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>
#include <complex>
using namespace std;
int f1(int &y) {
  y++; return y - 2;
}
int f2(int &z) {
  z--; return z + 2;
}
void f3(int &x, int y, int z(int &x)) {
  x += z(y);
  cout << complex<double>(x++, y) << endl;
}
int main() {
  int (*f[5])(int &x) = {f1, f2, f2, f1, f2};
  int y(5), z(2);
  for(int x = 0; x < 5; x++) f3(z, x, f[x]);
  cout << z;
  return 0;
}
```

Rješenje:

```
(1,1)
(4,0)
(8,1)
(11,4)
(17,3)
18
```

Zadatak 2. (2,5 poena)

Za neki broj kažemo da je *savršen* ukoliko je jednak sumi svih svojih djelilaca. Na primjer, 28 je savršen broj: njegovi djelioci su 1, 2, 4, 7 i 14, a 1 + 2 + 4 + 7 + 14 = 28. Napišite funkciju koja prima vektor cijelih brojeva kao parametar, a koja kao rezultat vraća novi vetor koji sadrži samo one brojeve iz vektora koji je zadan kao parametar koji su savršeni brojevi.

Rješenje:

Većina mogućih rješenja po logici rada bliska su ovdje ponuđenom rješenju (naravno, moguće su male varijacije na ponuđeno rješenje):

```
vector<int> PrepisiSavrsene(const vector<int> &v) {
  vector<int> v1;
  for(int i = 0; i < v.size(); i++) {
    int suma_djelilaca(0);
    for(int j = 1; j <= v[i] / 2; j++)
      if(v[i] % j == 0) suma_djelilaca += j;
    if(v[i] == suma_djelilaca) v1.push_back(v[i]);
  }
  return v1;
}
```

Zadatak 3. (3 poena)

Napišite funkciju "`OdstraniParneCifre`" koja ima jedan cjelobrojni parametar. Funkcija treba da transformira taj parametar, tako da se kao rezultat transformacije dobije novi broj iz kojeg su odstranjene sve cifre koji su parni brojevi, dok ostale cifre zadržavaju isti poredak kao u izvornom broju. Na primjer, ako se funkciji proslijedi promjenljiva koja sadrži vrijednost 32564718, ta ista promjenljiva po završetku funkcije imaće vrijednost 3571. Ukoliko su sve cifre bile parne, kao rezultat transformacije se dobija 0. Napišite i kratki isječak programa u kojem ćete demonstrirati kako se upotrebljava napisana funkcija.

Rješenje:

Moguće su razne varijante, uključujući i varijantu u kojoj se sve cifre izdvoje u vektor, a zatim od tako izdvojenih cifara formira broj sa traženim svojstvima. Međutim, ovdje ponuđena varijanta je vjerovatno najbolja moguća, jer ne koristi niti pomoćni vektor, niti funkciju za stepenovanje. U svakom slučaju, treba obratiti pažnju da je u pitanju funkcija koja ne vraća vrijednost,nego modificira svoj parametar:

```
void OdstraniParneCifre(int &x) {
  int x1(0), tezina(1);
  while(x != 0) {
    int cifra(x % 10);
    if(cifra % 2 != 0) {
      x1 += cifra * tezina; tezina *= 10;
    }
    x /= 10;
  }
  x = x1;
}
...
int broj;
cin >> broj;
OdstraniParneCifre(broj);
cout << broj;
```

Zadatak 4. (3 poena)

Nazovimo neku riječ "korektnom" ukoliko se u njoj ne pojavljuje skupina uzastopnih suglasnika duža od 2 slova. Na primjer, riječ "badlekuvje" je korektna, a riječ "balgftekrihvnu" nije korektna u smislu date definicije, jer sadrži 2 grupe suglasnika sa više od 2 uzastopna suglasnika ("lgft" i "hvn"). Napišite funkciju koja prihvata string kao parametar, a koja kao rezultat vraća logičku vrijednost "tačno" ukoliko string koji joj je proslijeđen kao parametar predstavlja korektnu riječ u smislu gore date definicije, a u suprotnom vraća logičku vrijednost "netačno".

Rješenje:

I ovdje su moguća razna rješenja. Ovdje ponuđeno rješenje broji uzastopne suglasnike, tako da ga je lako prilagoditi za proizvoljan broj uzastopnih suglasnika, što nije slučaj sa drugim rješenjima. Bitno je primijetiti potrebu za "resetiranjem" brojača uzastopnih suglasnika, bez kojeg rješenje neće raditi kako treba. Poziv funkcije "`tolower`" nije potreban ukoliko pretpostavimo da riječ sadrži isključivo mala slova:

```
bool DaLiJeRijecKorektna(const string &s) {
  int brojac_uzastopnih(0);
  for(int i = 0; i < s.length(); i++) {
    char c(tolower(s[i]));
    if(c != 'a' && c != 'e' && c != 'i' && c != 'o' && c != 'u')
      brojac_uzastopnih++;
    else brojac_uzastopnih = 0;
    if(brojac_uzastopnih > 2) return false;
  }
  return true;
}
```

Zadatak 5. (2,5 poena)

Napišite generičku funkciju "`Analiza`" sa 4 parametra. Prva dva parametra omeđuju blok elemenata za koje se pretpostavlja da se mogu međusobno porediti (tj. prvi parametar pokazuje na početak bloka, a drugi parametar neposredno iza njegovog kraja). Funkcija treba da pretraži zadani blok elemenata i da *smjesti* u treći i četvrti parametar respektivno broj elemenata u bloku koji su jednaki najmanjem, odnosno najvećem elementu bloka. Napišite i isječak programa u kojem ćete demonstrirati kako se napisana funkcija može iskoristiti na primjeru jednog fiksnog niza od 10 realnih brojeva.

Rješenje:

Kod ovog zadatka daćemo više različitih rješenja, zbog toga što se ovom problemu može pristupiti na različite načine. Prvo rješenje je rješenje u dva prolaza, kako bi većina studenata vjerovatno radila. Pored toga, prikazano rješenje koristi djelimičnu dedukciju, uz pretpostavku da su prva dva parametra isključivo pokazivači (a ne recimo iteratori). Prvo ćemo prikazati traženu funkciju:

```
template <typename Tip>
void Analiza(Tip *pocetak, Tip *kraj, int &br_min, int &br_max) {
  Tip min(*pocetak), max(*pocetak);
  for(Tip *p = pocetak; p != kraj; p++) {
    if(*p < min) min = *p;
    if(*p > max) max = *p;
  }
  br_min = br_max = 0;
  for(Tip *p = pocetak; p != kraj; p++) {
    if(*p == min) br_min++;
    if(*p == max) br_max++;
  }
}
```

Verzija funkcije koja koristi potpunu dedukciju neznatno je logički komplikovanija, ali je mnogo generalnija, jer nema ograničenja da prva dva parametra moraju biti pokazivači (upotrijebljena je riječ "`PokTip`" za ime metatipa da se naglasi da se očekuje da se ti parametri ponašaju poput pokazivača):

```
template <typename PokTip>
void Analiza(PokTip pocetak, PokTip kraj, int &br_min, int &br_max) {
  PokTip gdje_je_min(pocetak), gdje_je_max(pocetak);
  for(PokTip p = pocetak; p != kraj; p++) {
    if(*p < *gdje_je_min) gdje_je_min = p;
    if(*p > *gdje_je_max) gdje_je_max = p;
  }
  br_min = br_max = 0;
  for(PokTip p = pocetak; p != kraj; p++) {
    if(*p == *gdje_je_min) br_min++;
    if(*p == *gdje_je_max) br_max++;
  }
}
```

Međutim, posebno je interesantna činjenica da je rješenje moguće realizirati u samo jednom prolasku kroz petlju (uz "resetiranje" brojača), tako da je dobijeno rješenje dvostruko efikasnije od prethodnih. Prikazana je varijanta sa djelimičnom dedukcijom, a analogno bi se mogla izvesti i prepravka rješenja koje koristi potpunu dedukciju:

