stablo zdravo metoda vraća **false**, ako stablo nije zdravo polje **zdravo** postavlja na **true** i metoda vraća **true** za uspešan hemijski tretman

Napisati generičku klasu **Vocnjak** koja se sastoji:

- polje **sadrzajVocnjaka** koje modeluje koja se stabla jabuka ili šljiva nalaze u voćnjaku tipa **List<Stablo *>**, polje **godinaEksploatacije** tipa int koja u sebi čuva starost najstarijeg stabla u voćnjaku
- **Konstruktor bez parametara**
- **Konstruktor sa parametrima**
- **Konstruktor kopije**
- Metodu **bool dodajStablo (Stablo &)** koja dodaje u voćnjak stabla pri tome da dva stabla sa istim **lotBrojem** ne mogu postojati u voćnjaku, ako se pokuša dodati isto stablo metoda ne menja listu I vraća **false**, u suprotnom ako je dodavanje na kraj liste uspešno vraća **true**
- Slobodnu funkciju **double prosecniPrinos(const Vocnjak &)** koja računa prosečni prinos u voćnjaku tako što sumu prinosa stabala podeli sa brojem stabala u voćnjaku
- Metodu **void sortiraj(const DinString &nacinSortiranja)** koja uređuje redosled stabala na spisku na osnovu polja **lotBroj** I prosleđenog parametra **nacinSortiranja**. Za sve ostale slučajeve pogrešno prosleđenog parametra **nacinSortiranja** listu ne uređivati.
- Metodu **Stablo* pronadji(const int lotBrojStabla)** koje vraća stablo u listi na osnovu prosleđenog parametra **lotBrojStabla** koje predstavlja unikatni broj stabla, u suprotnom vraća **NULL**
- Metodu **int prebroj(const DinString &ts) const** koja prebrojava koliko puta se traženi tip stabla pojavljuje u voćnjaku
- **Preklopljen operator ispisa**

Napisati kratak test program I istestirati sve metode I konstruktore.

REŠENJE

STABLO.hpp

```
#ifndef STABLO_HPP_INCLUDED
#define STABLO_HPP_INCLUDED

#include "dinstring.hpp"
#include "list.hpp"

class Stablo {
    protected:
        DinString carstvo;
        DinString porodica;
        DinString rod;
        int lotBroj;
        static int rbrSadnice;
    public:
        Stablo() : carstvo(""), porodica(""), rod(""), lotBroj(0) {
                rbrSadnice++;
        }

        Stablo(const DinString &carstvo, const DinString &porodica,
               const DinString &rod, const int lotBroj) : carstvo(carstvo),
               porodica(porodica), rod(rod), lotBroj(lotBroj) {
               rbrSadnice++;
        }

        ~Stablo() { rbrSadnice--; }

        DinString getCarstvo() const {
            return carstvo;
        }
        DinString getPorodica() const {
            return porodica;
        }
        DinString getRod() const {
            return rod;
        }
        int getLotBroj() const {
            return lotBroj;
        }
        int getRbrSadnice() const {
            return rbrSadnice;
        }

        void setCarstvo(const DinString &carstvo) {
             this -> carstvo = carstvo;
        }
        void setPorodica(const DinString &porodica) {
             this -> porodica = porodica;
        }
        void setRod(const DinString &rod) {
             this -> rod = rod;
        }
        void setLotBroj(const int lotBroj) {
             this -> lotBroj = lotBroj;
        }
```

```
        virtual DinString getTipPloda() const = 0;
        virtual double getPrinos() const = 0;
        virtual DinString getTipStabla() const = 0;

        virtual void prevariOstream(ostream& os) const = 0;

        friend ostream& operator<<(ostream& os, const Stablo& stablo) {
            stablo.prevariOstream(os);
            return os;
        }
};

#endif // STABLO_HPP_INCLUDED
```

STABLOJABUKE.hpp

```
#ifndef STABLOJABUKE_HPP_INCLUDED
#define STABLOJABUKE_HPP_INCLUDED

#include "Stablo.hpp"

enum BojaJabuke {CRVENA, ZELENA, ZUTA};

class StabloJabuke : public Stablo {
    private:
        DinString tipStabla;
        int starost;
        DinString datumSadnje;
        BojaJabuke bojaJabuke;
