II PARCIJALNI ISPIT IZ PREDMETA "TEHNIKE PROGRAMIRANJA" (GRUPA A)

NAPOMENA: *Sve funkcije čija je implementacija duža od dvije naredbe obavezno implementirajte izvan klase. Također, sve metode koje su inspektori obavezno deklarirajte kao takve.*

Zadatak 1 (6 poena):

Svima koji se bave tehničkim naukama dobro su poznati komplesni brojevi i njihova svojstva, koji se mogu formalno posmatrati kao uređeni parovi realnih brojeva oblika (a, b) gdje su a i b realni brojevi. Međutim, iako su kompleksni brojevi najpoznatije i najprimjenljivije tvorevine takve vrste, one nisu i jedini korisni matematički objekti koji se modeliraju kao parovi realnih brojeva. Tako su u modernoj fizici, a posebno u specijalnoj teoriji relativnosti (uključujući i špekulativne teorije koje predviđaju postojanje i opisuju svojstva čestica koje se kreću nadsvjetlosnom brzinom) veliku primjenu našli bliski srodnici komplesnih brojeva poznati kao *perpleksni brojevi* (susreću se i nazivi *hiperbolički brojevi*, *kontrakompleksni brojevi* i još neki drugi). Oni posjeduju prividno bizarno svojstvo da im apsolutna vrijednost može biti i *negativna*. Intuitivno, perpleksni brojevi su brojevi oblika $x + y\,h$, gdje su x i y realni brojevi, dok je h tzv. *hiperbolička* (*kontraimaginarna*, *halucinogena*) *jedinica*, koja je blizak srodnik imaginarne jedinice i, ali za nju vrijedi $h^2 = 1$ (dok je $i^2 = -1$, kao što znamo). Pri tome je bitno naglasiti da iako je $h^2 = 1$, vrijedi $h \neq 1$ i $h \neq -1$ (postojanje ovakvih bizarnih objekata može se lako opravdati pomoću linearne algebre). Interesantno je da se h od 1 i -1 razlikuje i po bizarnom svojstvu $|h| = -1$. Formalno, perpleksni brojevi se mogu također predstaviti kao parovi realnih brojeva, koje ćemo označiti sa $(x|y)$ da bismo ih razlikovali od kompleksnih brojeva (x, y), i za koje vrijede sljedeća pravila računanja:

$$(x_1|y_1) \pm (x_2|y_2) = (x_1 \pm x_2 \,|\, y_1 \pm y_2) \qquad (x_1|y_1) \cdot (x_2|y_2) = (x_1x_2 + y_1y_2 \,|\, x_1y_2 + x_2y_1)$$

$$(x_1|y_1) / (x_2|y_2) = \left(\frac{x_1x_2 - y_1y_2}{{x_2}^2 - {y_2}^2} \,\middle|\, \frac{x_2y_1 - x_1y_2}{{x_2}^2 - {y_2}^2}\right)$$

Može se primijetiti upadljiva sličnost sa računom sa kompleksnim brojevima, uz male izmjene u predznacima (koje su posljedica činjenice da je h^2 jednako 1 a ne -1). Uz ovakav formalizam, imamo $h = (0|1)$. Broj x se naziva *realni*, a broj y *hiperbolni* (*kontraimaginarni*, *halucinogeni*) dio perpleksnog broja $(x|y)$. Negacija perpleksnog broja $(x|y)$ data je kao $(-x|-y)$, dok je njegova konjugacija data kao $(x|-y)$. Apsolutna vrijednost perpleksnog broja $(x|y)$ data je kao

$$|(x|y)| = \sqrt{x^2 - y^2} \text{ za } x^2 \geq y^2, \qquad |(x|y)| = -\sqrt{y^2 - x^2} \text{ za } x^2 < y^2$$

Perpleksni broj $(x|0)$ može se poistovijetiti sa realnim brojem x. Dva perpleksna broja $(x_1|y_1)$ i $(x_2|y_2)$ jednaki su ako i samo ako je $x_1 = x_2$ i $y_1 = y_2$. Poredak perpleksnih brojeva se ne definira.

Definirajte *klasu za rad sa perpleksnim brojevima*. Klasa bi trebala da sadrži konstruktor koji kreira perpleksni broj na osnovu zadanog realnog i hiperbolnog dijela koji se zadaju kao parametri. Oba parametra trebaju imati podrazumijevanu vrijednost 0, tako da će se ovaj konstruktor moći koristiti i kao konstruktor bez parametara odnosno sa jednim parametrom. Klasa treba podržavati dvije pristupne metode bez parametara, kojima se pristupa realnom i hiperbolnom dijelu perpleksnog broja. Dalje, treba podržavati operatore "`+`", "`-`", "`*`" i "`/`" za obavljanje četiri osnovne računske operacije, zatim kombinovane operatore "`+=`", "`-=`", "`*=`" i "`/=`" sa uobičajenim značenjem, relacione operatore "`==`" i "`!=`", unarni operator negacije "`-`", te unarni operator "`++`" koji povećava realni dio perpleksnog broja za jedinicu, dok hiperbolni dio ostaje isti (potrebno je podržati i prefiksnu i postfiksnu verziju ovog operatora). Također je potrebno podržati i operatore "`<<`" i "`>>`" za ispis perpleksnih brojeva na izlazni tok i njihov unos sa ulaznog toka. Perpleksne brojeve treba ispisivati kao parove oblika $(x|y)$ bez obzira da li su im neke komponente nule ili nisu, dok unos perpleksnih brojeva sa ulaznog toka treba podržati bilo kao par oblika $(x|y)$, bilo kao običan realan broj, koji se potom interpretira kao perpleksni broj. Konačno, treba implementirati i četiri obične funkcije (ne članice) koje kao rezultat daju respektivno apsolutnu vrijednost, konjugaciju, realni i hiperbolni dio perpleksnog broja (neka Vas ne zbunjuje što postoje i metode koje daju realni i hiperbolni dio perpleksnog broja – ovako se omogućava veća sintaksna šarolikost u primjeni).