```
template <typename Tip>
void Analiza(Tip *pocetak, Tip *kraj, int &br_min, int &br_max) {
  Tip min(*pocetak), max(*pocetak);
  br_min = br_max = 0;
  for(Tip *p = pocetak; p != kraj; p++) {
    if(*p < min) { min = *p; br_min = 0; }
    if(*p > max) { max = *p; br_max = 0; }
    if(*p == min) br_min++;
    if(*p == max) br_max++;
  }
}
```

Bez obzira na verziju, traženi isječak programa mogao bi izgledati recimo ovako:

```
double niz[10] = {3, 5.12, 2.4, 7.8, 4, 3, 7.8, 5.723, 7.8, 6.21};
int br_min, br_max;
Analiza(niz, niz + 10, br_min, br_max);
cout << "Najmanji element niza se javlja " << br_min << "puta, a najveći "
  "element niza " << br_max << " puta.\n";
```

Zadatak 6. (2,5 poena)

Postoje razni metodi pomoću kojih je moguće izračunati približnu vrijednost integrala neke funkcije $f(x)$ na intervalu (a, b). Jedan od najjednostavnijih ali ujedno i najmanje tačnih metoda je tzv. *trapezno ili Eulerovo pravilo*, prema kojem je

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{n}\left[\frac{1}{2}f(a)+\frac{1}{2}f(b)+\sum_{k=1}^{n-1} f\left(a+\frac{b-a}{n}k\right)\right]$$

gdje je n broj podintervala na koji dijelimo interval (a, b). Veći broj podintervala daje i veću tačnost, ali do određene granice. Napišite funkciju "`TrapeznoPravilo`" koja prima kao parametre f, a, b i n (f je *funkcija čiji se integral računa*) a koja kao rezultat daje približnu vrijednost integrala računatu pomoću trapeznog pravila. Parametar n treba imati podrazumijevanu vrijednost 100, koja se koristi ukoliko se prilikom poziva funkcije ovaj parametar izostavi. Napišite i isječak programa u kojem ćete demonstrirati kako se napisana funkcija može iskoristiti da nađete približnu vrijednost integrala funkcije $1/x$ na intervalu $(1, 2)$.

Rješenje:

U ovom zadatku nisu moguća velika odstupanja od ponuđenog rješenja, osim u redoslijedu računanja pojedinih članova koji tvore formulu. Bitno je primijetiti da se funkcija koja računa vrijednost 1/x mora realizirati kao posebna funkcija sa imenom, jer se samo tako može proslijediti kao parametar:

```
double TrapeznoPravilo(double f(double), double a, double b, int n = 100) {
  double suma((f(a) + f(b) / 2);
  for(int k = 1; k < n; k++) suma += f(a + (b - a) * k / n);
  return (b - a) * suma / n;
}
...
double ReciprocnaVrijednost(double x) { return 1 / x; }
...
cout << TrapeznoPravilo(ReciprocnaVrijednost, 1, 2);
```

Zadatak 7. (2,5 poena)

Napišite funkciju sa jednim parametrom n koja kreira dinamički niz od n cijelih brojeva, popunjava ga sa prvih n Fibonačijevih brojeva i vraća kao rezultat pokazivač na prvi element kreiranog niza (Fibonačijevi brojevi F_n definirani su relacijom $F_n = F_{n-1} + F_{n-2}$, pri čemu je $F_1 = F_2 = 1$). Ukoliko parametar n nije prirodan broj, ili ukoliko nema dovoljno memorije za alokaciju, funkcija treba baciti izuzetak. Napišite i isječak programa u kojem ćete sa tastature unijeti broj n, kreirati dinamički niz prvih n Fibonačijevih brojeva putem napisane funcije, ispisati njegove elemente na ekran i, na kraju, izbrisati alocirani niz. U tom isječku obavezno predvidite i hvatanje izuzetaka koji mogu eventualno biti bačeni iz funkcije.

Rješenje:

U ovom zadatku treba paziti da se lako može previdjeti neka činjenica iz postavke problema, poput one da se drugi element niza smije postaviti samo ako niz ima barem dva člana:

```
int *KreirajNiz(int n) {
  if(n < 1) throw "Nelegalna vrijednost parametra!";
  try {
    int *niz(new int[n]);
    niz[0] = 1;
```

```
    if(n > 1) niz[1] = 1;
    for(int i = 2; i < n; i++) niz[i] = niz[i - 1] + niz[i - 2];
    return niz;
  }
  catch(...) {
    throw "Nema dovoljno memorije!";
  }
}
...
try {
  int n;
  cin >> n;
  int *niz(KreirajNiz(n));
  for(int i = 0; i < n; i++) cout << niz[i] << endl;
  delete[] niz;
}
catch(const char poruka[]) {
  cout << poruka << endl;
}
```

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>
#include <complex>
using namespace std;
int f1(int &y) {
  y--; return y + 2;
}
int f2(int &z) {
  z++; return z - 2;
}
void f3(int &x, int y, int z(int &x)) {
  x += z(y);
  cout << complex<double>(x++, y) << endl;
}
int main() {
  int (*f[5])(int &x) = {f1, f2, f2, f1, f2};
  int y(2), z(5);
  for(int x = 0; x < 5; x++) f3(z, x, f[x]);
  cout << z;
  return 0;
}
```

Rješenje:

```
(6,-1)
(7,2)
(9,3)
(14,2)
(18,5)
19
```

Zadatak 2. (2,5 poena)

Za neki broj kažemo da je *savršen* ukoliko je jednak sumi svih svojih djelilaca. Na primjer, 28 je savršen broj: njegovi djelioci su 1, 2, 4, 7 i 14, a 1 + 2 + 4 + 7 + 14 = 28. Napišite funkciju koja prima vektor cijelih brojeva kao parametar, a koja kao rezultat vraća novi vetor koji sadrži sve brojeve iz vektora koji je zadan kao parametar, osim onih koji su savršeni brojevi.

Rješenje:

Većina mogućih rješenja po logici rada bliska su ovdje ponuđenom rješenju (naravno, moguće su male varijacije na ponuđeno rješenje):

```
vector<int> PrepisiSveOsimSavrsenih(const vector<int> &v) {
  vector<int> v1;
  for(int i = 0; i < v.size(); i++) {
    int suma_djelilaca(0);
    for(int j = 1; j <= v[i] / 2; j++)
      if(v[i] % j == 0) suma_djelilaca += j;
    if(v[i] != suma_djelilaca) v1.push_back(v[i]);
  }
  return v1;
}
```

Zadatak 3. (3 poena)

Napišite funkciju "`OdstraniNeparneCifre`" koja ima jedan cjelobrojni parametar. Funkcija treba da transformira taj parametar, tako da se kao rezultat transformacije dobije novi broj iz kojeg su odstranjene sve cifre koji su neparni brojevi, dok ostale cifre zadržavaju isti poredak kao u izvornom broju. Na primjer, ako se funkciji proslijedi promjenljiva koja sadrži vrijednost 32564718, ta ista promjenljiva po završetku funkcije imaće vrijednost 2648. Ukoliko su sve cifre bile neparne, kao rezultat transformacije se dobija 0. Napišite i kratki isječak programa u kojem ćete demonstrirati kako se upotrebljava napisana funkcija.

