POPRAVNI ISPIT IZ PREDMETA "TEHNIKE PROGRAMIRANJA" (I PARCIJALNI, GRUPA A)

Zadatak 1. (4 poena)

Utvrdite šta će ispisati sljedeći program (odgovor treba biti obrazložen):

```
#include <iostream>
using namespace std;
int &f(int &a, int *b, int c) {
  c *= 20;
  a = *b *a + c;
  int &d = *b;
  a++;
  return d;
}
int main() {
  int a, b, c;
  a = b = c = 30;
  a += ++b;
  f(c, &b, a) = 10;
  cout << a << endl << b << endl << c << endl;
  return 0;
}
```

Rješenje:

```
61
10
2151
```

Zadatak 2. (3 poena)

Napišite funkciju koja prima vektor od n realnih brojeva a_1, a_2, ... a_n kao parametar, i koja računa i vraća kao rezultat vrijednost izraza

$$\sqrt{a_1+\sqrt{a_2+\sqrt{...+\sqrt{a_n}}}}$$

U slučaju da su elementi takvi da rezultat nije realan broj, funkcija treba baciti izuzetak. Vodite računa da u jeziku C++ indeksi vektora idu od 0, a ne od 1!

Rješenje:

```
double KaskadaKorijena(const vector<double> &v) {
  double vrijednost(0);
  for(int i = v.size() - 1; i >= 0; i--) {
    if(v[i] + vrijednost < 0) throw "Negativan broj pod korijenom!\n";
    vrijednost = sqrt(v[i] + vrijednost);
  }
  return vrijednost;
}
```

Zadatak 3. (3 poena)

Napišite funkciju "Sredine" koja treba da ima 3 parametra "N", "AS" i "HS". Funkcija treba da izračuna aritmetičku i harmonijsku sredinu svih cifara u parametru "N" koji je prirodan broj, i da smjesti izračunate vrijednosti u parametre "AS" i "HS" respektivno (podsjetimo se da se harmonijska sredina za n brojeva definira kao recipročna vrijednost aritmetičke sredine njihovih recipročnih vrijednosti). Napišite i mali isječak programa u kojem ćete demonstrirati kako se upotrebljava napisana funkcija.

Rješenje:

```
void Sredine(int N, double &AS, double &HS) {
  int broj_cifara(0);
  double suma1(0), suma2(0);
  while(N > 0) {
    broj_cifara++;
    suma1 += N % 10; suma2 += 1. / (N % 10);
    N /= 10;
  }
  AS = suma1 / broj_cifara; HS = broj_cifara / suma2;
}
...
double a_s, h_s;
Sredine(134215,a_s, h_s);
cout << a_s << endl << h_s << endl;
```

Zadatak 4. (3 poena)

Za neki element a_n nekog niza brojeva kaže se da je *lokalni maksimum* tog niza brojeva ukoliko je veći od oba svoja susjeda, tj. ukoliko je $a_n > a_{n-1}$ i $a_n > a_{n+1}$. Napišite generičku funkciju koja prima dva parametra, od kojih je prvi parametar niz elemenata proizvoljnog tipa, dok je drugi parametar broj elemenata u tom nizu. Funkcija treba da kao rezultat vrati broj lokalnih maksimuma u tom nizu. Napišite i isječak programa u kojem ćete demonstrirati kako se napisana funkcija može primijeniti na jednom fiksno zadanom nizu od 10 cijelih brojeva.

Rješenje:

```
template <typename UporediviTip>
int BrojLokalnihMaksimuma(UporediviTip niz[], int broj_elemenata) {
  int brojac(0);
  for(int i = 1; i < broj_elemenata - 1; i++)
    if(niz[i] > niz[i - 1] && niz[i] > niz[i + 1]) brojac++;
  return brojac;
}
...
int a[] = {3, 5, 2, 1, 8, 7, 9, 4, 2, 3};
cout << BrojLokalnihMaksimuma(a, 10);
```

Zadatak 5. (3 poena)

Poznato je da se izvod neke funkcije f u tački x može približno izračunati pomoću formule

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

ukoliko je h dovoljno mala vrijednost (tačna vrijednost izvoda dobija se za $h \to 0$). Napišite funkciju "`Izvod`" sa 3 parametra "`f`", "`x`" i "`h`" koja računa približnu vrijednost izvoda funkcije "`f`" u tački "`x`". Pri tome je parametar "`f`" funkcija koja prima realni argument i daje realan rezultat, dok su "`x`" i "`h`" realne vrijednosti. Pri tome, parametar "`h`" se može izostaviti, i tada se uzima podrazumijevana vrijednost $h = 10^{-5}$. Također demonstrirajte kako biste napisanu funkciju "`Izvod`" upotrijebili da izračunate približnu vrijednost izvoda funkcije *sin* u tački $x = 0$.

Rješenje:

```
double Izvod(double f(double), double x, double h = 1e-5) {
  return (f(x + h) - f(x)) / h;
}
...
cout << Izvod(sin, 0);
```

Zadatak 6. (4 poena)

Napišite funkciju sa jednim cjelobrojnim parametrom *n* koja dinamički alocira niz od *n* cijelih brojeva, popunjava ga sa prvih *n* prostih brojeva, i vraća pokazivač na prvi element tako alociranog niza. Ukoliko alokacija ne uspije ili ukoliko je *n* negativan ili nula, funkcija treba baciti izuzetak. Napisanu funkciju demonstrirajte u isječku programa koji ilustrira kako biste mogli pozvati ovu funkciju i osloboditi alociranu memoriju kada ona više nije potrebna. Predvidite i hvatanje eventualno bačenih izuzetaka.