Rješenje:

```
class Perpleksni {
  double re, hip;
public:
  Perpleksni(double realni = 0, double hiperbolni = 0) : re(realni),
    hip(hiperbolni) {}
  double DajRealni() const { return re; }
  double DajHiperbolni() const { return hip; }
  friend Perpleksni operator +(const Perpleksni &p1, const Perpleksni &p2) {
    return Perpleksni(p1.re + p2.re, p1.hip + p2.hip);
  }
  friend Perpleksni operator -(const Perpleksni &p1, const Perpleksni &p2) {
    return Perpleksni(p1.re - p2.re, p1.hip - p2.hip);
  }
  friend Perpleksni operator *(const Perpleksni &p1, const Perpleksni &p2) {
    return Perpleksni(p1.re * p2.re + p1.hip * p2.hip,
      p1.re * p2.hip + p2.re * p1.hip);
  }
  friend Perpleksni operator /(const Perpleksni &p1, const Perpleksni &p2) {
    double nazivnik(p2.re * p2.re - p2.hip * p2.hip);
    return Perpleksni((p1.re * p2.re + p1.hip * p2.hip) / nazivnik,
      (p2.re * p1.hip - p1.re * p2.hip) / nazivnik);
  }
  Perpleksni &operator +=(const Perpleksni &p) { return *this = *this + p; }
  Perpleksni &operator -=(const Perpleksni &p) { return *this = *this - p; }
  Perpleksni &operator *=(const Perpleksni &p) { return *this = *this * p; }
  Perpleksni &operator /=(const Perpleksni &p) { return *this = *this / p; }
  friend bool operator ==(const Perpleksni &p1, const Perpleksni &p2) {
    return p1.re == p2.re && p1.hip == p2.hip;
  }
  friend bool operator !=(const Perpleksni &p1, const Perpleksni &p2) {
    return !(p1 == p2);
  }
  Perpleksni operator -() { return Perpleksni(-re, -hip); }
  Perpleksni &operator ++() { re++; return *this; }
  Perpleksni operator ++(int) { Perpleksni p(*this); re++; return p; }
  friend ostream &operator <<(ostream &cout, const Perpleksni &p) {
    return cout << "(" << p.re << "|" << p.hip << ")";
  }
  friend istream &operator >>(istream &cin, Perpleksni &p);
  friend double ApsolutnaVrijednost(const Perpleksni &p);
  friend Perpleksni Konjugirani(const Perpleksni &p) {
    return Perpleksni(p.re, -p.hip);
  }
  friend double RealniDio(const Perpleksni &p) { return p.re; }
  friend double HiperbolniDio(const Perpleksni &p) { return p.hip; }
};
double ApsolutnaVrijednost(const Perpleksni &p) {
  double norma(p.re * p.re - p.hip * p.hip);
  if(norma >= 0) return sqrt(norma);
  else return -sqrt(-norma);
}
istream &operator >>(istream &cin, Perpleksni &p) {
  if(cin.peek() != '(') {
    cin >> p.re; p.hip = 0;
  }
  else {
    char znak;
    cin >> znak;
    if(znak != '(') cin.setstate(ios::failbit);
    cin >> p.re >> znak;
    if(znak != '|') cin.setstate(ios::failbit);
    cin >> p.hip >> znak;
    if(znak != ')') cin.setstate(ios::failbit);
  }
  return cin;
}
```

Zadatak 2 (8 poena):

Uprava neke radne organizacije je odlučila da napravi interni telefonski imenik svojih uposlenika. Podaci o jednom uposleniku čuvaju se u strukturi koja kao svoje atribute sadrži ime uposlenika zajedno sa prezimenom (tipa niza od max. 20 znakova), naziv odjeljenja (tipa niza od max. 10 znakova) i broj telefona (cjelobrojnog tipa). Potrebno je razviti kontejnersku klasu koja će čuvati kolekciju podataka o uposlenicima i njihovim telefonskim brojevima. Ovi podaci će se alocirati dinamički, a pristupaće im se preko dinamički alociranog niza pokazivača koji pokazuju na ove podatke. Interfejs klase treba sadržavati sljedeće elemente:

a) Konstruktor sa jednim parametrom koji vrši neophodnu dinamičku alokaciju, pri čemu parametar predstavlja maksimalni broj uposlenika koji se mogu evidentirati. Pri tome se ovaj konstruktor ne smije koristiti za automatsku konverziju cjelobrojnih podataka u tip ove klase.
b) Destruktor, koji oslobađa sve resurse koje su primjerci ove klase zauzeli tokom svog života.
c) Konstruktor kopije i preklopljeni operator dodjele koji omogućavaju da se primjerci ove klase mogu bezbjedno kopirati i međusobno dodjeljivati.
d) Metodu koja dodaje u kolekciju podatke o novom uposleniku, pri čemu su parametri metode podaci o uposleniku (ime i prezime, odjeljenje i broj telefona). U slučaju da su podaci o imenu i prezimenu ili odjeljenju predugi, ili ukoliko se dostigne maksimalni broj uposlenika koji se mogu evidentirati, treba baciti izuzetak.
e) Metodu koja ispisuju podatke o uposleniku čije se ime i prezime zadaje kao parametar i metodu koja ispisuje podatke o uposleniku čiji se broj telefona zadaje kao parametar.
f) Metodu koja ispisuje telefonski imenik za sve uposlenike koji se nalaze u odjeljenju koje se zadaje kao parametar, zatim metodu koja ispisuje telefonski imenik za sve uposlenike čije ime počinje slovom koje se zadaje kao parametar, te metodu koja ispisuje čitav telefonski imenik.
g) Metodu koje sortira imenik po abecednom poretku uposlenika (koristiti funkciju "`sort`").
h) Preklopljeni operator "`[]`" koji vraća referencu na broj telefona uposlenika čije se ime zadaje unutar uglastih zagrada (referenca se vraća sa ciljem da se omogući izmjena broja). U slučaju da se ovaj operator primijeni na konstantni objekat, umjesto reference treba vratiti kopiju broja.
i) Metodu koja snima čitav telefonski imenik u binarnu datoteku čije se ime zadaje kao parametar, te konstruktor sa jednim parametrom koji prilikom kreiranja imenika obnavlja njegov sadržaj iz binarne datoteke čije se ime zadaje kao parametar konstruktora.