Rješenje:

Moguće su razne varijante, uključujući i varijantu u kojoj se sve cifre izdvoje u vektor, a zatim od tako izdvojenih cifara formira broj sa traženim svojstvima. Međutim, ovdje ponuđena varijanta je vjerovatno najbolja moguća, jer ne koristi niti pomoćni vektor, niti funkciju za stepenovanje. U svakom slučaju, treba obratiti pažnju da je u pitanju funkcija koja ne vraća vrijednost,nego modificira svoj parametar:

```
void OdstraniNeparneCifre(int &x) {
  int x1(0), tezina(1);
  while(x != 0) {
    int cifra(x % 10);
    if(cifra % 2 == 0) {
      x1 += cifra * tezina; tezina *= 10;
    }
    x /= 10;
  }
  x = x1;
}
...
int broj;
cin >> broj;
OdstraniNeparneCifre(broj);
cout << broj;
```

Zadatak 4. (3 poena)

Nazovimo neku riječ "korektnom" ukoliko se u njoj ne pojavljuje skupina uzastopnih suglasnika duža od 2 slova. Na primjer, riječ "badlekuvje" je korektna, a riječ "balgftekrihvnu" nije korektna u smislu date definicije, jer sadrži 2 grupe suglasnika sa više od 2 uzastopna suglasnika ("lgft" i "hvn"). Napišite funkciju koja prihvata string kao parametar, a koja kao rezultat vraća logičku vrijednost "tačno" ukoliko string koji joj je proslijeđen kao parametar ne predstavlja korektnu riječ u smislu gore date definicije, a u suprotnom vraća logičku vrijednost "netačno".

Rješenje:

I ovdje su moguća razna rješenja. Ovdje ponuđeno rješenje broji uzastopne suglasnike, tako da ga je lako prilagoditi za proizvoljan broj uzastopnih suglasnika, što nije slučaj sa drugim rješenjima. Bitno je primijetiti potrebu za "resetiranjem" brojača uzastopnih suglasnika, bez kojeg rješenje neće raditi kako treba. Poziv funkcije "`tolower`" nije potreban ukoliko pretpostavimo da riječ sadrži isključivo mala slova:

```
bool DaLiJeRijecNekorektna(const string &s) {
  int brojac_uzastopnih(0);
  for(int i = 0; i < s.length(); i++) {
    char c(tolower(s[i]));
    if(c != 'a' && c != 'e' && c != 'i' && c != 'o' && c != 'u')
      brojac_uzastopnih++;
    else brojac_uzastopnih = 0;
    if(brojac_uzastopnih > 2) return true;
  }
  return false;
}
```

Zadatak 5. (2,5 poena)

Napišite generičku funkciju "`Analiza`" sa 4 parametra. Prva dva parametra omeđuju blok elemenata za koje se pretpostavlja da se mogu međusobno porediti (tj. prvi parametar pokazuje na početak bloka, a drugi parametar neposredno iza njegovog kraja). Funkcija treba da pretraži zadani blok elemenata i da *smjesti* u treći i četvrti parametar respektivno broj elemenata u bloku koji su jednaki najmanjem, odnosno najvećem elementu bloka. Napišite i isječak programa u kojem ćete demonstrirati kako se napisana funkcija može iskoristiti na primjeru jednog fiksnog niza od 10 cijelih brojeva.

Rješenje:

Kod ovog zadatka daćemo više različitih rješenja, zbog toga što se ovom problemu može pristupiti na različite načine. Prvo rješenje je rješenje u dva prolaza, kako bi većina studenata vjerovatno radila. Pored toga, prikazano rješenje koristi djelimičnu dedukciju, uz pretpostavku da su prva dva parametra isključivo pokazivači (a ne recimo iteratori). Prvo ćemo prikazati traženu funkciju:

```
template <typename Tip>
void Analiza(Tip *pocetak, Tip *kraj, int &br_min, int &br_max) {
  Tip min(*pocetak), max(*pocetak);
  for(Tip *p = pocetak; p != kraj; p++) {
    if(*p < min) min = *p;
    if(*p > max) max = *p;
  }
  br_min = br_max = 0;
  for(Tip *p = pocetak; p != kraj; p++) {
    if(*p == min) br_min++;
    if(*p == max) br_max++;
  }
}
```

Verzija funkcije koja koristi potpunu dedukciju neznatno je logički komplikovanija, ali je mnogo generalnija, jer nema ograničenja da prva dva parametra moraju biti pokazivači (upotrijebljena je riječ "`PokTip`" za ime metatipa da se naglasi da se očekuje da se ti parametri ponašaju poput pokazivača):

```
template <typename PokTip>
void Analiza(PokTip pocetak, PokTip kraj, int &br_min, int &br_max) {
  PokTip gdje_je_min(pocetak), gdje_je_max(pocetak);
  for(PokTip p = pocetak; p != kraj; p++) {
    if(*p < *gdje_je_min) gdje_je_min = p;
    if(*p > *gdje_je_max) gdje_je_max = p;
  }
  br_min = br_max = 0;
  for(PokTip p = pocetak; p != kraj; p++) {
    if(*p == *gdje_je_min) br_min++;
    if(*p == *gdje_je_max) br_max++;
  }
}
```

Međutim, posebno je interesantna činjenica da je rješenje moguće realizirati u samo jednom prolasku kroz petlju (uz "resetiranje" brojača), tako da je dobijeno rješenje dvostruko efikasnije od prethodnih. Prikazana je varijanta sa djelimičnom dedukcijom, a analogno bi se mogla izvesti i prepravka rješenja koje koristi potpunu dedukciju:

```
template <typename Tip>
void Analiza(Tip *pocetak, Tip *kraj, int &br_min, int &br_max) {
  Tip min(*pocetak), max(*pocetak);
  br_min = br_max = 0;
  for(Tip *p = pocetak; p != kraj; p++) {
    if(*p < min) { min = *p; br_min = 0; }
    if(*p > max) { max = *p; br_max = 0; }
    if(*p == min) br_min++;
    if(*p == max) br_max++;
  }
}
```

Bez obzira na verziju, traženi isječak programa mogao bi izgledati recimo ovako:

```
int niz[10] = {3, 5, 2, 7, 4, 3, 7, 5, 7, 6};
int br_min, br_max;
Analiza(niz, niz + 10, br_min, br_max);
cout << "Najmanji element niza se javlja " << br_min << "puta, a najveći "
  "element niza " << br_max << " puta.\n";
```

Zadatak 6. (2,5 poena)

Postoje razni metodi pomoću kojih je moguće izračunati približnu vrijednost integrala neke funkcije $f(x)$ na intervalu (a, b). Jedan od najjednostavnijih ali ujedno i najmanje tačnih metoda je tzv. *trapezno ili Eulerovo pravilo*, prema kojem je

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{n}\left[\frac{1}{2}f(a)+\frac{1}{2}f(b)+\sum_{k=1}^{n-1} f\left(a+\frac{b-a}{n}k\right)\right]$$

gdje je n broj podintervala na koji dijelimo interval (a, b). Veći broj podintervala daje i veću tačnost, ali do određene granice. Napišite funkciju "`TrapeznoPravilo`" koja prima kao parametre f, a, b i n (f je *funkcija čiji se integral računa*) a koja kao rezultat daje približnu vrijednost integrala računatu pomoću trapeznog pravila. Parametar n treba imati podrazumijevanu vrijednost 150, koja se koristi ukoliko se prilikom poziva funkcije ovaj parametar izostavi. Napišite i isječak programa u kojem ćete demonstrirati kako se napisana funkcija može iskoristiti da nađete približnu vrijednost integrala funkcije x^3 na intervalu $(0, 5)$.