        static int brojSadnice;
    public:
        StabloJabuke() : Stablo(), tipStabla("JABUKA"), starost(1),
                         datumSadnje("01.01.2022."), bojaJabuke(ZELENA) {
               brojSadnice++;
        }

        StabloJabuke(const DinString &carstvo, const DinString &porodica,
                     const DinString &rod, const int lotBroj,
                     const DinString &tipStabla, const int starost,
                     const DinString &datumSadnje, const BojaJabuke bojaJabuke)
                     : Stablo(carstvo, porodica, rod, lotBroj),
                       tipStabla(tipStabla), starost(starost),
                       datumSadnje(datumSadnje), bojaJabuke(bojaJabuke) {
                brojSadnice++;
        }

        StabloJabuke(const StabloJabuke &s) : Stablo(s.carstvo, s.porodica,
                     s.rod, s.lotBroj), tipStabla(s.tipStabla),
                     starost(s.starost), datumSadnje(s.datumSadnje),
                     bojaJabuke(s.bojaJabuke) { brojSadnice++; }

        ~StabloJabuke() { brojSadnice--; }
```

```
DinString getTipStabla() const {
	return tipStabla;
}
int getStarost() const {
	return starost;
}
DinString getDatumSadnje() const {
	return datumSadnje;
}
BojaJabuke getBojaJabuke() const {
	return bojaJabuke;
}
int getBrojSadnice() const {
	return brojSadnice;
}

void setTipStabla(const DinString &tipStabla) {
	this -> tipStabla = tipStabla;
}
void setStarost(const int starost) {
	this -> starost = starost;
}
void setDatumSadnje(const DinString &datumSadnje) {
	this -> datumSadnje = datumSadnje;
}
void setBojaJabuke(const BojaJabuke bojaJabuke) {
	this -> bojaJabuke = bojaJabuke;
}

DinString getTipPloda() const {
    DinString boja;

    if(bojaJabuke == ZELENA)
        boja = "ZELENA";
    else if(bojaJabuke == ZUTA)
        boja = "ZUTA";
    else
        boja = "CRVENA";

    return boja + " " + tipStabla;
}

double getPrinos() const {
    double koeficijentPrinosaSorte = 0.0;

    if(bojaJabuke == ZELENA)
        koeficijentPrinosaSorte = 22.9;
    else if(bojaJabuke == ZUTA)
        koeficijentPrinosaSorte = 31.5;
    else
        koeficijentPrinosaSorte = 24.5;

    return (starost * 0.8) * koeficijentPrinosaSorte;
}
```

```
        StabloJabuke& operator=(const StabloJabuke &s) {
            carstvo     = s.carstvo;
            porodica    = s.porodica;
            rod         = s.rod;
            lotBroj     = s.lotBroj;
            tipStabla   = s.tipStabla;
            starost     = s.starost;
            datumSadnje = s.datumSadnje;
            bojaJabuke  = s.bojaJabuke;

            return *this;
        }

        friend ostream& operator<<(ostream &out, const StabloJabuke &s) {
            out << "---------------- STABLO JABUKE ------------------" << endl;
            out << "\tCarstvo: "    << s.carstvo << endl;
            out << "\tPorodica: "   << s.porodica << endl;
            out << "\tRod: "        << s.rod << endl;
            out << "\tLot. broj: "  << s.lotBroj << endl;
            out << "\tTip stabla: " << s.tipStabla << endl;
            out << "\tStarost: "    << s.starost << endl;
            out << "\tDatum sadnje: " << s.datumSadnje << endl;
            out << "\tBoja jabuke: " <<
                   (s.bojaJabuke == ZELENA ? "ZELENA" :
                   (s.bojaJabuke == ZUTA ? "ZUTA" : "CRVENA")) << endl << endl;

            return out;
        }

        void prevariOstream(ostream &out) const {
            out << "---------------- STABLO JABUKE ------------------" << endl;
            out << "\tCarstvo: " << carstvo << endl;
            out << "\tPorodica: " << porodica << endl;
            out << "\tRod: " << rod << endl;
            out << "\tLot. broj: " << lotBroj << endl;
            out << "\tTip stabla: " << tipStabla << endl;
            out << "\tStarost: " << starost << endl;