Rješenje:

Napomena: U prikazanom rješenju, struktura "`Uposlenik`" umjesto da se deklarira izvan klase na globalnom nivou, deklarirana je lokalno u privatnom dijelu klase, tako da joj samo metode klase "`Imenik`" mogu pristupiti, odnosno ostatak programa "ne zna" za postojanje ove strukture. Mada se to nije očekivalo od studenta da uradi, ovdje je prikazano čisto kao ilustracija da se takve tehnike često koriste.

```
class Imenik {
  struct Uposlenik {
    char ime[20], odjeljenje[10];
    int telefon;
  };
  int kapacitet, broj_uposlenika;
  Uposlenik **uposlenici;
  static void IspisiPodatke(Uposlenik *u); // Pomoćna funkcija za ispis
  static bool KriterijSortiranja(Uposlenik *u1, Uposlenik *u2) {
    return strcmp(u1->ime, u2->ime) < 0;
  }
public:
  explicit Imenik(int kapacitet) : kapacitet(kapacitet), broj_uposlenika(0),
    uposlenici(new Uposlenik*[kapacitet]) {}
  ~Imenik();
  Imenik(const Imenik &im);
  Imenik &operator=(const Imenik &im);
  void DodajUposlenika(const char ime[], const char odjeljenje[], int telefon);
  void IspisiUposlenikaSaImenom(const char ime[]) const;
  void IspisiUposlenikaSaTelefonom(int telefon) const;
  void IspisiImenikZaOdjeljenje(const char odjeljenje[]) const;
```

```
  void IspisiImenikNaSlovo(char slovo) const;
  void IspisiCitavImenik() const;
  void SortirajImenik() {
    sort(uposlenici, uposlenici + broj_uposlenika, KriterijSortiranja);
  }
  int &operator[](const char ime[]);
  int operator[](const char ime[]) const;
  void Sacuvaj(const char ime_datoteke[]);
  explicit Imenik(const char ime_datoteke[]);
};
Imenik::~Imenik() {
  for(int i = 0; i < broj_uposlenika; i++) delete uposlenici[i];
  delete[] uposlenici;
}
Imenik::Imenik(const Imenik &im) : broj_uposlenika(im.broj_uposlenika),
  kapacitet(im.kapacitet), uposlenici(new Uposlenik*[im.kapacitet]) {
    for(int i = 0; i < broj_uposlenika; i++)
      uposlenici[i] = new Uposlenik(*im.uposlenici[i]);
}
Imenik &Imenik::operator =(const Imenik &im) {
  if(&im == this) return *this;
  for(int i = 0; i < broj_uposlenika; i++) delete uposlenici[i];
  delete[] uposlenici;
  broj_uposlenika = im.broj_uposlenika; kapacitet = im.kapacitet;
  uposlenici = new Uposlenik*[kapacitet];
  for(int i = 0; i < broj_uposlenika; i++)
    uposlenici[i] = new Uposlenik(*im.uposlenici[i]);
}
void Imenik::DodajUposlenika(const char ime[], const char odjeljenje[],
  int telefon) {
    if(broj_uposlenika == kapacitet) throw "Popunjen kapacitet!";
    Uposlenik *novi(new Uposlenik);
    strcpy(novi->ime, ime); strcpy(novi->odjeljenje, odjeljenje);
    novi->telefon = telefon;
    uposlenici[broj_uposlenika++] = novi;
}
void Imenik::IspisiPodatke(Uposlenik *u) {
  cout << "Ime i prezime: " << u->ime << endl << "Odjeljenje: "
    << u->odjeljenje << endl << "Broj telefona: " << u->telefon << endl;
}
void Imenik::IspisiUposlenikaSaImenom(const char ime[]) const {
  int brojac(0);
  for(int i = 0; i < broj_uposlenika; i++)
    if(strcmp(uposlenici[i]->ime, ime) == 0) {
      IspisiPodatke(uposlenici[i]); brojac++;
    }
  if(brojac == 0) throw "Nema uposlenika sa tim imenom!";
}
void Imenik::IspisiUposlenikaSaTelefonom(int telefon) const {
  int brojac(0);
  for(int i = 0; i < broj_uposlenika; i++)
    if(uposlenici[i]->telefon == telefon) {
      IspisiPodatke(uposlenici[i]); brojac++;
    }
  if(brojac == 0) throw "Nema uposlenika sa tim brojem telefona!";
}
void Imenik::IspisiImenikZaOdjeljenje(const char odjeljenje[]) const {
  for(int i = 0; i < broj_uposlenika; i++)
    if(strcmp(uposlenici[i]->odjeljenje, odjeljenje) == 0)
      IspisiPodatke(uposlenici[i]);
}
void Imenik::IspisiImenikNaSlovo(char slovo) const {
  for(int i = 0; i < broj_uposlenika; i++)
    if(uposlenici[i]->ime[0] == slovo) IspisiPodatke(uposlenici[i]);
}
```

```
void Imenik::IspisiCitavImenik() const {
  for(int i = 0; i < broj_uposlenika; i++) IspisiPodatke(uposlenici[i]);
}
int &Imenik::operator[](const char ime[]) {
  for(int i = 0; i < broj_uposlenika; i++)
    if(strcmp(uposlenici[i]->ime, ime) == 0) return uposlenici[i]->telefon;
  throw "Nema uposlenika sa tim imenom!";
}
int Imenik::operator[](const char ime[]) const {
  for(int i = 0; i < broj_uposlenika; i++)
    if(strcmp(uposlenici[i]->ime, ime) == 0) return uposlenici[i]->telefon;
  throw "Nema uposlenika sa tim imenom!";
}
void Imenik::Sacuvaj(const char ime_datoteke[]) {
  ofstream datoteka(ime_datoteke);
  if(!datoteka) throw "Kreiranje datoteke nije uspjelo!";
  datoteka.write((char*)this, sizeof(*this));
  for(int i = 0; i < broj_uposlenika; i++)
    datoteka.write((char*)uposlenici[i], sizeof(**uposlenici));
  if(!datoteka) throw "Snimanje nije uspjelo!";
}
Imenik::Imenik(const char ime_datoteke[]) {
  ifstream datoteka(ime_datoteke);
  if(!datoteka) throw "Datoteka ne postoji!";
  datoteka.read((char*)this, sizeof(*this));
  uposlenici = new Uposlenik*[kapacitet];
  for(int i = 0; i < broj_uposlenika; i++) {
    uposlenici[i] = new Uposlenik;
    datoteka.read((char*)uposlenici[i], sizeof(**uposlenici));
  }
  if(!datoteka) throw "Citanje nije uspjelo!";
}
```