Rješenje:

Predloženo rješenje zasniva se na činjenici da je za testiranje prostosti broja dovoljno ispitati njegovu djeljivost sa prostim brojevima manjim od njega, koji su već nađeni. Jasno je da postoje i drugačija, ali manje efikasna rješenja (postoje i još efikasnija rješenja, ali su ona mnogo složenija):

```
int *AlocirajNizProstih(int broj_elemenata) {
  if(broj_elemenata <= 0) throw "Nekorektan parametar!\n";
  int *niz;
  try {
    niz = new int[broj_elemenata];
  }
  catch(...) {
    throw "Alokacija nije uspjela!\n";
  }
  int brojac(1), tekuci_broj(3);
  niz[0] = 2;
  while(brojac <= broj_elemenata) {
    bool prost(true);
    for(int i = 0; i < brojac; i++)
      if(tekuci_broj % niz[i] == 0) prost = false;
    if(prost) niz[brojac++] = tekuci_broj;
    tekuci_broj += 2;
  }
  return niz;
}
...
try {
  int *niz = AlocirajNizProstih(10);
  for(int i = 0; i < 10; i++) cout << niz[i] << endl;
  delete[] niz;
}
catch(const char poruka[]) {
  cout << poruka;
}
```

POPRAVNI ISPIT IZ PREDMETA "TEHNIKE PROGRAMIRANJA" (I PARCIJALNI, GRUPA B)

Zadatak 1. (4 poena)

Utvrdite šta će ispisati sljedeći program (odgovor treba biti obrazložen):

```
#include <iostream>
using namespace std;
int &f(int &a, int *b, int c) {
  c *= 10;
  a = *b *a + c;
  int &d = *b;
  a++;
  return d;
}
int main() {
  int a, b, c;
  a = b = c = 20;
  a += ++b;
  f(c, &b, a) = 30;
  cout << a << endl << b << endl << c << endl;
  return 0;
}
```

Rješenje:

```
41
30
831
```

Zadatak 2. (3 poena)

Napišite funkciju koja prima vektor od n realnih brojeva a_1, a_2, ... a_n kao parametar, i koja računa i vraća kao rezultat vrijednost izraza

$$\frac{1}{a_1}+\frac{1}{a_1+a_2}+\frac{1}{a_1+a_2+a_3}+\ldots+\frac{1}{a_1+a_2+a_3+\ldots+a_n}$$

U slučaju da su elementi takvi da se neki od nazivnika anulira, funkcija treba baciti izuzetak. Vodite računa da u jeziku C++ indeksi vektora idu od 0, a ne od 1!

Rješenje:

```
double SumaRazlomaka(const vector<double> &v) {
  double suma(0), nazivnik(0);
  for(int i = 0; i < v.size(); i++) {
    nazivnik += v[i];
    if(nazivnik == 0) throw "Negativan broj pod korijenom!\n";
    suma += 1 / nazivnik;
  }
  return suma;
}
```

Zadatak 3. (3 poena)

Napišite funkciju "Sredine" koja treba da ima 3 parametra "N", "AS" i "GS". Funkcija treba da izračuna aritmetičku i geometrijsku sredinu svih cifara u parametru "N" koji je prirodan broj, i da smjesti izračunate vrijednosti u parametre "AS" i "GS" respektivno (podsjetimo se da se geometrijska sredina za n brojeva definira kao n-ti korijen iz njihovog produkta). Napišite i mali isječak programa u kojem ćete demonstrirati kako se upotrebljava napisana funkcija.

Rješenje:

```
void Sredine(int N, double &AS, double &GS) {
  int broj_cifara(0);
  double suma(0), produkt(1);
  while(N > 0) {
    broj_cifara++;
    suma += N % 10; produkt *= N % 10;
    N /= 10;
  }
  AS = suma / broj_cifara; GS = pow(produkt, 1. /broj_cifara);
}
...
double a_s, g_s;
Sredine(134215,a_s, g_s);
cout << a_s << endl << g_s << endl;
```

Zadatak 4. (3 poena)

Za neki element a_n nekog niza brojeva kaže se da je *lokalni minimum* tog niza brojeva ukoliko je manji od oba svoja susjeda, tj. ukoliko je $a_n < a_{n-1}$ i $a_n < a_{n+1}$. Napišite generičku funkciju koja prima dva parametra, od kojih je prvi parametar niz elemenata proizvoljnog tipa, dok je drugi parametar broj elemenata u tom nizu. Funkcija treba da kao rezultat vrati broj lokalnih minimuma u tom nizu. Napišite i isječak programa u kojem ćete demonstrirati kako se napisana funkcija može primijeniti na jednom fiksno zadanom nizu od 10 realnih brojeva.

Rješenje:

```
template <typename UporediviTip>
int BrojLokalnihMinimuma(UporediviTip niz[], int broj_elemenata) {
  int brojac(0);
  for(int i = 1; i < broj_elemenata - 1; i++)
    if(niz[i] < niz[i - 1] && niz[i] < niz[i + 1]) brojac++;
  return brojac;
}
...
double a[] = {3, 5, 2, 1, 8, 7, 9, 4, 2, 3};
cout << BrojLokalnihMinimuma(a, 10);
```

Zadatak 5. (3 poena)

Poznato je da se izvod neke funkcije *f* u tački *x* može približno izračunati pomoću formule

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

ukoliko je h dovoljno mala vrijednost (tačna vrijednost izvoda dobija se za $h \to 0$). Napišite funkciju "`Izvod`" sa 3 parametra "`f`", "`x`" i "`h`" koja računa približnu vrijednost izvoda funkcije "`f`" u tački "`x`". Pri tome je parametar "`f`" funkcija koja prima realni argument i daje realan rezultat, dok su "`x`" i "`h`" realne vrijednosti. Pri tome, parametar "`h`" se može izostaviti, i tada se uzima podrazumijevana vrijednost $h = 10^{-5}$. Također demonstrirajte kako biste napisanu funkciju "`Izvod`" upotrijebili da izračunate približnu vrijednost izvoda funkcije *sin* u tački $x = 0$.