            out << "\tDatum sadnje: " << datumSadnje << endl;
            out << "\tBoja jabuke: " <<
                   (bojaJabuke == ZELENA ? "ZELENA" :
                   (bojaJabuke == ZUTA ? "ZUTA" : "CRVENA")) << endl << endl;
        }

};

#endif // STABLOJABUKE_HPP_INCLUDED
```

STABLOSLJIVE.hpp

```
#ifndef STABLOSLJIVE_HPP_INCLUDED
#define STABLOSLJIVE_HPP_INCLUDED

#include "Stablo.hpp"

class StabloSljive : public Stablo {
        private:
        DinString tipStabla;
        int starost;
        DinString datumSadnje;
        static int brojSadnice;
        bool zdravo;
    public:
        StabloSljive() : Stablo(), tipStabla("SLJIVA"), starost(1),
                         datumSadnje("01.01.2022."), zdravo(true) {
              brojSadnice++;
        }

        StabloSljive(const DinString &carstvo, const DinString &porodica,
                     const DinString &rod, const int lotBroj,
                     const DinString &tipStabla, const int starost,
                     const DinString &datumSadnje, const bool zdravo) :
        Stablo(carstvo, porodica, rod, lotBroj), tipStabla(tipStabla),
        starost(starost), datumSadnje(datumSadnje), zdravo(zdravo) {
               brojSadnice++;
        }

        StabloSljive(const StabloSljive &s) : Stablo(s.carstvo, s.porodica,
                     s.rod, s.lotBroj), tipStabla(s.tipStabla),
                     starost(s.starost), datumSadnje(s.datumSadnje),
                     zdravo(s.zdravo) {
               brojSadnice++;
        }

        ~StabloSljive() { brojSadnice--; }

        DinString getTipStabla() const {
             return tipStabla;
        }
        int getStarost() const {
             return starost;
        }
        DinString getDatumSadnje() const {
             return datumSadnje;
        }
        bool getZdravo() const {
             return zdravo;
        }
        int getBrojSadnice() const {
             return brojSadnice;
        }

        void setTipStabla(const DinString &tipStabla) {
             this -> tipStabla = tipStabla;
        }
```

```
        void setStarost(const int starost) {
             this -> starost = starost;
        }
        void setDatumSadnje(const DinString &datumSadnje) {
             this -> datumSadnje = datumSadnje;
        }
        void setZdravo(const bool zdravo) {
            this -> zdravo = zdravo;
        }

        DinString getTipPloda() const {
            return tipStabla;
        }

        double getPrinos() const {
            double koeficijentPrinosaSorte = 12.45;
            int uzeto = brojSadnice, generacija = 0;

            while(uzeto >= 0) {
                generacija++;
                uzeto -= 3;
            }

            while(generacija >= 0) {
                koeficijentPrinosaSorte *= 0.97;
                generacija--;
            }
            return (starost * 0.65) * koeficijentPrinosaSorte;
        }

        bool hemijskiTretman() {
            if(zdravo) {
                return false;
            }
            else {
                zdravo = true;
                return true;
            }
        }

        friend ostream& operator<<(ostream &out, const StabloSljive &s) {
            out << "----------------- STABLO SLJIVE ------------------" << endl;
            out << "\tCarstvo: "        << s.carstvo << endl;
            out << "\tPorodica: "       << s.porodica << endl;
            out << "\tRod: "            << s.rod << endl;
            out << "\tLot. broj: "      << s.lotBroj << endl;
            out << "\tTip stabla: "     << s.tipStabla << endl;
            out << "\tStarost: "        << s.starost << endl;
            out << "\tDatum sadnje: "   << s.datumSadnje << endl;
            out << "\tZdravo: "         << (s.zdravo == true ? "DA" : "NE")
                << endl << endl;