Zadatak 3 (6 poena):

Neka je *vozilo* objekat koji je, između ostalog, karakteriziran svojom težinom. *Putničko vozilo* je specijalna vrsta vozila, koje može primiti određeni broj putnika od kojih svaki ima svoju težinu. *Teretno vozilo* je specijalna vrsta vozila koje se može nakrcati teretom određene težine. Razvijte hijerarhiju klasa koje opisuju ove objekte. Bazna klasa "`Vozilo`" posjedovaće atribut koji predstavlja težinu vozila (tipa cijeli broj), konstruktor koji inicijalizira ovaj atribut, te dvije metode nazvane "`DajTezinu`" i "`DajUkupnuTezinu`". Prva metoda daje vlastitu težinu vozila, dok druga metoda daje ukupnu težinu vozila u koju je uračunata i težina svega što se u vozilu nalazi (u slučaju bazne klase "`Vozilo`" obje metode će vraćati istu vrijednost). Klasa "`PutnickoVozilo`" nasljeđuje se iz klase "`Vozilo`", a posjeduje dodatni atribut koji je tipa vektor cijelih brojeva, koji čuva težine putnika u vozilu. Konstruktor ove klase ima dodatni parametar (vektor težina putnika) i izmijenjenu metodu "`DajUkupnuTezinu`", koja uračunava težine putnika. Klasa "`TeretnoVozilo`" se također nasljeđuje iz klase "`Vozilo`", a posjeduje dodatni cjelobrojni atribut koji predstavlja težinu tereta. Ova klasa također ima dodatni parametar u konstruktoru (težina tereta) i izmijenjenu metodu za računanje ukupne težine. Metoda "`DajUkupnuTezinu`" mora biti izvedena tako da ukoliko se svim opisanim objektima pristupa preko pokazivača na baznu klasu "`Vozilo`", uvijek bude pozvana ispravna verzija metode, koja će uzeti u obzir specifičnosti objekta.

Napisane klase demonstrirajte u testnom programu koji čita podatke o vozilima iz tekstualna datoteke u vektor, sortira vozila po ukupnoj težini u rastući poredak i na kraju ispisuje težine vozila nakon sortiranja (elementi vektora će biti tipa pokazivači na "`Vozilo`" da bi se mogao koristiti polimorfizam). Svaki red datoteke sadrži podatke o jednom vozilu, poput "`V500`" za obično vozilo težine 500, "`P500 3 80 60 75`" za putničko vozilo težine 500 sa 3 putnika težina 80, 60 i 75 respektivno, te "`T500 1200`" za teretno vozilo težine 500 natovareno sa teretom težine 1200. Pretpostavite da datoteka sadrži ispravne podatke.

Rješenje:

```
class Vozilo {
  int tezina;
public:
  Vozilo(int tezina) : tezina(tezina) {}
  int DajTezinu() const { return tezina; }
  virtual int DajUkupnuTezinu() const { return tezina; }
  virtual ~Vozilo() {}
};
class PutnickoVozilo : public Vozilo {
  vector<int> putnici;
public:
  PutnickoVozilo(int tezina, vector<int> putnici) : Vozilo(tezina),
    putnici(putnici) {}
  int DajUkupnuTezinu() const;
};
int PutnickoVozilo::DajUkupnuTezinu() const {
  int ukupna_tezina(DajTezinu());
  for(int i = 0; i < putnici.size(); i++) ukupna_tezina += putnici[i];
  return ukupna_tezina;
}
class TeretnoVozilo : public Vozilo {
  int teret;
public:
  TeretnoVozilo(int tezina, int teret) : Vozilo(tezina), teret(teret) {}
  int DajUkupnuTezinu() { return DajTezinu() + teret; }
};
bool KriterijSortiranja(Vozilo *v1, Vozilo *v2) {
  return v1->DajUkupnuTezinu() < v2->DajUkupnuTezinu();
}
int main() {
  ifstream datoteka("VOZILA.TXT");
  vector<Vozilo*> vozila;
  for(;;) {
    int tezina, teret, broj_putnika;
    char vrsta;
    datoteka >> vrsta >> tezina;
    if(!datoteka) break;
    if(vrsta == 'V') vozila.push_back(new Vozilo(tezina));
    else if(vrsta == 'P') {
      datoteka >> broj_putnika;
      vector<int> putnici(broj_putnika);
      for(int i = 0; i < broj_putnika; i++) datoteka >> putnici[i];
      vozila.push_back(new PutnickoVozilo(tezina, putnici));
    }
    else {
      datoteka >> teret;
      vozila.push_back(new TeretnoVozilo(tezina, teret));
    }
  }
  sort(vozila.begin(), vozila.end(), KriterijSortiranja);
  for(int i = 0; i < vozila.size(); i++)
    cout << vozila[i]->DajUkupnuTezinu() << endl;
  for(int i = 0; i < vozila.size(); i++) delete vozila[i];
  return 0;
}
```