Rješenje:

```
double Izvod(double f(double), double x, double h = 1e-5) {
  return (f(x + h) - f(x)) / h;
}
...
cout << Izvod(sin, 0);
```

Zadatak 6. (4 poena)

Napišite funkciju sa jednim cjelobrojnim parametrom *n* koja dinamički alocira niz od *n* cijelih brojeva, popunjava ga sa prvih *n* prostih brojeva, i vraća pokazivač na prvi element tako alociranog niza. Ukoliko alokacija ne uspije ili ukoliko je *n* negativan ili nula, funkcija treba baciti izuzetak. Napisanu funkciju demonstrirajte u isječku programa koji ilustrira kako biste mogli pozvati ovu funkciju i osloboditi alociranu memoriju kada ona više nije potrebna. Predvidite i hvatanje eventualno bačenih izuzetaka.

Rješenje:

Predloženo rješenje zasniva se na činjenici da je za testiranje prostosti broja dovoljno ispitati njegovu djeljivost sa prostim brojevima manjim od njega, koji su već nađeni. Jasno je da postoje i drugačija, ali manje efikasna rješenja (postoje i još efikasnija rješenja, ali su ona mnogo složenija):

```
int *AlocirajNizProstih(int broj_elemenata) {
  if(broj_elemenata <= 0) throw "Nekorektan parametar!\n";
  int *niz;
  try {
    niz = new int[broj_elemenata];
  }
  catch(...) {
    throw "Alokacija nije uspjela!\n";
  }
  int brojac(1), tekuci_broj(3);
  niz[0] = 2;
  while(brojac <= broj_elemenata) {
    bool prost(true);
    for(int i = 0; i < brojac; i++)
      if(tekuci_broj % niz[i] == 0) prost = false;
    if(prost) niz[brojac++] = tekuci_broj;
    tekuci_broj += 2;
  }
  return niz;
}
...
try {
  int *niz = AlocirajNizProstih(10);
  for(int i = 0; i < 10; i++) cout << niz[i] << endl;
  delete[] niz;
}
catch(const char poruka[]) {
  cout << poruka;
}
```

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

Za potrebe evidencije pacijenata koji dolaze na pregled kod ljekara, neki računarski program definira i implementira klase nazvane "`Pregled`" i "`Pregledi`". Klasa "`Pregled`" opisuje podatke o jednom zakazanom pregledu, dok klasa "`Pregledi`" predstavlja kolekciju podataka o zakazanim pregledima, odnosno objekata tipa "`Pregled`". Klasa "`Pregled`" sadrži sljedeće elemente:

- Atribute koji predstavljaju evidencijski broj pacijenta, ime i prezime pacijenta (tipa "`string`"), datum zakazanog pregleda (dan i mjesec), kao i vrijeme pregleda (sati i minute).
- Konstruktor koji inicijalizira sve atribute klase na vrijednosti zadane parametrima. Pri tome se testira da li su zadani legalan datum i vrijeme (ukoliko nisu, potrebno je baciti izuzetak). Radi jednostavnosti, pretpostavite da februar uvijek ima 28 dana.
- Trivijalne pristupne metode, kojima se omogućava čitanje privatnih atributa klase.
- Preklopljeni operator "<<" koji ispisuje podatke o pregledu na ekran (format ispisa odredite po volji).
- Preklopljeni relacioni operator "<" koji daje kao rezultat "**`true`**" ukoliko je prvi pregled zakazan prije drugog pregleda, a "**`false`**" u suprotnom slučaju.
- Preklopljeni operator "++" koji pomjera datum pregleda za jedan dan unaprijed. Potrebno je podržati i prefiksnu i postfiksnu verziju ovog operatora.
- Preklopljeni operator "+", pri čemu ukoliko je "`p`" neki objekat tipa "`Pregled`" a "`n`" nenegativan cijeli broj, "`p + n`" daje novi objekat tipa "`Pregled`" u kojem je datum pregleda pomjeren za "`n`" dana unaprijed (u nedostatku bolje ideje, možete primijeniti operator "++" u petlji). Ukoliko "`n`" nije nenegativan broj, treba baciti izuzetak.
- Preklopljeni operator "+=" koji obezbjeđuje da izraz poput "`p += n`" uvijek ima isto značenje kao i izraz "`p = p + n`".

Klasa "`Pregledi`" predstavlja kolekciju podataka o zakazanim pregledima, izvedenu kao dinamički niz pokazivača na objekte tipa "`Pregled`". Ova klasa sadrži sljedeće elemente:

- Atribute koji predstavljaju broj pohranjenih pregleda, maksimalni broj pregleda koji se mogu pohraniti, kao i pokazivač za pristup dinamičkom nizu pokazivača na pohranjene objekte.
- Konstruktor sa jednim parametrom, koji vrši odgovarajuću dinamičku alokaciju memorije, pri čemu parametar predstavlja maksimalan broj pregleda koji se mogu pohraniti. Pri tome je potrebno zabraniti da se ovaj konstruktor koristi za automatsku konverziju objekata tipa "**`int`**" u objekte tipa "`Pregledi`".
- Destruktor, koji oslobađa memoriju koju je objekat tipa "`Pregledi`" zauzeo tokom svog života.
- Konstruktor kopije i preklopljeni operator dodjele, koji obezbjeđuju sigurno kopiranje i međusobno dodjeljivanje primjeraka klase "`Pregledi`".
- Metodu koja vrši kreiranje i dodavanje podataka o novom pregledu, koja ima iste parametre kao i konstruktor klase "`Pregled`". Ova metoda kreira novi objekat koji sadrži podatke o zakazanom pregledu (u skladu sa navedenim parametrima) i smješta ga u kolekciju.
- Metodu bez parametara, koja sortira sve preglede u rastući redoslijed.
- Metodu bez parametara, koja vraća referencu na najraniji zakazani pregled (u slučaju da je kolekcija prazna, treba baciti izuzetak).
- Metodu bez parametara, koja uklanja iz kolekcije najraniji zakazani pregled (u slučaju da je kolekcija prazna, treba baciti izuzetak).
- Metodu koja ispisuje sve preglede koji su zakazani na zadani datum, pri čemu se dan i mjesec prosljeđuju kao parametri (ukoliko takvih pregleda nema, treba baciti izuzetak).
- Metode sa jednim parametrom koje snimaju sadržaj kolekcije u binarnu datoteku, odnosno obnavljaju sadržaj kolekcije iz binarne datoteke. Parametar ovih metoda predstavlja ime datoteke.

- Preklopljeni operator "<<" koji ispisuje sve pohranjene preglede (u slučaju da je kolekcija prazna, treba ispisati odgovarajući komentar).
- Preklopljeni unarni operator "`!`", koji primijenjen na objekat tipa "`Pregledi`" daje kao rezultat logičku vrijednost "**`true`**" ako i samo ako objekat predstavlja praznu kolekciju, a u suprotnom daje kao rezultat logičku vrijednost "**`false`**".
- Preklopljeni operator "`[]`" koji omogućava pristup pacijentima preko njihovog evidencijskog broja. Ovaj operator vraća referencu na pacijenta pohranjenog u kolekciji čiji je evidencijski broj naveden unutar uglastih zagrada. Ukoliko takvog pacijenta nema, treba baciti izuzetak.

Sve metode implementirajte izvan klase, osim metoda sa kratkom implementacijom, koje možete implementirati odmah unutar klase.

Napisane klase testirajte u testnom programu koji kreira nekoliko zahtjeva prema podacima unesenim sa tastature, a zatim ispisuje sve unijete zahtjeve, sortirane u rastući poredak.

Rješenje:

Prikazano rješenje prikazuje implementacije klasa "`Pregled`" i "`Pregledi`", bez glavnog programa. Pri tome se podrazumijeva da su sve neophodne biblioteke uključene u program.

```
class Pregled {
  int evidencijski_broj;
  string ime, prezime;
  int dan, mjesec, sati, minute;
  static int BrojDana(int mjesec);  // Pomoćna funkcija...
public:
  Pregled(int evidencijski_broj, string ime, string prezime, int dan,
    int mjesec, int sati, int minute);
  int DajEvidencijskiBroj() { return evidencijski_broj; }
  string DajIme() { return ime; }
  string DajPrezime() { return prezime; }
  int DajDan() { return dan; }
  int DajMjesec() { return mjesec; }
  int DajSate() { return sati; }
  int DajMinute() { return minute; }
  friend ostream &operator <<(ostream &cout, const Pregled &p);
  friend bool operator <(const Pregled &p1, const Pregled &p2);
  Pregled &operator ++();
  Pregled operator ++(int) { Pregled p(*this); ++(*this); return p; }
  friend Pregled operator +(const Pregled &p, int n);
  Pregled &operator +=(int n) { return *this = *this + n; }
};
int Pregled::BrojDana(int mjesec) {
  if(mjesec == 2) return 28;
  if(mjesec == 4 || mjesec == 6 || mjesec == 9 || mjesec == 11) return 30;
  return 31;
}
Pregled::Pregled(int evidencijski_broj, string ime, string prezime, int dan,
  int mjesec, int sati, int minute) : evidencijski_broj(evidencijski_broj),
  ime(ime), prezime(prezime) {
    if(dan < 1 || dan > BrojDana(mjesec) || mjesec < 1 || mjesec > 12 ||
      sati < 0 || sati > 23 || minute < 0 || minute > 59)
        throw "Neispravni podaci!\n";
    Pregled::dan = dan; Pregled::mjesec = mjesec;
    Pregled::sati = sati; Pregled::minute = minute;
}
ostream &operator <<(ostream &cout, const Pregled &p) {
  return cout << "Pacijent " << p.ime << " " << p.prezime
    << " (evidencijski broj " << p.evidencijski_broj
    << ") zakazao pregled dana " << p.dan << "." << p. mjesec << " u "
    << p.sati << ":" << p.minute;
}
```

*Redoslijed naredbi u izvedbi operatora "<" je jako bitan (uzeti u obzir da naredba "**return**" trenutno prekida izvođenje funkcije):*

```
bool operator <(const Pregled &p1, const Pregled &p2) {
  if(p1.mjesec < p2.mjesec) return true;
  if(p1.dan < p2.dan) return true;
  return p1.sati * 60 + p1.minute < p2.sati * 60 + p2.minute;
}

Pregled &Pregled::operator ++() {
  dan++;
  if(dan > BrojDana(mjesec)) {
    dan = 1; mjesec++;
    if(mjesec > 12) mjesec = 1;
  }
}
```