            return out;
        }
```

```
        void prevariOstream(ostream &out) const {
            out << "----------------- STABLO SLJIVE -----------------" << endl;
            out << "\tCarstvo: "           << carstvo << endl;
            out << "\tPorodica: "          << porodica << endl;
            out << "\tRod: "               << rod << endl;
            out << "\tLot. broj: "         << lotBroj << endl;
            out << "\tTip stabla: "        << tipStabla << endl;
            out << "\tStarost: "           << starost << endl;
            out << "\tDatum sadnje: "      << datumSadnje << endl;
            out << "\tZdravo: "            << (zdravo == true ? "DA" : "NE")
                << endl << endl;
        }
};

#endif // STABLOSLJIVE_HPP_INCLUDED
```

VOCNJAK.hpp

```
#ifndef VOCNJAK_HPP_INCLUDED
#define VOCNJAK_HPP_INCLUDED

#include "StabloJabuke.hpp"
#include "StabloSljive.hpp"

class Vocnjak {
    private:
        List<Stablo *> sadrzajVocnjaka;
        int godinaEksploatacije;
    public:
        Vocnjak() : sadrzajVocnjaka(), godinaEksploatacije(0) { }
        Vocnjak(const int godinaEksploatacije) : godinaEksploatacije(0) { }
        Vocnjak(const Vocnjak &v) : godinaEksploatacije(v.godinaEksploatacije){}

        List<Stablo *> getSadrzajVocnjaka() const {
            return sadrzajVocnjaka;
        }

        bool dodajStablo(Stablo &s) {
            if(sadrzajVocnjaka.empty()) {
                return sadrzajVocnjaka.add(1, &s);
            }

            Stablo *tmp;

            for(int i = 1; i <= sadrzajVocnjaka.size(); i++) {
                sadrzajVocnjaka.read(i, tmp);

                if(tmp -> getLotBroj() == s.getLotBroj()) {
                    return false;
                }
            }
            return sadrzajVocnjaka.add(sadrzajVocnjaka.size() + 1, &s);
        }
```

```
        void sortiraj(const DinString &nacinSortiranja) {
            if(sadrzajVocnjaka.empty()) {
                cout << endl << "VOCNJAK JE PRAZAN!" << endl;
                return;
            }

            if(nacinSortiranja == "RASTUCE") {
                Stablo *s1, *s2;

                 for(int i = 1; i < sadrzajVocnjaka.size(); i++) {
                        for(int j = i + 1; j <= sadrzajVocnjaka.size(); j++) {
                            sadrzajVocnjaka.read(i, s1);
                            sadrzajVocnjaka.read(j, s2);

                            if(s1 -> getLotBroj() < s2 -> getLotBroj()) {
                                sadrzajVocnjaka.remove(i);
                                sadrzajVocnjaka.add(i, s2);
                                sadrzajVocnjaka.remove(j);
                                sadrzajVocnjaka.add(j, s1);
                            }
                        }
                    }
            }
            else if(nacinSortiranja == "OPADAJUCE") {
                Stablo *s1, *s2;

                for(int i = 1; i < sadrzajVocnjaka.size(); i++) {
                        for(int j = i + 1; j <= sadrzajVocnjaka.size(); j++) {
                            sadrzajVocnjaka.read(i, s1);
                            sadrzajVocnjaka.read(j, s2);

                            if(s1 -> getLotBroj() > s2 -> getLotBroj()) {
                                sadrzajVocnjaka.remove(i);
                                sadrzajVocnjaka.add(i, s2);
                                sadrzajVocnjaka.remove(j);
                                sadrzajVocnjaka.add(j, s1);
                            }
                        }
                    }
            }
            else {
                return;
            }
        }

        Stablo* pronadji(const int lotBrojStabla) {
            if(sadrzajVocnjaka.empty())
                return NULL;
            else {
                Stablo *tmp;

                for(int i = 1; i <= sadrzajVocnjaka.size(); i++) {