II PARCIJALNI ISPIT IZ PREDMETA "TEHNIKE PROGRAMIRANJA" (GRUPA B)

NAPOMENA: *Sve funkcije čija je implementacija duža od dvije naredbe obavezno implementirajte izvan klase. Također, sve metode koje su inspektori obavezno deklarirajte kao takve.*

Zadatak 1 (6 poena):

Svima koji se bave tehničkim naukama dobro su poznati komplesni brojevi i njihova svojstva, koji se mogu formalno posmatrati kao uređeni parovi realnih brojeva oblika (a, b) gdje su a i b realni brojevi. Međutim, iako su kompleksni brojevi najpoznatije i najprimjenljivije tvorevine takve vrste, one nisu i jedini korisni matematički objekti koji se modeliraju kao parovi realnih brojeva. U posljednje vrijeme, velika pažnja se posvećuje tzv. *dualnim brojevima*. Mada su prvobitno nastali kao eksperiment unutar linearne algebre, našli su izuzetno veliku primjenu upravo u računarstvu i to u situacijama kada treba isprogramirati numeričke algoritme u kojima se javlja potreba za računanjem izvoda funkcija (ovo je odlična tema za završni rad za one koji budu zainteresirani). Ovi neobični brojevi našli su primjenu i u nekim oblastima fizike, recimo za potrebe proučavanja dinamike krutih tijela i za razne perturbacione metode u fizici. Intuitivno, dualni brojevi su brojevi oblika $x + y\,\varepsilon$, gdje su x i y realni brojevi, dok je ε tzv. *nilpotenta*, donekle bizaran objekat za koji vrijedi $\varepsilon^2 = 0$ mada je $\varepsilon \neq 0$ (postojanje ovakvih bizarnih objekata može se lako opravdati pomoću linearne algebre). Formalno, dualni brojevi se mogu također predstaviti kao parovi realnih brojeva, koje ćemo označiti sa $\langle x\,'\,y\rangle$ da bismo ih razlikovali od kompleksnih brojeva (x, y), i za koje vrijede sljedeća pravila računanja:

$$\langle x_1\,'\,y_1\rangle \pm \langle x_2\,'\,y_2\rangle = \langle x_1 \pm x_2\,'\,y_1 \pm y_2\rangle \qquad \langle x_1\,'\,y_1\rangle \cdot \langle x_2\,'\,y_2\rangle = \langle x_1 x_2\,'\,x_1 y_2 + x_2 y_1\rangle$$

$$\langle x_1\,'\,y_1\rangle / \langle x_2\,'\,y_2\rangle = \left\langle \frac{x_1}{x_2}\,,\ \frac{x_2 y_1 - x_1 y_2}{{x_2}^2} \right\rangle$$

Može se primijetiti izvjesna sličnost sa računom sa kompleksnim brojevima, uz gubljenje pojedinih članova (koje su posljedica činjenice da je ε^2 jednako 0, dok je i^2 jednako –1). Uz ovakav formalizam, imamo $\varepsilon = \langle 0\,'1\rangle$. Broj x se naziva *vidljivi*, a broj y *skriveni* dio dualnog broja $\langle x\,'\,y\rangle$. Negacija dualnog broja $\langle x\,'\,y\rangle$ data je kao $\langle -x\,'\,-y\rangle$, dok je njegova konjugacija data kao $\langle x\,'\,-y\rangle$. Apsolutna vrijednost dualnog broja $\langle x\,'\,y\rangle$ data je kao

$$|\langle x\,'\,y\rangle| = |x|$$

Drugim riječima, skriveni dio dualnog broja ne figurira u njegovoj apsolutnoj vrijednosti. Dualni broj $\langle x\,'\,0\rangle$ može se poistovijetiti sa realnim brojem x. Dva dualna broja $\langle x_1\,'\,y_1\rangle$ i $\langle x_2\,'\,y_2\rangle$ jednaki su ako i samo ako je $x_1 = x_2$ i $y_1 = y_2$. Poredak dualnih brojeva se ne definira.

Definirajte *klasu za rad sa dualnim brojevima*. Klasa bi trebala da sadrži konstruktor koji kreira dualni broj na osnovu zadanog vidljivog i skrivenog dijela koji se zadaju kao parametri. Oba parametra trebaju imati podrazumijevanu vrijednost 0, tako da će se ovaj konstruktor moći koristiti i kao konstruktor bez parametara odnosno sa jednim parametrom. Klasa treba podržavati dvije pristupne metode bez parametara, kojima se pristupa vidljivom i skrivenom dijelu dualnog broja. Dalje, treba podržavati operatore "`+`", "`-`", "`*`" i "`/`" za obavljanje četiri osnovne računske operacije, zatim kombinovane operatore "`+=`", "`-=`", "`*=`" i "`/=`" sa uobičajenim značenjem, relacione operatore "`==`" i "`!=`", unarni operator negacije "`-`", te unarni operator "`++`" koji povećava realni dio dualnog broja za jedinicu, dok skriveni dio ostaje isti (potrebno je podržati i prefiksnu i postfiksnu verziju ovog operatora). Također je potrebno podržati i operatore "`<<`" i "`>>`" za ispis dualnih brojeva na izlazni tok i njihov unos sa ulaznog toka. Perpleksne brojeve treba ispisivati kao parove oblika $\langle x\,'\,y\rangle$ bez obzira da li su im neke komponente nule ili nisu, dok unos dualnih brojeva sa ulaznog toka treba podržati bilo kao par oblika $\langle x\,'\,y\rangle$, bilo kao običan realan broj, koji se potom interpretira kao dualni broj. Konačno, treba implementirati i četiri obične funkcije (ne članice) koje kao rezultat daju respektivno apsolutnu vrijednost, konjugaciju, vidljivi i skriveni dio dualnog broja (neka Vas ne zbunjuje što postoje i metode koje daju vidljivi i skriveni dio dualnog broja – ovako se omogućava veća sintaksna šarolikost u primjeni).