Prikazana izvedba operatora "+" predstavlja dobar kompromis između jednostavnosti i efikasnosti:

```
Pregled operator +(const Pregled &p, int n) {
  Pregled p1(p);
  p1.dan += n;
  while(p1.dan > Pregled::BrojDana(p1.mjesec)) {
    p1.dan -= Pregled::BrojDana(p1.mjesec); p1.mjesec++;
    if(p1.mjesec > 12) p1.mjesec = 1;
  }
  return p1;
}

class Pregledi {
  int broj_pregleda, max_broj_pregleda;
  Pregled **kolekcija;
  static bool Kriterij(const Pregled *p1, const Pregled *p2) {
    return *p1 < *p2;
  }
public:
  explicit Pregledi(int n) : broj_pregleda(0), max_broj_pregleda(n),
    kolekcija(new Pregled*[n]) {}
  ~Pregledi();
  Pregledi(const Pregledi &p);
  Pregledi &operator =(const Pregledi &p);
  void DodajPregled(int evidencijski_broj, string ime, string prezime,
    int dan, int mjesec, int sati, int minute);
  void Sortiraj() { sort(kolekcija, kolekcija + broj_pregleda, Kriterij); }
  Pregled &DajNajranijiPregled();
  void UkloniNajranijiPregled();
  void IspisiPregledeNaDatum(int dan, int mjesec);
  void Snimi(const char ime[]);
  void Obnovi(const char ime[]);
  friend ostream &operator <<(ostream &cout, const Pregledi &p);
  bool operator !() { return broj_pregleda == 0; }
  Pregled &operator [](int evidencijski_broj);
};

Pregledi::~Pregledi() {
  for(int i = 0; i < broj_pregleda; i++) delete kolekcija[i];
  delete[] kolekcija;
}

Pregledi::Pregledi(const Pregledi &p) : broj_pregleda(p.broj_pregleda),
  max_broj_pregleda(p.max_broj_pregleda),
  kolekcija(new Pregled*[p.broj_pregleda]) {
    for(int i = 0; i < broj_pregleda; i++)
      kolekcija[i] = new Pregled(*p.kolekcija[i]);
}
```

Izvedba operatora dodjele kakva je ovdje implementirana definitivno nije najbolja moguća, ali je vjerovatno najjednostavnija (strategija "uništi stari sadržaj odredišta, a nakon toga kopiraj"):

```
Pregledi &Pregledi::operator =(const Pregledi &p) {
  if(&p == this) return *this;
```

```
  for(int i = 0; i < broj_pregleda; i++) delete kolekcija[i];
  delete[] kolekcija;
  broj_pregleda = p.broj_pregleda;
  max_broj_pregleda = p.max_broj_pregleda;
  kolekcija = new Pregled*[broj_pregleda];
  for(int i = 0; i < broj_pregleda; i++)
    kolekcija[i] = new Pregled(*p.kolekcija[i]);
  return *this;
}
void Pregledi::DodajPregled(int evidencijski_broj, string ime,
  string prezime, int dan, int mjesec, int sati, int minute) {
    if(broj_pregleda == max_broj_pregleda) throw "Popunjen kapacitet!\n";
    kolekcija[broj_pregleda++] = new Pregled(evidencijski_broj, ime, prezime,
      dan, mjesec, sati, minute);
}
```

Metoda "DajNajranijiPregled" *zasnovana je na manipulaciji sa pokazivačima, s obzirom da je to efikasnije nego manipulacija sa objektima. Na kraju se vrši dereferenciranje sa ciljem da se vrati referirani objekat:*

```
Pregled &Pregledi::DajNajranijiPregled() {
  if(broj_pregleda == 0) throw "Nema zakazanih pregleda!\n";
  Pregled *najraniji(kolekcija[0]);
  for(int i = 1; i < broj_pregleda; i++)
    if(*kolekcija[i] < *najraniji) najraniji = kolekcija[i];
  return *najraniji;
}
```

Prikazana izvedba metode "UkloniNajranijiPregled" *oslanja se na izvedbu prethodne metode i činjenicu da se objekti tipa* "Pregled" *mogu međusobno dodjeljivati, čime je izvedba ispala veoma kratka. U prikazanom rješenju, brisanje je realizirano tako što se na mjesto objekta koji se briše dovodi objekat koji je prije bio na kraju kolekcije. Naravno, moguća su i druga rješenja:*

```
void Pregledi::UkloniNajranijiPregled() {
  DajNajranijiPregled() = *kolekcija[broj_pregleda - 1];
  delete kolekcija[--broj_pregleda];
}
void Pregledi::IspisiPregledeNaDatum(int dan, int mjesec) {
  bool ima_pregleda(false);
  for(int i = 0; i < broj_pregleda; i++)
    if(kolekcija[i]->DajDan() == dan && kolekcija[i]->DajMjesec()) {
      cout << *kolekcija[i] << endl;
      ima_pregleda = true;
    }
  if(!ima_pregleda) throw "Nema pregleda na traženi dan!\n";
}
ostream &operator <<(ostream &cout, const Pregledi &p) {
  if(p.broj_pregleda == 0) return cout << "Nema zakazanih pregleda!\n";
  for(int i = 0; i << p.broj_pregleda; i++) cout << *p.kolekcija[i] << endl;
  return cout;
}
Pregled &Pregledi::operator [](int evidencijski_broj) {
  for(int i = 0; i < broj_pregleda; i++)
    if(kolekcija[i]->DajEvidencijskiBroj() == evidencijski_broj)
      return *kolekcija[i];
  throw "Nije nadjen pacijent!\n";
}
```