                    sadrzajVocnjaka.read(i, tmp);
                    if(tmp -> getLotBroj() == lotBrojStabla)
                        return tmp;
                }
                return NULL;
            }
        }
```

```
        int prebroj(const DinString &ts) const {
            if(sadrzajVocnjaka.empty()) {
                return 0;
            }
            else {
                Stablo *tmp;
                int brojStabla = 0;

                for(int i = 1; i <= sadrzajVocnjaka.size(); i++) {
                    sadrzajVocnjaka.read(i, tmp);

                    if(tmp -> getTipStabla() == ts) {
                        brojStabla++;
                    }
                }
                return brojStabla;
            }
        }

        friend ostream& operator<<(ostream &out, const Vocnjak &v) {
            if(v.sadrzajVocnjaka.empty()) {
                out << endl << "VOCNJAK JE PRAZAN!" << endl << endl;
            }
            else
            {
                out << "------------------- VOCNJAK -----------------" << endl;

                for(int i = 1; i <= v.sadrzajVocnjaka.size(); i++) {
                    Stablo *tmp;
                    v.sadrzajVocnjaka.read(i, tmp);
                    out << *tmp;
                }
            }

            return out;
        }
};

#endif // VOCNJAK_HPP_INCLUDED
```

MAIN.cpp

```
#include "Vocnjak.hpp"

int Stablo::rbrSadnice = 0;
int StabloJabuke::brojSadnice = 0;
int StabloSljive::brojSadnice = 0;

double prosecniPrinos(const Vocnjak &v) {
    double prosek = 0.0;
    int brojac = 0;

    List<Stablo *> voce = v.getSadrzajVocnjaka();
    Stablo *tmp;
```

```
    if(voce.empty()) {
        return 0.0;
    }
    else {
        for(int i = 1; i <= voce.size(); i++) {
            voce.read(i, tmp);
            prosek += tmp -> getPrinos();
            brojac++;
        }
        prosek /= brojac;
        return prosek;
    }
}

int main(void)
{
    // STABLO JABUKE
    StabloJabuke sj1;
    StabloJabuke sj2("Pate", "Apple", "A", 123, "JABUKA", 2, "1.1.2023.", ZUTA);
    StabloJabuke sj3(sj1);

    cout << sj1 << sj2 << sj3;

    sj1.getBojaJabuke();   sj1.getBrojSadnice();  sj1.getCarstvo();
    sj1.getDatumSadnje();  sj1.getLotBroj();      sj1.getPorodica();
    sj1.getRbrSadnice();   sj1.getRod();          sj1.getStarost();

    sj1.setBojaJabuke(ZUTA);
    sj1.setStarost(3);
    sj1.setCarstvo("Biljaka");
    sj1.setDatumSadnje("23.12.2021.");
    sj1.setLotBroj(112);
    sj1.setPorodica("Jabuke");
    sj1.setRod("JAB");
    sj1.setTipStabla("JABUKA");

    cout << endl << "SJ1 = SJ2" << endl;
    sj1 = sj2;
    cout << sj1;

    // STABLO SLJIVE
    StabloSljive ss1;
    StabloSljive ss2("Pla", "Sljive", "Nat", 3, "SLJIVA", 4, "1.1.2022.", true),
    StabloSljive ss3(ss1);

    cout << ss1 << ss2 << ss3;

    ss1.getZdravo();       ss1.getBrojSadnice();  ss1.getCarstvo();
    ss1.getDatumSadnje();  ss1.getLotBroj();      ss1.getPorodica();
    ss1.getRbrSadnice();   ss1.getRod();          ss1.getStarost();

    ss1.setZdravo(false);
    ss1.setStarost(3);
    ss1.setCarstvo("Biljaka");
    ss1.setDatumSadnje("23.12.2021.");
    ss1.setLotBroj(112);
    ss1.setPorodica("Jabuke");
    ss1.setRod("SLJIVA");
    ss1.setTipStabla("SLJIVA");
```

```
    cout << ss1;

    // VOCNJAK
    Vocnjak v1;

    cout << "DODAJ SJ1: " <<
            (v1.dodajStablo(sj1) == true ? "DODATO" : "NIJE DODATO") << endl;
    cout << "DODAJ SJ2: " <<
            (v1.dodajStablo(sj2) == true ? "DODATO" : "NIJE DODATO") << endl;
    cout << "DODAJ SJ2: " <<
            (v1.dodajStablo(sj2) == true ? "DODATO" : "NIJE DODATO") << endl;
    cout << "DODAJ SS1: " <<
            (v1.dodajStablo(ss1) == true ? "DODATO" : "NIJE DODATO") << endl;
    cout << "DODAJ SS1: " <<
            (v1.dodajStablo(ss1) == true ? "DODATO" : "NIJE DODATO") << endl;