Rješenje:

```
class Dualni {
  double vid, skr;
public:
  Dualni(double vidljivi = 0, double skriveni = 0) : vid(vidljivi),
    skr(skriveni) {}
  double DajVidljivi() const { return vid; }
  double DajSkriveni() const { return skr; }
  friend Dualni operator +(const Dualni &d1, const Dualni &d2) {
    return Dualni(d1.vid + d2.vid, d1.skr + d2.skr);
  }
  friend Dualni operator -(const Dualni &d1, const Dualni &d2) {
    return Dualni(d1.vid - d2.vid, d1.skr - d2.skr);
  }
  friend Dualni operator *(const Dualni &d1, const Dualni &d2) {
    return Dualni(d1.vid * d2.vid, d1.vid * d2.skr + d2.vid * d1.skr);
  }
  friend Dualni operator /(const Dualni &d1, const Dualni &d2) {
    return Dualni(d1.vid / d2.vid, (d2.vid * d1.skr
      - d1.vid * d2.skr) / (d2.skr * d2.skr));
  }
  Dualni &operator +=(const Dualni &d) { return *this = *this + d; }
  Dualni &operator -=(const Dualni &d) { return *this = *this - d; }
  Dualni &operator *=(const Dualni &d) { return *this = *this * d; }
  Dualni &operator /=(const Dualni &d) { return *this = *this / d; }
  friend bool operator ==(const Dualni &d1, const Dualni &d2) {
    return d1.vid == d2.vid && d1.skr == d2.skr;
  }
  friend bool operator !=(const Dualni &d1, const Dualni &d2) {
    return !(d1 == d2);
  }
  Dualni operator -() { return Dualni(-vid, -skr); }
  Dualni &operator ++() { vid++; return *this; }
  Dualni operator ++(int) { Dualni d(*this); vid++; return d; }
  friend ostream &operator <<(ostream &cout, const Dualni &d) {
    return cout << "(" << d.vid << "|" << d.skr << ")";
  }
  friend istream &operator >>(istream &cin, Dualni &d);
  friend double ApsolutnaVrijednost(const Dualni &d) { return fabs(d.vid); }
  friend Dualni Konjugirani(const Dualni &d) { return Dualni(d.vid, -d.skr); }
  friend double VidljiviDio(const Dualni &d) { return d.vid; }
  friend double SkriveniDio(const Dualni &d) { return d.skr; }
};
istream &operator >>(istream &cin, Dualni &d) {
  if(cin.peek() != '<') {
    cin >> d.vid; d.skr = 0;
  }
  else {
    char znak;
    cin >> znak;
    if(znak != '<') cin.setstate(ios::failbit);
    cin >> d.vid >> znak;
    if(znak != '\'') cin.setstate(ios::failbit);
    cin >> d.skr >> znak;
    if(znak != '>') cin.setstate(ios::failbit);
  }
  return cin;
}
```

Zadatak 2 (8 poena):

Neka trgovačka firma je odlučila da napravi interni telefonski imenik svojih klijenata. Podaci o jednom klijentu čuvaju se u strukturi koja kao svoje atribute sadrži ime klijenta zajedno sa prezimenom (tipa niza od max. 20 znakova), grad u kojem se klijent nalazi (tipa niza od max. 10 znakova) i broj telefona (cjelobrojnog tipa). Potrebno je razviti kontejnersku klasu koja će čuvati

kolekciju podataka o klijentima i njihovim telefonskim brojevima. Ovi podaci će se alocirati dinamički, a pristupaće im se preko dinamički alociranog niza pokazivača koji pokazuju na ove podatke. Interfejs klase treba sadržavati sljedeće elemente:

a) Konstruktor sa jednim parametrom koji vrši neophodnu dinamičku alokaciju, pri čemu parametar predstavlja maksimalni broj uposlenika koji se mogu evidentirati. Pri tome se ovaj konstruktor ne smije koristiti za automatsku konverziju cjelobrojnih podataka u tip ove klase.
b) Destruktor, koji oslobađa sve resurse koje su primjerci ove klase zauzeli tokom svog života.
c) Konstruktor kopije i preklopljeni operator dodjele koji omogućavaju da se primjerci ove klase mogu bezbjedno kopirati i međusobno dodjeljivati.
d) Metodu koja dodaje u kolekciju podatke o novom klijentu, pri čemu su parametri metode podaci o klijentu (ime i prezime, grad i broj telefona). U slučaju da su podaci o imenu i prezimenu ili gradu predugi, ili ukoliko se dostigne maksimalni broj klijenata koji se mogu evidentirati, treba baciti izuzetak.
e) Metodu koja ispisuju podatke o klijentu čije se ime i prezime zadaje kao parametar i metodu koja ispisuje podatke o klijentu čiji se broj telefona zadaje kao parametar.
f) Metodu koja ispisuje telefonski imenik za sve klijente koji žive u gradu koje se zadaje kao parametar, zatim metodu koja ispisuje telefonski imenik za sve klijente čije ime počinje slovom koje se zadaje kao parametar, te metodu koja ispisuje čitav telefonski imenik.
g) Metodu koje sortira telefonski imenik po abecednom poretku klijenata (koristiti funkciju "`sort`").
h) Preklopljeni operator "`[]`" koji vraća referencu na broj telefona klijenta čije se ime zadaje unutar uglastih zagrada (referenca se vraća sa ciljem da se omogući izmjena broja). U slučaju da se ovaj operator primijeni na konstantni objekat, umjesto reference treba vratiti kopiju broja.
i) Metodu koja snima čitav telefonski imenik u binarnu datoteku čije se ime zadaje kao parametar, te konstruktor sa jednim parametrom koji prilikom kreiranja imenika obnavlja njegov sadržaj iz binarne datoteke čije se ime zadaje kao parametar konstruktora.