Rješenje:

U ovom zadatku nisu moguća velika odstupanja od ponuđenog rješenja, osim u redoslijedu računanja pojedinih članova koji tvore formulu. Bitno je primijetiti da se funkcija koja računa vrijednost x^3 mora realizirati kao posebna funkcija sa imenom, jer se samo tako može proslijediti kao parametar:

```
double TrapeznoPravilo(double f(double), double a, double b, int n = 100) {
  double suma((f(a) + f(b) / 2);
  for(int k = 1; k < n; k++) suma += f(a + (b - a) * k / n);
  return (b - a) * suma / n;
}
...
double Kub(double x) { return x * x * x; }
...
cout << TrapeznoPravilo(Kub, 0, 5);
```

Zadatak 7. (2,5 poena)

Napišite funkciju sa jednim parametrom n koja kreira dinamički niz od n cijelih brojeva, popunjava ga sa prvih n Fibonačijevih brojeva i vraća kao rezultat pokazivač na prvi element kreiranog niza (Fibonačijevi brojevi F_n definirani su relacijom $F_n = F_{n-1} + F_{n-2}$, pri čemu je $F_1 = F_2 = 1$). Ukoliko parametar n nije prirodan broj, ili ukoliko nema dovoljno memorije za alokaciju, funkcija treba baciti izuzetak. Napišite i isječak programa u kojem ćete sa tastature unijeti broj n, kreirati dinamički niz prvih n Fibonačijevih brojeva putem napisane funcije, ispisati njegove elemente na ekran i, na kraju, izbrisati alocirani niz. U tom isječku obavezno predvidite i hvatanje izuzetaka koji mogu eventualno biti bačeni iz funkcije.

Rješenje:

U ovom zadatku treba paziti da se lako može previdjeti neka činjenica iz postavke problema, poput one da se drugi element niza smije postaviti samo ako niz ima barem dva člana:

```
int *KreirajNiz(int n) {
  if(n < 1) throw "Nelegalna vrijednost parametra!";
  try {
    int *niz(new int[n]);
```

```
    niz[0] = 1;
    if(n > 1) niz[1] = 1;
    for(int i = 2; i < n; i++) niz[i] = niz[i - 1] + niz[i - 2];
    return niz;
  }
  catch(...) {
    throw "Nema dovoljno memorije!";
  }
}
...
try {
  int n;
  cin >> n;
  int *niz(KreirajNiz(n));
  for(int i = 0; i < n; i++) cout << niz[i] << endl;
  delete[] niz;
}
catch(const char poruka[]) {
  cout << poruka << endl;
}
```

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

NAPOMENA: U svim klasama koje treba razviti, trivijalne metode koje se mogu implementirati u jednoj ili najviše dvije naredbe možete implementirati direktno unutar deklaracije klase, dok sve ostale metode trebate implementirati izvan deklaracije klase.

Za potrebe vođenja evidencije o knjigama u nekoj biblioteci, u evidencijskom programu koriste se klase nazvane "`Knjiga`", "`Udzbenik`" i "`Biblioteka`". Vaš zadatak je da definirate i implementirate te klase i da napišete glavni program koji koristi te klase. Klasa "`Knjiga`" opisuje podatke o jednoj knjizi, a sadrži sljedeće elemente:

- Privatni evidencijski broj knjige (cijeli broj).
- Privatne podatke o naslovu knjige, imenu pisca i žanr (svi ovi podaci su tipa "`string`").
- Privatni podatak o godini izdanja knjige (cijeli broj).
- Privatni podatak o članskom broju čitaoca kod koga se trenutno nalazi knjiga, ili 0 ukoliko knjiga trenutno nije na čitanju (tj. ukoliko je trenutno raspoloživa).
- Privatni podatak koliko dugo dana je knjiga na čitanju (ukoliko knjiga trenutno nije na čitanju, vrijednost ovog podatka je nebitna).
- Konstruktor, koji inicijalizira podatke o evidencijskom broju, naslovu knjige, imenu pisca, žanru i godini izdanja na vrijednosti zadane parametrima. Pri tome se podaci o čitaocu inicijaliziraju tako da signaliziraju da je knjiga slobodna.
- Trivijalne pristupne metode, kojim se može saznati vrijednost svih privatnih podataka.
- Metodu koja knjigu proglašava "zaduženom". Parametar metode je članski broj čitaoca koji zadužuje knjigu. Pri zaduženju, informacija o tome koliko dugo je knjiga na čitanju postavlja se na nulu.
- Metodu bez parametara koja "razdužuje" knjigu.
- Preklopljeni unarni operator "`!`" koji vraća logičku vrijednost "**`true`**" ukoliko je knjiga zadužena, a "**`false`**" ukoliko nije.
- Preklopljeni unarni operator "++" koji povećava evidenciju o tome koliko je dugo knjiga na čitanju za jedinicu, ukoliko je knjiga zadužena. Ukoliko knjiga nije zadužena, ovaj operator ne radi ništa. Potrebno je podržati i prefiksnu i postfiksnu verziju ovog operatora.
- Preklopljene relacione operatore "==" i "`!=`" koji testiraju da li su dvije knjige iste ili nisu. Smatra se da su dvije knjige iste ukoliko im se slažu naslov, ime pisca, žanr i godina izdanja.
- Privatnu virtuelnu metodu koja ispisuje na izlazni tok (recimo, ekran) podatke o evidencijskom broju, naslovu, imenu pisca, žanru i godini izdanja knjige. Parametar metode je referenca na objekat toka preko kojeg se vrši ispis (tipa "`ostream`"). Stil ispisa odredite po želji.
- Preklopljeni operator ispisa "<<" koji ispisuje na izlazni tok iste podatke kao i prethodna metoda (u principu, ovaj operator samo treba pozvati prethodnu metodu).

Klasa "`Udzbenik`" je naslijeđena iz klase "`Knjiga`". Novi elementi ove klase u odnosu na baznu klasu "`Knjiga`" su sljedeći:

- Privatni podatak o tome za koji je predmet udžbenik namijenjen (tipa "`string`").
- Pristupna metoda pomoću koje se može saznati za koji je predmet udžbenik namijenjen.
- Konstruktor ove klase je proširen da omogući i zadavanje predmeta za koji je udžbenik namijenjen.
- Privatna metoda za ispis je izmijenjena da predvidi i ispis predmeta za koji je udžbenik namijenjen.

Svi ostali elementi klase "`Udzbenik`" se prosto preuzimaju iz klase "`Knjiga`". Klasa "`Biblioteka`" opisuje kolekciju knjiga (uključujući i udžbenike), a sadrži sljedeće elemente:

- Privatne informacije o broju knjiga u biblioteci, kao i o maksimalnom broju knjiga koje biblioteka može evidentirati. Pri tome, podatak o maksimalnom broju knjiga koje biblioteka može evidentirati treba biti izveden kao konstantni atribut.
- Podatke o evidentiranim knjigama, izvedene kao dinamički niz pokazivača na dinamički alocirane objekte tipa "`Knjiga`", kojima se pristupa preko privatnog dvojnog pokazivača.