    cout << "DODAJ SS2: " <<
            (v1.dodajStablo(ss2) == true ? "DODATO" : "NIJE DODATO") << endl;

    cout << endl << "PROSECNI PRINOS: " << prosecniPrinos(v1) << endl;

    cout << v1;
    v1.sortiraj("RASTUCE");
    cout << endl << "NAKON SORTIRANJA" << endl << v1;

    cout << "TRAZI SJ1 (112 LB): " << endl;
    Stablo *trazi = v1.pronadji(112);

    if(trazi != NULL)
        cout << *trazi;
    else
        cout << "NIJE PRONADJENO!" << endl;

    cout << "BROJ STABALA SLJIVA U VOCNJAKU JE: "
         << v1.prebroj("SLJIVA") << endl;
    cout << "BROJ STABALA JABUKA U VOCNJAKU JE: "
         W<< v1.prebroj("JABUKA") << endl;

    return 0;
}
```

DINSTRING.hpp

```
#ifndef DINSTRING_DEF
#define DINSTRING_DEF
#include <iostream>

using namespace std;

class DinString {
    private:
        int duzina;
        char *text;
    public:
        DinString();
        DinString(const char[]);
```

```
        DinString(const DinString&);
        ~DinString();

        int length() const;

        char& operator[] (int);
        char operator[] (int) const;

        DinString& operator=(const DinString&);
        DinString& operator+=(const DinString&);

        friend bool operator==(const DinString&, const DinString&);
        friend bool operator!=(const DinString&, const DinString&);

        friend DinString operator+(const DinString&, const DinString&);

        friend ostream& operator<<(ostream&, const DinString&);
};

#endif
```

DINSTRING.cpp

```
#include "dinstring.hpp"

DinString::DinString() {
    duzina = 0;
    text = NULL;
}

DinString::DinString(const char ulaz[]) {
    duzina = 0;
    while(ulaz[duzina] != '\0')
        duzina++;

    text = new char[duzina + 1];
    for(int i = 0; i < duzina; i++)
        text[i] = ulaz[i];

    text[duzina] = '\0';
}

DinString::DinString(const DinString& ds) {
    duzina = ds.duzina;

    text = new char[duzina + 1];
    for(int i = 0; i < duzina; i++)
        text[i] = ds.text[i];

    text[duzina] = '\0';
}

DinString::~DinString() {
    delete[] text;
}
```

```cpp
int DinString::length() const {
    return duzina;
}

char& DinString::operator[] (int i) {
    return text[i];
}

char DinString::operator[] (int i) const {
    return text[i];
}

DinString& DinString::operator=(const DinString& ds) {
    if(this != &ds) {
        delete[] text;
        duzina = ds.duzina;
        text = new char[duzina + 1];
        for (int i = 0; i < duzina; i++)
            text[i] = ds.text[i];

        text[duzina] = '\0';
    }

    return *this;
}

DinString& DinString::operator+=(const DinString& ds) {
    int i;
    char *tempText = new char[duzina + ds.duzina + 1];
    for (i = 0; i < duzina; i++)
        tempText[i] = text[i];
    for (i = 0; i < ds.duzina; i++)
        tempText[duzina + i] = ds.text[i];
    tempText[duzina + ds.duzina] = '\0';

    duzina += ds.duzina;

    delete[] text;
    text = tempText;

    return *this;
}

bool operator==(const DinString& ds1, const DinString& ds2){
    if(ds1.duzina != ds2.duzina)
        return false;

    for(int i = 0; i < ds1.duzina; i++)
        if(ds1.text[i] != ds2.text[i])
            return false;

    return true;
}
```

```
bool operator!=(const DinString& ds1, const DinString& ds2){
    if(ds1.duzina != ds2.duzina)
        return true;

    for(int i = 0; i < ds1.duzina; i++)
        if(ds1.text[i] != ds2.text[i])
            return true;

    return false;
}

DinString operator+(const DinString& ds1, const DinString& ds2){
    DinString temp;
    temp.duzina = ds1.duzina + ds2.duzina;

    temp.text = new char[temp.duzina + 1];

    int i;
    for(i = 0; i < ds1.duzina; i++)
        temp.text[i] = ds1.text[i];

    for(i = 0; i < ds2.duzina; i++)
        temp.text[ds1.duzina + i] = ds2.text[i];
    temp.text[temp.duzina] = '\0';