Slijede izvedbe metoda "Snimi" *i* "Obnovi" *onako kako je to zamislio predmetni nastavnik. Nažalost, prikazane metode NE RADE, bez obzira što je predmetni nastavnik ovakva rješenja priznavao kao korektna. Naime, predmetni nastavnik je previdio činjenicu da tip* "string" *sadržan u klasi* "Pregled" *nije POD (Plain Old Data) tip podataka. Stoga bi izvedbe metoda* "Snimi" *i* "Obnovi" *koje zaista rade bile vrlo kompliciranе, i prosječan student ih ne bi bio u stanju napisati, tako da neće biti prikazane:*

```
void Pregledi::Snimi(const char ime[]) {
  ofstream izlaz(ime, ios::out | ios::binary);
```

```
  izlaz.write((char*)kolekcija, sizeof(Pregledi));
  for(int i = 0; i < broj_pregleda; i++)
    izlaz.write((char*)kolekcija[i], sizeof(Pregled));
  if(!izlaz) throw "Nešto nije u redu sa upisom!\n";
}
void Pregledi::Obnovi(const char ime[]) {
  ifstream ulaz(ime, ios::in | ios::binary);
  if(!ulaz) throw "Datoteka ne postoji!\n";
  for(int i = 0; i < broj_pregleda; i++) delete kolekcija[i];
  delete[] kolekcija;
  ulaz.read((char*)kolekcija, sizeof(Pregledi));
  for(int i = 0; i < broj_pregleda; i++) {
    kolekcija[i] = new Pregled(1, "", "", 1, 1, 1, 1);
    ulaz.read((char*)kolekcija[i], sizeof(Pregled));
  }
  if(!ulaz) throw "Obnova nije uspjela!\n";
}
```

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

Za potrebe evidencije klijenata koji traže intervencije kod servisera, neki računarski program definira i implementira klase nazvane "`Popravka`" i "`Popravke`". Klasa "`Popravka`" opisuje podatke o jednoj zakazanoj intervenciji (popravci), dok klasa "`Popravke`" predstavlja kolekciju podataka o zakazanim intervencijama, odnosno objekata tipa "`Popravka`". Klasa "`Popravka`" sadrži sljedeće elemente:

- Atribute koji predstavljaju ime i prezime klijenta (tipa "`string`"), vrstu popravke (također tipa "`string`"), datum zakazane intervencije (dan i mjesec), kao i vrijeme intervencije (sati i minute).
- Konstruktor koji inicijalizira sve atribute klase na vrijednosti zadane parametrima. Pri tome se testira da li su zadani legalan datum i vrijeme (ukoliko nisu, potrebno je baciti izuzetak). Radi jednostavnosti, pretpostavite da februar uvijek ima 28 dana.
- Trivijalne pristupne metode, kojima se omogućava čitanje privatnih atributa klase.
- Preklopljeni operator "<<" koji ispisuje podatke o intervenciji na ekran (format ispisa odredite po volji).
- Preklopljeni relacioni operator "<" koji daje kao rezultat "**`true`**" ukoliko je prva intervencija zakazaa prije druge intervencije, a "**`false`**" u suprotnom slučaju.
- Preklopljeni operator "`--`" koji pomjera datum popravke za jedan dan unazad. Potrebno je podržati i prefiksnu i postfiksnu verziju ovog operatora.
- Preklopljeni operator "`-`", pri čemu ukoliko je "`p`" neki objekat tipa "`Popravka`" a "`n`" nenegativan cijeli broj, "`p - n`" daje novi objekat tipa "`Popravka`" u kojem je datum popravke pomjeren za "`n`" dana unazad (u nedostatku bolje ideje, možete primijeniti operator "`--`" u petlji). Ukoliko "`n`" nije nenegativan broj, treba baciti izuzetak.
- Preklopljeni operator "`-=`" koji obezbjeđuje da izraz poput "`p -= n`" uvijek ima isto značenje kao i izraz "`p = p - n`".

Klasa "`Popravke`" predstavlja kolekciju podataka o zakazanim popravkama, izvedenu kao dinamički niz pokazivača na objekte tipa "`Popravke`". Ova klasa sadrži sljedeće elemente:

- Atribute koji predstavljaju broj pohranjenih intervencija, maksimalni broj intervencija koje se mogu pohraniti, kao i pokazivač za pristup dinamičkom nizu pokazivača na pohranjene objekte.
- Konstruktor sa jednim parametrom, koji vrši odgovarajuću dinamičku alokaciju memorije, pri čemu parametar predstavlja maksimalan broj intervencija koje se mogu pohraniti. Pri tome je potrebno zabraniti da se ovaj konstruktor koristi za automatsku konverziju objekata tipa "**`int`**" u objekte tipa "`Popravke`".
- Destruktor, koji oslobađa memoriju koju je objekat tipa "`Popravke`" zauzeo tokom svog života.
- Konstruktor kopije i preklopljeni operator dodjele, koji obezbjeđuju sigurno kopiranje i međusobno dodjeljivanje primjeraka klase "`Popravke`".
- Metodu koja vrši kreiranje i dodavanje podataka o novoj popravci, koja ima iste parametre kao i konstruktor klase "`Popravka`". Ova metoda kreira novi objekat koji sadrži podatke o zakazanoj intervenciji (u skladu sa navedenim parametrima) i smješta ga u kolekciju.
- Metodu bez parametara, koja sortira sve intervencije u rastući redoslijed.
- Metodu bez parametara, koja vraća referencu na najraniju zakazanu intervenciju (u slučaju da je kolekcija prazna, treba baciti izuzetak).
- Metodu bez parametara, koja uklanja iz kolekcije najraniju zakazanu intervenciju (u slučaju da je kolekcija prazna, treba baciti izuzetak).
- Metodu koja ispisuje sve intervencije koje su zakazane na zadani datum, pri čemu se dan i mjesec prosljeđuju kao parametri (ukoliko takvih intervencija nema, treba baciti izuzetak).
- Metode sa jednim parametrom koje snimaju sadržaj kolekcije u binarnu datoteku, odnosno obnavljaju sadržaj kolekcije iz binarne datoteke. Parametar ovih metoda predstavlja ime datoteke.