- Konstruktor koji vrši dinamičku alokaciju memorije, pri čemu parametar predstavlja maksimalan broj knjiga koje biblioteka može evidentirati, kao i odgovarajući destruktor. Pri tome, treba zabraniti da se ovaj konstruktor koristi za automatsku konverziju cijelih brojeva u objekte tipa "`Biblioteka`". Pored toga, kopiranje i međusobno dodjeljivanje objekata tipa "`Biblioteka`" treba zabraniti.
- Metodu koja kreira novu knjigu (običnu) i evidentira je u biblioteci. Parametri ove metode su isti kao u konstruktoru klase "`Knjiga`".
- Metodu koja kreira novi udžbenik i evidentira ga u biblioteci. Parametri ove metode su isti kao u konstruktoru klase "`Udzbenik`".
- Metodu koja "zadužuje" knjigu, pri čemu se kao parametri zadaju evidencijski broj knjige i članski broj čitaoca koji zadužuje knjigu. Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Metodu koja "razdužuje" knjigu, pri čemu se kao parametri zadaje evidencijski broj knjige koja se razdužuje. Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Metodu koja vraća logičku vrijednost "tačno" ili "netačno", ovisno da li je knjiga čiji je evidencijski broj zadan kao parametar trenutno zadužena ili ne. Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Metodu koja vraća članski broj čitaoca kod kojeg se nalazi knjiga čiji je evidencijski broj zadan kao parametar (odnosno nulu ukoliko knjiga nije zadužena). Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Medode koje ispisuje podatke o svim slobodnim knjigama, svim zaduženim knjigama, i knjigama koje su na čitanju duže od *n* dana, pri čemu se *n* zadaje kao parametar.
- Metodu koja sortira sve knjige u takav poredak da se knjige koje se su duže vremena zadužene nađu ispred onih koje su kraće vremena zadužene. Ukoliko dvije knjige imaju isto trajanje zaduženja, knjiga čiji naziv dolazi prije po abecedi treba da bude ispred knjige koja dolazi kasnije po abecedi.
- Preklopljeni operator "`[]`" koji vraća referencu na knjigu čiji se evidencijski broj navodi unutar uglastih zagrada. Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Preklopljeni unarni operator "++" koji povećava evidenciju o tome koliko je dugo knjiga na čitanju za jedinicu za sve zadužene knjige. Potrebno je podržati samo prefiksnu verziju ovog operatora.

Napišite i glavni program, koji će iz tekstualnih datoteka pročitati podatke o knjigama i njihovim zaduženjima i nakon toga ispisati podatke o svim zaduženim i svim slobodnim knjigama. Podaci o knjigama nalaze se u tekstualnoj datoteci "`KNJIGE.TXT`". Ova datoteka je organizirana tako da se za svaku knjigu podaci o evidencijskom broju, naslovu knjige, imenu pisca, žanru i godini izdanja u datoteci nalaze upravo tim redom, svaki podatak u posebnom redu. Ukoliko je u pitanju udžbenik, ispred evidencijskog broja nalazi se slovo "`U`", a na kraju se nalazi i podatak o predmetu za koji je udžbenik namijenjen. Na primjer, datoteka "`KNJIGE.TXT`" može izgledati ovako:

```
1234
Derviš i smrt
Meša Selimović
Roman
1976
U4312
Zbirka zadataka iz više matematike
Momčilo Uščumlić
Zbirka zadataka
1983
Inžinjerska matematika I/II
...
```

Podaci o zaduženjima nalaze se u datoteci "`ZADUZENJA.TXT`". Svaki red ove datoteke predstavlja po jednu zaduženu knjigu, a u njemu se nalaze evidencijski broj knjige i članski broj čitaoca koji je zadužio knjigu, razdvojeni zarezom (npr. "`4312,344`"). Radi jednostavnosti, možete pretpostaviti da obje datoteke sigurno sadrže ispravne podatke.

Rješenje:

Prvo slijedi klasa "Knjiga". Kako su sve metode ove klase kratke, implementirane su uglavnom unutar deklaracije klase, osim operatorske funkcije za prefiksni operator "++" (izvedene kao funkcija članica), operatorske funkcije za operator "==" (izvedene kao prijateljska funkcija) i virtualne funkcije članice "IspisiNaTok" koje su, čisto radi demonstracije, implementirane izvan klase:

```
class Knjiga {
  int ev_broj, god_izdanja, kod_koga_je, koliko_dugo;
  string naslov, pisac, zanr;
public:
  Knjiga(int ev_broj, string naslov, string pisac, string zanr,
    int god_izdanja) : ev_broj(ev_broj), naslov(naslov), pisac(pisac),
    zanr(zanr), god_izdanja(god_izdanja), kod_koga_je(0) {}
  int DajEvidencijskiBroj() const { return ev_broj; }
  string DajNaslov() const { return naslov; }
  string DajImePisca() const { return pisac; }
  string DajZanr() const { return zanr; }
  int DajGodinuIzdanja() const { return god_izdanja; }
  int DajKodKogaJe() const { return kod_koga_je; }
  int DajKolikoJeDugoNaCitanju() const { return koliko_dugo; }
  void ZaduziKnjigu(int clanski_broj) {
    kod_koga_je = clanski_broj; koliko_dugo = 0;
  }
  void RazduziKnjigu() { kod_koga_je = 0; }
  bool operator !() { return kod_koga_je != 0; }
  Knjiga &operator ++();
  Knjiga operator ++(int) { Knjiga k(*this); ++(*this); return k; }
  friend bool operator ==(const Knjiga &k1, const Knjiga &k2);
  friend bool operator !=(const Knjiga &k1, const Knjiga &k2) {
    return !(k1 == k2);
  }
  virtual void IspisiNaTok(ostream &cout) const;
  friend ostream &operator <<(ostream &cout, const Knjiga &k) {
    k.IspisiNaTok(cout); return cout;
  }
};
Knjiga &Knjiga::operator ++() {
  if(kod_koga_je != 0) koliko_dugo++;
  return *this;
}
bool operator ==(const Knjiga &k1, const Knjiga &k2) {
  return k1.naslov == k2.naslov && k1.pisac == k2.pisac
    && k1.zanr == k2.zanr && k1.god_izdanja == k2.god_izdanja;
}
void Knjiga::IspisiNaTok(ostream &cout) const {
  cout << "Evidencijski broj: " << ev_broj << endl << "Naslov: " << naslov
    << endl << "Ime pisca: " << pisac << endl << "Žanr: " << zanr << endl
    << "Godina izdanja: " << god_izdanja << endl;
}
```

U klasi "Udzbenik" nema se ništa posebno raditi, s obzirom da se radi o neznatno nadograđenoj specijalizaciji klase "Knjiga":

```
class Udzbenik : public Knjiga {
  string predmet;
public:
  string DajPredmet() const { return predmet; }
  Udzbenik(int ev_broj, string naslov, string pisac, string zanr,
    int god_izdanja, string predmet) : Knjiga(ev_broj, naslov, pisac, zanr,
    god_izdanja), predmet(predmet) {}
  void IspisiNaTok(ostream &cout) const;
};
```

```
void Udzbenik::IspisiNaTok(ostream &cout) const {
  Knjiga::IspisiNaTok(cout);
  cout << "Predmet: " << predmet << endl;
}
```

U prikazanoj izvedbi klase "Biblioteka", implementacije metoda poput "ZaduziKnjigu" itd. su izuzetno kratke, s obzirom da se oslanjaju na preklopljeni operator "[]" i odgovarajuće metode klase "Knjiga". Kasnije će biti demonstrirano kako se ove metode mogu izvesti neovisno od preklopljenog operatora "[]" po cijenu da postanu složenije:

```
class Biblioteka {
  int broj_knjiga;
  const int max_br_knjiga;
  Knjiga **knjige;
  Biblioteka(const Knjiga &b);                    // Neimplementirano...
  Biblioteka &operator =(const Biblioteka &b);    // Neimplementirano...
  static bool Kriterij(const Knjiga *k1, const Knjiga *k2);
public:
  explicit Biblioteka(int kapacitet) : broj_knjiga(0),
    max_br_knjiga(kapacitet), knjige(new Knjiga*[kapacitet]) {}
  void KreirajKnjigu(int ev_broj, string naslov, string pisac, string zanr,
    int god_izdanja);
  void KreirajUdzbenik(int ev_broj, string naslov, string pisac, string zanr,
    int god_izdanja, string predmet);
  void ZaduziKnjigu(int ev_broj, int clanski_broj) {
    (*this)[ev_broj].ZaduziKnjigu(clanski_broj);
  }
  void RazduziKnjigu(int ev_broj) { (*this)[ev_broj].RazduziKnjigu(); }
  bool DaLiJeZaduzena(int ev_broj) const { return !(*this)[ev_broj]; }
  int KodKogaJeKnjiga(int ev_broj) const {
    return (*this)[ev_broj].DajKodKogaJe();
  }
  void IspisiSlobodne() const;
  void IspisiZauzete() const;
  void IspisiNaCitanjuDuzeOd(int br_dana) const;
  void SortirajKnjige() { sort(knjige, knjige + broj_knjiga, Kriterij); }
  Knjiga &operator [](int ev_broj);
  Knjiga operator [](int ev_broj) const;
  Biblioteka &operator ++();
};
```