Rješenje:

Napomena: U prikazanom rješenju, struktura "`Klijent`" umjesto da se deklarira izvan klase na globalnom nivou, deklarirana je lokalno u privatnom dijelu klase, tako da joj samo metode klase "`Imenik`" mogu pristupiti, odnosno ostatak programa "ne zna" za postojanje ove strukture. Mada se to nije očekivalo od studenta da uradi, ovdje je prikazano čisto kao ilustracija da se takve tehnike često koriste.

```
class Imenik {
  struct Klijent {
    char ime[20], grad[10];
    int telefon;
  };
  int kapacitet, broj_klijenata;
  Klijent **klijenti;
  static void IspisiPodatke(Klijent *k); // Pomoćna funkcija za ispis
  static bool KriterijSortiranja(Klijent *k1, Klijent *k2) {
    return strcmp(k1->ime, k2->ime) < 0;
  }
public:
  explicit Imenik(int kapacitet) : kapacitet(kapacitet), broj_klijenata(0),
    klijenti(new Klijent*[kapacitet]) {}
  ~Imenik();
  Imenik(const Imenik &im);
  Imenik &operator=(const Imenik &im);
  void DodajKlijenta(const char ime[], const char grad[], int telefon);
  void IspisiKlijentaSaImenom(const char ime[]) const;
  void IspisiKlijentaSaTelefonom(int telefon) const;
  void IspisiImenikZaGrad(const char grad[]) const;
  void IspisiImenikNaSlovo(char slovo) const;
  void IspisiCitavImenik() const;
  void SortirajImenik() {
    sort(klijenti, klijenti + broj_klijenata, KriterijSortiranja);
  }
  int &operator[](const char ime[]);
  int operator[](const char ime[]) const;
```

```
  void Sacuvaj(const char ime_datoteke[]);
  explicit Imenik(const char ime_datoteke[]);
};
Imenik::~Imenik() {
  for(int i = 0; i < broj_klijenata; i++) delete klijenti[i];
  delete[] klijenti;
}
Imenik::Imenik(const Imenik &im) : broj_klijenata(im.broj_klijenata),
  kapacitet(im.kapacitet), klijenti(new Klijent*[im.kapacitet]) {
    for(int i = 0; i < broj_klijenata; i++)
      klijenti[i] = new Klijent(*im.klijenti[i]);
}
Imenik &Imenik::operator =(const Imenik &im) {
  if(&im == this) return *this;
  for(int i = 0; i < broj_klijenata; i++) delete klijenti[i];
  delete[] klijenti;
  broj_klijenata = im.broj_klijenata; kapacitet = im.kapacitet;
  klijenti = new Klijent*[kapacitet];
  for(int i = 0; i < broj_klijenata; i++)
    klijenti[i] = new Klijent(*im.klijenti[i]);
}
void Imenik::DodajKlijenta(const char ime[], const char grad[], int telefon) {
  if(broj_klijenata == kapacitet) throw "Popunjen kapacitet!";
  Klijent *novi(new Klijent);
  strcpy(novi->ime, ime); strcpy(novi->grad, grad);
  novi->telefon = telefon;
  klijenti[broj_klijenata++] = novi;
}
void Imenik::IspisiPodatke(Klijent *k) {
  cout << "Ime i prezime: " << k->ime << endl << "Grad: " << k->grad << endl
    << "Broj telefona: " << k->telefon << endl;
}
void Imenik::IspisiKlijentaSaImenom(const char ime[]) const {
  int brojac(0);
  for(int i = 0; i < broj_klijenata; i++)
    if(strcmp(klijenti[i]->ime, ime) == 0) {
      IspisiPodatke(klijenti[i]); brojac++;
    }
  if(brojac == 0) throw "Nema klijenata sa tim imenom!";
}
void Imenik::IspisiKlijentaSaTelefonom(int telefon) const {
  int brojac(0);
  for(int i = 0; i < broj_klijenata; i++)
    if(klijenti[i]->telefon == telefon) {
      IspisiPodatke(klijenti[i]); brojac++;
    }
  if(brojac == 0) throw "Nema klijenata sa tim brojem telefona!";
}
void Imenik::IspisiImenikZaGrad(const char grad[]) const {
  for(int i = 0; i < broj_klijenata; i++)
    if(strcmp(klijenti[i]->grad, grad) == 0) IspisiPodatke(klijenti[i]);
}
void Imenik::IspisiImenikNaSlovo(char slovo) const {
  for(int i = 0; i < broj_klijenata; i++)
    if(klijenti[i]->ime[0] == slovo) IspisiPodatke(klijenti[i]);
}
void Imenik::IspisiCitavImenik() const {
  for(int i = 0; i < broj_klijenata; i++) IspisiPodatke(klijenti[i]);
}
int &Imenik::operator[](const char ime[]) {
  for(int i = 0; i < broj_klijenata; i++)
    if(strcmp(klijenti[i]->ime, ime) == 0) return klijenti[i]->telefon;
  throw "Nema klijenata sa tim imenom!";
}
```

```
int Imenik::operator[](const char ime[]) const {
  for(int i = 0; i < broj_klijenata; i++)
    if(strcmp(klijenti[i]->ime, ime) == 0) return klijenti[i]->telefon;
  throw "Nema klijenata sa tim imenom!";
}
void Imenik::Sacuvaj(const char ime_datoteke[]) {
  ofstream datoteka(ime_datoteke);
  if(!datoteka) throw "Kreiranje datoteke nije uspjelo!";
  datoteka.write((char*)this, sizeof(*this));
  for(int i = 0; i < broj_klijenata; i++)
    datoteka.write((char*)klijenti[i], sizeof(**klijenti));
  if(!datoteka) throw "Snimanje nije uspjelo!";
}
Imenik::Imenik(const char ime_datoteke[]) {
  ifstream datoteka(ime_datoteke);
  if(!datoteka) throw "Datoteka ne postoji!";
  datoteka.read((char*)this, sizeof(*this));
  klijenti = new Klijent*[kapacitet];
  for(int i = 0; i < broj_klijenata; i++) {
    klijenti[i] = new Klijent;
    datoteka.read((char*)klijenti[i], sizeof(**klijenti));
  }
  if(!datoteka) throw "Citanje nije uspjelo!";
}
```