- Preklopljeni operator "<<" koji ispisuje sve pohranjene intervencije (u slučaju da je kolekcija prazna, treba ispisati odgovarajući komentar).
- Preklopljeni unarni operator "`!`", koji primijenjen na objekat tipa "`Popravke`" daje kao rezultat logičku vrijednost "**`true`**" ako i samo ako objekat predstavlja praznu kolekciju, a u suprotnom daje kao rezultat logičku vrijednost "**`false`**".
- Preklopljeni operator "`()`" koji omogućava pristup klijentima preko njihovog imena i prezimena. Ovaj operator vraća referencu na podatke o intervenciji za klijenta čije se ime i prezime zadaju unutar okruglih zagrada. Ukoliko takvog klijenta nema, treba baciti izuzetak.

Sve metode implementirajte izvan klase, osim metoda sa kratkom implementacijom, koje možete implementirati odmah unutar klase.

Napisane klase testirajte u testnom programu koji kreira nekoliko zahtjeva prema podacima unesenim sa tastature, a zatim ispisuje sve unijete zahtjeve, sortirane u rastući poredak.

Rješenje:

Prikazano rješenje prikazuje implementacije klasa "`Pregled`" i "`Pregledi`", bez glavnog programa. Pri tome se podrazumijeva da su sve neophodne biblioteke uključene u program.

```
class Popravka {
  string ime, prezime, vrsta_popravke;
  int dan, mjesec, sati, minute;
  static int BrojDana(int mjesec);
public:
  Popravka(string ime, string prezime, string vrsta_popravke, int dan,
    int mjesec, int sati, int minute);
  string DajIme() { return ime; }
  string DajPrezime() { return prezime; }
  string DajVrstuPopravke() { return vrsta_popravke; }
  int DajDan() { return dan; }
  int DajMjesec() { return mjesec; }
  int DajSate() { return sati; }
  int DajMinute() { return minute; }
  friend ostream &operator <<(ostream &cout, const Popravka &p);
  friend bool operator <(const Popravka &p1, const Popravka &p2);
  Popravka &operator --();
  Popravka operator --(int) { Popravka p(*this); --(*this); return p; }
  friend Popravka operator -(const Popravka &p, int n);
  Popravka &operator -=(int n) { return *this = *this - n; }
};
int Popravka::BrojDana(int mjesec) {
  if(mjesec == 2) return 28;
  if(mjesec == 4 || mjesec == 6 || mjesec == 9 || mjesec == 11) return 30;
  return 31;
}
Popravka::Popravka(string ime, string prezime, string vrsta_popravke,
  int dan, int mjesec, int sati, int minute) : ime(ime), prezime(prezime),
  vrsta_popravke(vrsta_popravke) {
    if(dan < 1 || dan > BrojDana(mjesec) || mjesec < 1 || mjesec > 12 ||
      sati < 0 || sati > 23 || minute < 0 || minute > 59)
        throw "Neispravni podaci!\n";
    Popravka::dan = dan; Popravka::mjesec = mjesec;
    Popravka::sati = sati; Popravka::minute = minute;
}
ostream &operator <<(ostream &cout, const Popravka &p) {
  return cout << "Klijent " << p.ime << " " << p.prezime
    << " zakazao je popravku " << p.vrsta_popravke << " dana " << p.dan
    << "." << p. mjesec << " u " << p.sati << ":" << p.minute;
}
```

*Redoslijed naredbi u izvedbi operatora "<" je jako bitan (uzeti u obzir da naredba "**return**" trenutno prekida izvođenje funkcije):*

```
bool operator <(const Popravka &p1, const Popravka &p2) {
  if(p1.mjesec < p2.mjesec) return true;
  if(p1.dan < p2.dan) return true;
  return p1.sati * 60 + p1.minute < p2.sati * 60 + p2.minute;
}

Popravka &Popravka::operator --() {
  dan--;
  if(dan == 0) {
    mjesec--;
    if(mjesec == 0) mjesec = 12;
    dan = BrojDana(mjesec);
  }
}
```

Prikazana izvedba operatora "–" predstavlja dobar kompromis između jednostavnosti i efikasnosti:

```
Popravka operator -(const Popravka &p, int n) {
  Popravka p1(p);
  p1.dan -= n;
  while(p1.dan <= 0) {
    p1.mjesec--;
    if(p1.mjesec == 0) p1.mjesec = 12;
    p1.dan += Popravka::BrojDana(p1.mjesec);
  }
  return p1;
}

class Popravke {
  int broj_popravki, max_broj_popravki;
  Popravka **kolekcija;
  static bool Kriterij(const Popravka *p1, const Popravka *p2) {
    return *p1 < *p2;
  }
public:
  explicit Popravke(int n) : broj_popravki(0), max_broj_popravki(n),
    kolekcija(new Popravka*[n]) {}
  ~Popravke();
  Popravke(const Popravke &p);
  Popravke &operator =(const Popravke &p);
  void DodajPopravku(string ime, string prezime, string vrsta_popravke,
    int dan, int mjesec, int sati, int minute);
  void Sortiraj() { sort(kolekcija, kolekcija + broj_popravki, Kriterij); }
  Popravka &DajNajranijuIntervenciju();
  void UkloniNajranijuIntervenciju();
  void IspisiIntervencijeNaDatum(int dan, int mjesec);
  void Snimi(const char ime[]);
  void Obnovi(const char ime[]);
  friend ostream &operator <<(ostream &cout, const Popravke &p);
  bool operator !() { return broj_popravki == 0; }
  Popravka &operator ()(string ime, string prezime);
};

Popravke::~Popravke() {
  for(int i = 0; i < broj_popravki; i++) delete kolekcija[i];
  delete[] kolekcija;
}

Popravke::Popravke(const Popravke &p) : broj_popravki(p.broj_popravki),
  max_broj_popravki(p.max_broj_popravki),
  kolekcija(new Popravka*[p.broj_popravki]) {
    for(int i = 0; i < broj_popravki; i++)
      kolekcija[i] = new Popravka(*p.kolekcija[i]);
}
```