*Konstrukcija "(*this)[ev_broj]" u metodama poput "ZaduziKnjigu" itd. može se zamijeniti ekvivalentnom konstrukcijom "operator [](ev_broj)" u kojoj se direktno poziva operatorska funkcija za operator "[]", recimo "operator [](ev_broj).ZaduziKnjigu(clanski_broj)". Alternativno, ove metode su se mogle napisati bez pozivanja na preklopljeni operator "[]", na primjer na sljedeći način koji pokazuje alternativnu izvedbu metode "ZaduziKnjigu" (srodne metode poput "RazduziKnjigu", "DaLiJeZaduzena" i "KodKogaJeKnjiga" mogu se izvesti analogno):*

```
void Biblioteka::ZaduziKnjigu(int ev_broj, int clanski_broj) {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajEvidencijskiBroj() == ev_broj) {
      knjige[i]->ZaduziKnjigu(clanski_broj);
      return;
    }
  throw "Nema knjige sa traženim evidencijskim brojem!";
}
```

Slijede implementacije ostalih metoda koje nisu implementirane unutar klase:

```
void Biblioteka::KreirajKnjigu(int ev_broj, string naslov, string pisac,
  string zanr, int god_izdanja) {
    if(broj_knjiga == max_br_knjiga) throw "Popunjen kapacitet!";
    knjige[broj_knjiga++] = new Knjiga(ev_broj, naslov, pisac, zanr,
      god_izdanja);
}
```

```
void Biblioteka::KreirajUdzbenik(int ev_broj, string naslov, string pisac,
  string zanr, int god_izdanja, string predmet) {
    if(broj_knjiga == max_br_knjiga) throw "Popunjen kapacitet!";
    knjige[broj_knjiga++] = new Udzbenik(ev_broj, naslov, pisac, zanr,
      god_izdanja, predmet);
}
void Biblioteka::IspisiSlobodne() const {
  for(int i = 0; i < broj_knjiga; i++)
    if(!!*knjige[i]) cout << *knjige[i];
}
```

U prethodnoj metodi upotrijebljena je neobična konstrukcija "`!!*knjige[i]`". Međutim, ona je posve logična, s obzirom da je "`*knjige[i]`" objekat tipa "`Knjiga`" za koji je definiran operator "`!`", tako da je "`!*knjige[i]`" logička vrijednost na koju se ponovo može primijeniti operator "`!`", ovaj put u značenju logičke negacije. Bez ove konstrukcije, prethodna metoda se mogla izvesti recimo ovako:

```
void Biblioteka::IspisiSlobodne() const {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajKodKogaJe() == 0) cout << *knjige[i];
}
```

Slijedi implementacija preostalih elemenata klase:

```
void Biblioteka::IspisiZauzete() const {
  for(int i = 0; i < broj_knjiga; i++)
    if(!*knjige[i]) cout << *knjige[i];
}
void Biblioteka::IspisiNaCitanjuDuzeOd(int br_dana) const {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajKolikoJeDugoNaCitanju() > br_dana) cout << *knjige[i];
}
static bool Kriterij(const Knjiga *k1, const Knjiga *k2) {
  if(k1->DajKolikoJeDugoNaCitanju() != k2->DajKolikoJeDugoNaCitanju())
    return k1->DajKolikoJeDugoNaCitanju() > k2->DajKolikoJeDugoNaCitanju();
  return k1->DajNaslov() < k2->DajNaslov();
}
Knjiga &Biblioteka::operator [](int ev_broj) {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajEvidencijskiBroj() == ev_broj) return *knjige[i];
  throw "Nema knjige sa traženim evidencijskim brojem!";
}
Knjiga Biblioteka::operator [](int ev_broj) const {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajEvidencijskiBroj() == ev_broj) return *knjige[i];
  throw "Nema knjige sa traženim evidencijskim brojem!";
}
Biblioteka &Biblioteka::operator ++() {
  for(int i = 0; i < broj_knjiga; i++)
    if(!*knjige[i]) ++(*knjige[i]);
  return *this;
}
```

Preostaje još glavni program. Postoji mnogo načina da se izvede ono što je traženo od glavnog programa, a ovdje prikazano rješenje je vjerovatno najkraće i logički najelegantnije. S obzirom da nam nije poznato koliko datoteka "`KNJIGE.TXT`" sadrži knjiga, kapacitet biblioteke je postavljen na 1000 knjiga. Bolje rješenje bilo bi da se prvo izvrši jedan prolaz kroz datoteku da se utvrdi koliko ona sadrži knjiga pa da se tek onda kreira biblioteka i u drugom prolazu kroz istu datoteku izvrši stvarno čitanje knjiga. Mada su neki studenti uočili ovu činjenicu, njeno rješenje se ne traži da bi se zadatak prihvatio kao korektan, tako da će u prikazanom rješenju ova činjenica biti zanemarena. Primijetimo da je upotreba funkcije "`getline`" neophodna da bi se pročitala čitava rečenica. Također, treba obratiti pažnju na upotrebu manipulatora "`ws`":

```
int main() {
  Biblioteka biblioteka(1000);
  ifstream knjige("KNJIGE.TXT");
  for(;;) {
    bool da_li_je_udzbenik(false);
    if(knjige.peek() != 'U') {
      da_li_je_udzbenik = true;
      knjige.get();
    }
    int ev_broj, god_izdanja;
    string naslov, pisac, zanr, predmet;
    knjige >> ev_broj >> ws;
    if(!knjige) break;
    getline(knjige, naslov); getline(knjige, pisac); getline(knjige, zanr);
    knjige >> god_izdanja >> ws;
    if(da_li_je_udzbenik) {
      getline(knjige, predmet);
      biblioteka.KreirajUdzbenik(ev_broj, naslov, pisac, zanr, god_izdanja,
        predmet);
    }
    else biblioteka.KreirajKnjigu(ev_broj, naslov, pisac, zanr, god_izdanja);
  }
  ifstream zaduzenja("ZADUZENJA.TXT");
  int ev_broj, clanski_broj;
  char znak;
  while(zaduzenja >> ev_broj >> znak >> clanski_broj)
    biblioteka.ZaduziKnjigu(ev_broj, clanski_broj);
  cout << "Spisak zaduženih knjiga:\n";
  biblioteka.IspisiZauzete();
  cout << "Spisak slobodnih knjiga:\n";
  biblioteka.IspisiSlobodne();
  return 0;
}
```

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

NAPOMENA: U svim klasama koje treba razviti, trivijalne metode koje se mogu implementirati u jednoj ili najviše dvije naredbe možete implementirati direktno unutar deklaracije klase, dok sve ostale metode trebate implementirati izvan deklaracije klase.