    return temp;

}

ostream& operator<<(ostream& out, const DinString& ds) {
    if(ds.duzina > 0)
        out << ds.text;

    return out;
}
```

LIST.hpp

```
#ifndef LIST_DEF
#define LIST_DEF

#include <stdlib.h>
#include <iostream>

using namespace std;

template <class T>
class List {
    private:
        struct listEl {
            T content;
            struct listEl* next;
        };

        listEl *head;
        listEl *tail;
        int noEl;
```

```
    public:
        List(){
            head = tail = NULL;
            noEl = 0;
        }

        List(const List<T>&);
        virtual ~List();

        List<T>& operator=(const List<T>&);
        int size() const { return noEl; }
        bool empty() const { return head == NULL ? 1 : 0; }

        bool add(int, const T&);
        bool remove(int);
        bool read(int, T&)const;
        void clear();
};

template <class T>
ostream& operator<<(ostream& out, const List<T>& rl) {
    out << endl;
    out << "--------" << endl;
    for(int i = 1; i <= rl.size(); i++){
        if(i != 1) out << ", ";
        T res;
        rl.read(i, res);
        out << res;
    }
    out << endl << "--------" << endl;
    return out;
}

template <class T>
List<T>::List(const List<T>& rl) {
    head = NULL;
    tail = NULL;
    noEl = 0;

    for(int i = 1; i <= rl.noEl; i++) {
        T res;
        if(rl.read(i, res))
            add(i, res);
    }
}

template <class T>
List<T>& List<T>::operator=(const List<T>& rl) {
    if(this != &rl) {
        clear();
        head = NULL;
        tail = NULL;
        noEl = 0;

        for(int i = 1; i <= rl.noEl; i++){
            T res;
            if(rl.read(i, res))
                add(i, res);
        }
```

```
    }
    return *this;
}

template <class T>
List<T>::~List() {
    while(!empty())
        remove(1);
}

template <class T>
bool List<T>::add(int n, const T& newContent) {
    if(n < 1 || n > noEl + 1)
        return false;
    else {
        listEl* newEl = new listEl;
        if(newEl == NULL)
            return false;
        else {
            newEl->content = newContent;
            if(n == 1) {
                newEl->next = head;
                head = newEl;
            } else if(n == noEl + 1) {
                newEl->next = NULL;
                tail->next = newEl;
            } else {
                listEl* temp = head;
                for(int i = 2; i < n; i++)
                    temp = temp->next;
                newEl->next = temp->next;
                temp->next = newEl;
            }
            noEl++;
            if(newEl->next == NULL)
                tail = newEl;
            return true;
        }
    }
}
```

```
template <class T>
bool List<T>::remove(int n){
    if(n < 1 || n > noEl)
        return false;
    else {
        if(n == 1) {
            listEl* del = head;
            head = head->next;
            if(tail == del)
                tail = NULL;
            delete del;
            noEl--;
        } else {
            listEl* temp = head;
            for(int i = 2; i < n; i++)
                temp = temp->next;
            listEl* del = temp->next;
            temp->next = del->next;
            if(tail == del)
                tail = temp;
            delete del;
            noEl--;
        }
        return true;
    }
}

template <class T>
bool List<T>::read(int n,T& retVal) const {
    if(n < 1 || n > noEl)
        return false;
    else {
        if(n == 1)
            retVal = head->content;
        else if(n == noEl)
            retVal = tail->content;
        else {
            listEl* temp = head;
            for(int i = 1; i < n; i++)
                temp = temp->next;
            retVal = temp->content;
        }
        return true;
    }
}

template <class T>
void List<T>::clear() {
    while(!empty())
        remove(1);
}

#endif
```