Zadatak 3 (6 poena):

Neka je *vozilo* objekat koji je, između ostalog, karakteriziran svojom težinom. *Automobil* je specijalna vrsta vozila, koje može primiti određeni broj putnika od kojih svaki ima svoju težinu. *Kamion* je specijalna vrsta vozila koje se može nakrcati teretom određene težine. Razvijte hijerarhiju klasa koje opisuju ove objekte. Bazna klasa "`Vozilo`" posjedovaće atribut koji predstavlja težinu vozila (tipa cijeli broj), konstruktor koji inicijalizira ovaj atribut, te dvije metode nazvane "`DajTezinu`" i "`DajUkupnuTezinu`". Prva metoda daje vlastitu težinu vozila, dok druga metoda daje ukupnu težinu vozila u koju je uračunata i težina svega što se u vozilu nalazi (u slučaju bazne klase "`Vozilo`" obje metode će vraćati istu vrijednost). Klasa "`Automobil`" nasljeđuje se iz klase "`Vozilo`", a posjeduje dodatni atribut koji je tipa vektor cijelih brojeva, koji čuva težine putnika u vozilu. Konstruktor ove klase ima dodatni parametar (vektor težina putnika) i izmijenjenu metodu "`DajUkupnuTezinu`", koja uračunava težine putnika. Klasa "`Kamion`" se također nasljeđuje iz klase "`Vozilo`", a posjeduje dodatni cjelobrojni atribut koji predstavlja težinu tereta. Ova klasa također ima dodatni parametar u konstruktoru (težina tereta) i izmijenjenu metodu za računanje ukupne težine. Metoda "`DajUkupnuTezinu`" mora biti izvedena tako da ukoliko se svim opisanim objektima pristupa preko pokazivača na baznu klasu "`Vozilo`", uvijek bude pozvana ispravna verzija metode, koja će uzeti u obzir specifičnosti objekta.

Napisane klase demonstrirajte u testnom programu koji čita podatke o vozilima iz tekstualna datoteke u vektor, sortira vozila po ukupnoj težini u rastući poredak i na kraju ispisuje težine vozila nakon sortiranja (elementi vektora će biti tipa pokazivači na "`Vozilo`" da bi se mogao koristiti polimorfizam). Svaki red datoteke sadrži podatke o jednom vozilu, poput "`V500`" za obično vozilo težine 500, "`A500 3 80 60 75`" za automobil težine 500 sa 3 putnika težina 80, 60 i 75 respektivno, te "`K500 1200`" za kamion težine 500 natovaren sa teretom težine 1200. Pretpostavite da datoteka sadrži ispravne podatke.

Rješenje:

```
class Vozilo {
  int tezina;
public:
  Vozilo(int tezina) : tezina(tezina) {}
  int DajTezinu() const { return tezina; }
  virtual int DajUkupnuTezinu() const { return tezina; }
  virtual ~Vozilo() {}
};
```

```
class Automobil : public Vozilo {
  vector<int> putnici;
public:
  Automobil(int tezina, vector<int> putnici) : Vozilo(tezina),
    putnici(putnici) {}
  int DajUkupnuTezinu() const;
};
int Automobil::DajUkupnuTezinu() const {
  int ukupna_tezina(DajTezinu());
  for(int i = 0; i < putnici.size(); i++) ukupna_tezina += putnici[i];
  return ukupna_tezina;
}
class Kamion : public Vozilo {
  int teret;
public:
  Kamion(int tezina, int teret) : Vozilo(tezina), teret(teret) {}
  int DajUkupnuTezinu() { return DajTezinu() + teret; }
};
bool KriterijSortiranja(Vozilo *v1, Vozilo *v2) {
  return v1->DajUkupnuTezinu() < v2->DajUkupnuTezinu();
}
int main() {
  ifstream datoteka("VOZILA.TXT");
  vector<Vozilo*> vozila;
  for(;;) {
    int tezina, teret, broj_putnika;
    char vrsta;
    datoteka >> vrsta >> tezina;
    if(!datoteka) break;
    if(vrsta == 'V') vozila.push_back(new Vozilo(tezina));
    else if(vrsta == 'A') {
      datoteka >> broj_putnika;
      vector<int> putnici(broj_putnika);
      for(int i = 0; i < broj_putnika; i++) datoteka >> putnici[i];
      vozila.push_back(new Automobil(tezina, putnici));
    }
    else {
      datoteka >> teret;
      vozila.push_back(new Kamion(tezina, teret));
    }
  }
  sort(vozila.begin(), vozila.end(), KriterijSortiranja);
  for(int i = 0; i < vozila.size(); i++)
    cout << vozila[i]->DajUkupnuTezinu() << endl;
  for(int i = 0; i < vozila.size(); i++) delete vozila[i];
  return 0;
}
```