Za potrebe vođenja evidencije o knjigama u nekoj biblioteci, u evidencijskom programu koriste se klase nazvane "`Knjiga`", "`Udzbenik`" i "`Biblioteka`". Vaš zadatak je da definirate i implementirate te klase i da napišete glavni program koji koristi te klase. Klasa "`Knjiga`" opisuje podatke o jednoj knjizi, a sadrži sljedeće elemente:

- Privatni evidencijski broj knjige (cijeli broj).
- Privatne podatke o naslovu knjige, imenu pisca i žanr (svi ovi podaci su tipa "`string`").
- Privatni podatak o godini izdanja knjige (cijeli broj).
- Privatni podatak o članskom broju čitaoca kod koga se trenutno nalazi knjiga, ili 0 ukoliko knjiga trenutno nije na čitanju (tj. ukoliko je trenutno raspoloživa).
- Privatni podatak koliko dugo dana je knjiga na čitanju (ukoliko knjiga trenutno nije na čitanju, vrijednost ovog podatka je nebitna).
- Konstruktor, koji inicijalizira podatke o evidencijskom broju, naslovu knjige, imenu pisca, žanru i godini izdanja na vrijednosti zadane parametrima. Pri tome se podaci o čitaocu inicijaliziraju tako da signaliziraju da je knjiga slobodna.
- Trivijalne pristupne metode, kojim se može saznati vrijednost svih privatnih podataka.
- Metodu koja knjigu proglašava "zaduženom". Parametar metode je članski broj čitaoca koji zadužuje knjigu. Pri zaduženju, informacija o tome koliko dugo je knjiga na čitanju postavlja se na nulu.
- Metodu bez parametara koja "razdužuje" knjigu.
- Preklopljeni unarni operator "`!`" koji vraća logičku vrijednost "**`true`**" ukoliko je knjiga zadužena, a "**`false`**" ukoliko nije.
- Preklopljeni unarni operator "++" koji povećava evidenciju o tome koliko je dugo knjiga na čitanju za jedinicu, ukoliko je knjiga zadužena. Ukoliko knjiga nije zadužena, ovaj operator ne radi ništa. Potrebno je podržati i prefiksnu i postfiksnu verziju ovog operatora.
- Preklopljene relacione operatore "==" i "`!=`" koji testiraju da li su dvije knjige iste ili nisu. Smatra se da su dvije knjige iste ukoliko im se slažu naslov, ime pisca, žanr i godina izdanja.
- Privatnu virtuelnu metodu koja ispisuje na izlazni tok (recimo, ekran) podatke o evidencijskom broju, naslovu, imenu pisca, žanru i godini izdanja knjige. Parametar metode je referenca na objekat toka preko kojeg se vrši ispis (tipa "`ostream`"). Stil ispisa odredite po želji.
- Preklopljeni operator ispisa "<<" koji ispisuje na izlazni tok iste podatke kao i prethodna metoda (u principu, ovaj operator samo treba pozvati prethodnu metodu).

Klasa "`Udzbenik`" je naslijeđena iz klase "`Knjiga`". Novi elementi ove klase u odnosu na baznu klasu "`Knjiga`" su sljedeći:

- Privatni podatak o tome za koji je predmet udžbenik namijenjen (tipa "`string`").
- Pristupna metoda pomoću koje se može saznati za koji je predmet udžbenik namijenjen.
- Konstruktor ove klase je proširen da omogući i zadavanje predmeta za koji je udžbenik namijenjen.
- Privatna metoda za ispis je izmijenjena da predvidi i ispis predmeta za koji je udžbenik namijenjen.

Svi ostali elementi klase "`Udzbenik`" se prosto preuzimaju iz klase "`Knjiga`". Klasa "`Biblioteka`" opisuje kolekciju knjiga (uključujući i udžbenike), a sadrži sljedeće elemente:

- Privatne informacije o broju knjiga u biblioteci, kao i o maksimalnom broju knjiga koje biblioteka može evidentirati. Pri tome, podatak o maksimalnom broju knjiga koje biblioteka može evidentirati treba biti izveden kao konstantni atribut.
- Podatke o evidentiranim knjigama, izvedene kao dinamički niz pokazivača na dinamički alocirane objekte tipa "`Knjiga`", kojima se pristupa preko privatnog dvojnog pokazivača.

- Konstruktor koji vrši dinamičku alokaciju memorije, pri čemu parametar predstavlja maksimalan broj knjiga koje biblioteka može evidentirati, kao i odgovarajući destruktor. Pri tome, treba zabraniti da se ovaj konstruktor koristi za automatsku konverziju cijelih brojeva u objekte tipa "`Biblioteka`". Pored toga, kopiranje i međusobno dodjeljivanje objekata tipa "`Biblioteka`" treba zabraniti.
- Metodu koja kreira novu knjigu (običnu) i evidentira je u biblioteci. Parametri ove metode su isti kao u konstruktoru klase "`Knjiga`".
- Metodu koja kreira novi udžbenik i evidentira ga u biblioteci. Parametri ove metode su isti kao u konstruktoru klase "`Udzbenik`".
- Metodu koja "zadužuje" knjigu, pri čemu se kao parametri zadaju evidencijski broj knjige i članski broj čitaoca koji zadužuje knjigu. Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Metodu koja "razdužuje" knjigu, pri čemu se kao parametri zadaje evidencijski broj knjige koja se razdužuje. Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Metodu koja vraća logičku vrijednost "tačno" ili "netačno", ovisno da li je knjiga čiji je evidencijski broj zadan kao parametar trenutno zadužena ili ne. Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Metodu koja vraća članski broj čitaoca kod kojeg se nalazi knjiga čiji je evidencijski broj zadan kao parametar (odnosno nulu ukoliko knjiga nije zadužena). Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Medode koje ispisuje podatke o svim slobodnim knjigama, svim zaduženim knjigama, i knjigama koje su na čitanju duže od *n* dana, pri čemu se *n* zadaje kao parametar.
- Metodu koja sortira sve knjige u takav poredak da se knjige koje se su duže vremena zadužene nađu ispred onih koje su kraće vremena zadužene. Ukoliko dvije knjige imaju isto trajanje zaduženja, knjiga čiji naziv dolazi prije po abecedi treba da bude ispred knjige koja dolazi kasnije po abecedi.
- Preklopljeni operator "`[]`" koji vraća referencu na knjigu čiji se evidencijski broj navodi unutar uglastih zagrada. Ukoliko knjiga sa zadanim evidencijskim brojem ne postoji, treba baciti izuzetak.
- Preklopljeni unarni operator "++" koji povećava evidenciju o tome koliko je dugo knjiga na čitanju za jedinicu za sve zadužene knjige. Potrebno je podržati samo prefiksnu verziju ovog operatora.

Napišite i glavni program, koji će iz tekstualnih datoteka pročitati podatke o knjigama i njihovim zaduženjima i nakon toga ispisati podatke o svim zaduženim i svim slobodnim knjigama. Podaci o knjigama nalaze se u tekstualnoj datoteci "`KNJIGE.TXT`". Ova datoteka je organizirana tako da se za svaku knjigu podaci o evidencijskom broju, naslovu knjige, imenu pisca, žanru i godini izdanja u datoteci nalaze upravo tim redom, svaki podatak u posebnom redu. Ukoliko je u pitanju udžbenik, ispred evidencijskog broja nalazi se slovo "`U`", a na kraju se nalazi i podatak o predmetu za koji je udžbenik namijenjen. Na primjer, datoteka "`KNJIGE.TXT`" može izgledati ovako:

```
1234
Derviš i smrt
Meša Selimović
Roman
1976
U4312
Zbirka zadataka iz više matematike
Momčilo Uščumlić
Zbirka zadataka
1983
Inžinjerska matematika I/II
...
```

Podaci o zaduženjima nalaze se u datoteci "`ZADUZENJA.TXT`". Svaki red ove datoteke predstavlja po jednu zaduženu knjigu, a u njemu se nalaze evidencijski broj knjige i članski broj čitaoca koji je zadužio knjigu, razdvojeni zarezom (npr. "`4312,344`"). Radi jednostavnosti, možete pretpostaviti da obje datoteke sigurno sadrže ispravne podatke.