Izvedba operatora dodjele kakva je ovdje implementirana definitivno nije najbolja moguća, ali je vjerovatno najjednostavnija (strategija "uništi stari sadržaj odredišta, a nakon toga kopiraj"):

```
Popravke &Popravke::operator =(const Popravke &p) {
  if(&p == this) return *this;
  for(int i = 0; i < broj_popravki; i++) delete kolekcija[i];
  delete[] kolekcija;
  broj_popravki = p.broj_popravki;
  max_broj_popravki = p.max_broj_popravki;
  kolekcija = new Popravka*[broj_popravki];
  for(int i = 0; i < broj_popravki; i++)
    kolekcija[i] = new Popravka(*p.kolekcija[i]);
  return *this;
}

void Popravke::DodajPopravku(string ime, string prezime,
  string vrsta_popravke, int dan, int mjesec, int sati, int minute) {
    if(broj_popravki == max_broj_popravki) throw "Popunjen kapacitet!\n";
    kolekcija[broj_popravki++] = new Popravka(ime, prezime, vrsta_popravke,
      dan, mjesec, sati, minute);
}
```

Metoda "DajNajranijuIntervenciju" *zasnovana je na manipulaciji sa pokazivačima, s obzirom da je to efikasnije nego manipulacija sa objektima. Na kraju se vrši dereferenciranje sa ciljem da se vrati referirani objekat:*

```
Popravka &Popravke::DajNajranijuIntervenciju() {
  if(broj_popravki == 0) throw "Nema zakazanih intervencija!\n";
  Popravka *najranija(kolekcija[0]);
  for(int i = 1; i < broj_popravki; i++)
    if(*kolekcija[i] < *najranija) najranija = kolekcija[i];
  return *najranija;
}
```

Prikazana izvedba metode "UkloniNajranijuintervenciju" *oslanja se na izvedbu prethodne metode i činjenicu da se objekti tipa* "Popravka" *mogu međusobno dodjeljivati, čime je izvedba ispala veoma kratka. U prikazanom rješenju, brisanje je realizirano tako što se na mjesto objekta koji se briše dovodi objekat koji je prije bio na kraju kolekcije. Naravno, moguća su i druga rješenja:*

```
void Popravke::UkloniNajranijuIntervenciju() {
  DajNajranijuIntervenciju() = *kolekcija[broj_popravki - 1];
  delete kolekcija[--broj_popravki];
}

void Popravke::IspisiIntervencijeNaDatum(int dan, int mjesec) {
  bool ima_pregleda(false);
  for(int i = 0; i < broj_popravki; i++)
    if(kolekcija[i]->DajDan() == dan && kolekcija[i]->DajMjesec()) {
      cout << *kolekcija[i] << endl;
      ima_pregleda = true;
    }
  if(!ima_pregleda) throw "Nema intervencija na traženi dan!\n";
}

ostream &operator <<(ostream &cout, const Popravke &p) {
  if(p.broj_popravki == 0) return cout << "Nema zakazanih intervencija!\n";
  for(int i = 0; i << p.broj_popravki; i++) cout << *p.kolekcija[i] << endl;
  return cout;
}

Popravka &Popravke::operator ()(string ime, string prezime) {
  for(int i = 0; i < broj_popravki; i++)
    if(kolekcija[i]->DajIme() == ime && kolekcija[i]->DajPrezime() == prezime)
      return *kolekcija[i];
  throw "Nije nadjen klijent!\n";
}
```

Slijede izvedbe metoda “Snimi” *i* “Obnovi” *onako kako je to zamislio predmetni nastavnik. Nažalost, prikazane metode NE RADE, bez obzira što je predmetni nastavnik ovakva rješenja priznavao kao korektna. Naime, predmetni nastavnik je previdio činjenicu da tip* “string” *sadržan u klasi* “Popravka” *nije POD (Plain Old Data) tip podataka. Stoga bi izvedbe metoda* “Snimi” *i* “Obnovi” *koje zaista rade bile vrlo komplicirane, i prosječan student ih ne bi bio u stanju napisati, tako da neće biti prikazane:*

```
void Popravke::Snimi(const char ime[]) {
  ofstream izlaz(ime, ios::out | ios::binary);
  izlaz.write((char*)kolekcija, sizeof(Popravke));
  for(int i = 0; i < broj_popravki; i++)
    izlaz.write((char*)kolekcija[i], sizeof(Popravka));
  if(!izlaz) throw "Nešto nije u redu sa upisom!\n";
}
void Popravke::Obnovi(const char ime[]) {
  ifstream ulaz(ime, ios::in | ios::binary);
  if(!ulaz) throw "Datoteka ne postoji!\n";
  for(int i = 0; i < broj_popravki; i++) delete kolekcija[i];
  delete[] kolekcija;
  ulaz.read((char*)kolekcija, sizeof(Popravke));
  for(int i = 0; i < broj_popravki; i++) {
    kolekcija[i] = new Popravka("", "", "", 1, 1, 1, 1);
    ulaz.read((char*)kolekcija[i], sizeof(Popravka));
  }
  if(!ulaz) throw "Obnova nije uspjela!\n";
}
```