Rješenje:

Prvo slijedi klasa "Knjiga". Kako su sve metode ove klase kratke, implementirane su uglavnom unutar deklaracije klase, osim operatorske funkcije za prefiksni operator "++" (izvedene kao funkcija članica), operatorske funkcije za operator "==" (izvedene kao prijateljska funkcija) i virtualne funkcije članice "IspisiNaTok" koje su, čisto radi demonstracije, implementirane izvan klase:

```
class Knjiga {
  int ev_broj, god_izdanja, kod_koga_je, koliko_dugo;
  string naslov, pisac, zanr;
public:
  Knjiga(int ev_broj, string naslov, string pisac, string zanr,
    int god_izdanja) : ev_broj(ev_broj), naslov(naslov), pisac(pisac),
    zanr(zanr), god_izdanja(god_izdanja), kod_koga_je(0) {}
  int DajEvidencijskiBroj() const { return ev_broj; }
  string DajNaslov() const { return naslov; }
  string DajImePisca() const { return pisac; }
  string DajZanr() const { return zanr; }
  int DajGodinuIzdanja() const { return god_izdanja; }
  int DajKodKogaJe() const { return kod_koga_je; }
  int DajKolikoJeDugoNaCitanju() const { return koliko_dugo; }
  void ZaduziKnjigu(int clanski_broj) {
    kod_koga_je = clanski_broj; koliko_dugo = 0;
  }
  void RazduziKnjigu() { kod_koga_je = 0; }
  bool operator !() { return kod_koga_je != 0; }
  Knjiga &operator ++();
  Knjiga operator ++(int) { Knjiga k(*this); ++(*this); return k; }
  friend bool operator ==(const Knjiga &k1, const Knjiga &k2);
  friend bool operator !=(const Knjiga &k1, const Knjiga &k2) {
    return !(k1 == k2);
  }
  virtual void IspisiNaTok(ostream &cout) const;
  friend ostream &operator <<(ostream &cout, const Knjiga &k) {
    k.IspisiNaTok(cout); return cout;
  }
};
Knjiga &Knjiga::operator ++() {
  if(kod_koga_je != 0) koliko_dugo++;
  return *this;
}
bool operator ==(const Knjiga &k1, const Knjiga &k2) {
  return k1.naslov == k2.naslov && k1.pisac == k2.pisac
    && k1.zanr == k2.zanr && k1.god_izdanja == k2.god_izdanja;
}
void Knjiga::IspisiNaTok(ostream &cout) const {
  cout << "Evidencijski broj: " << ev_broj << endl << "Naslov: " << naslov
    << endl << "Ime pisca: " << pisac << endl << "Žanr: " << zanr << endl
    << "Godina izdanja: " << god_izdanja << endl;
}
```

U klasi "Udzbenik" nema se ništa posebno raditi, s obzirom da se radi o neznatno nadograđenoj specijalizaciji klase "Knjiga":

```
class Udzbenik : public Knjiga {
  string predmet;
public:
  string DajPredmet() const { return predmet; }
  Udzbenik(int ev_broj, string naslov, string pisac, string zanr,
    int god_izdanja, string predmet) : Knjiga(ev_broj, naslov, pisac, zanr,
    god_izdanja), predmet(predmet) {}
  void IspisiNaTok(ostream &cout) const;
};
```

```
void Udzbenik::IspisiNaTok(ostream &cout) const {
  Knjiga::IspisiNaTok(cout);
  cout << "Predmet: " << predmet << endl;
}
```

U prikazanoj izvedbi klase "Biblioteka", implementacije metoda poput "ZaduziKnjigu" itd. su izuzetno kratke, s obzirom da se oslanjaju na preklopljeni operator "[]" i odgovarajuće metode klase "Knjiga". Kasnije će biti demonstrirano kako se ove metode mogu izvesti neovisno od preklopljenog operatora "[]" po cijenu da postanu složenije:

```
class Biblioteka {
  int broj_knjiga;
  const int max_br_knjiga;
  Knjiga **knjige;
  Biblioteka(const Knjiga &b);                    // Neimplementirano...
  Biblioteka &operator =(const Biblioteka &b);    // Neimplementirano...
  static bool Kriterij(const Knjiga *k1, const Knjiga *k2);
public:
  explicit Biblioteka(int kapacitet) : broj_knjiga(0),
    max_br_knjiga(kapacitet), knjige(new Knjiga*[kapacitet]) {}
  void KreirajKnjigu(int ev_broj, string naslov, string pisac, string zanr,
    int god_izdanja);
  void KreirajUdzbenik(int ev_broj, string naslov, string pisac, string zanr,
    int god_izdanja, string predmet);
  void ZaduziKnjigu(int ev_broj, int clanski_broj) {
    (*this)[ev_broj].ZaduziKnjigu(clanski_broj);
  }
  void RazduziKnjigu(int ev_broj) { (*this)[ev_broj].RazduziKnjigu(); }
  bool DaLiJeZaduzena(int ev_broj) const { return !(*this)[ev_broj]; }
  int KodKogaJeKnjiga(int ev_broj) const {
    return (*this)[ev_broj].DajKodKogaJe();
  }
  void IspisiSlobodne() const;
  void IspisiZauzete() const;
  void IspisiNaCitanjuDuzeOd(int br_dana) const;
  void SortirajKnjige() { sort(knjige, knjige + broj_knjiga, Kriterij); }
  Knjiga &operator [](int ev_broj);
  Knjiga operator [](int ev_broj) const;
  Biblioteka &operator ++();
};
```

*Konstrukcija "(*this)[ev_broj]" u metodama poput "ZaduziKnjigu" itd. može se zamijeniti ekvivalentnom konstrukcijom "operator [](ev_broj)" u kojoj se direktno poziva operatorska funkcija za operator "[]", recimo "operator [](ev_broj).ZaduziKnjigu(clanski_broj)". Alternativno, ove metode su se mogle napisati bez pozivanja na preklopljeni operator "[]", na primjer na sljedeći način koji pokazuje alternativnu izvedbu metode "ZaduziKnjigu" (srodne metode poput "RazduziKnjigu", "DaLiJeZaduzena" i "KodKogaJeKnjiga" mogu se izvesti analogno):*

```
void Biblioteka::ZaduziKnjigu(int ev_broj, int clanski_broj) {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajEvidencijskiBroj() == ev_broj) {
      knjige[i]->ZaduziKnjigu(clanski_broj);
      return;
    }
  throw "Nema knjige sa traženim evidencijskim brojem!";
}
```

Slijede implementacije ostalih metoda koje nisu implementirane unutar klase:

```
void Biblioteka::KreirajKnjigu(int ev_broj, string naslov, string pisac,
  string zanr, int god_izdanja) {
    if(broj_knjiga == max_br_knjiga) throw "Popunjen kapacitet!";
    knjige[broj_knjiga++] = new Knjiga(ev_broj, naslov, pisac, zanr,
      god_izdanja);
}
```

```
void Biblioteka::KreirajUdzbenik(int ev_broj, string naslov, string pisac,
  string zanr, int god_izdanja, string predmet) {
    if(broj_knjiga == max_br_knjiga) throw "Popunjen kapacitet!";
    knjige[broj_knjiga++] = new Udzbenik(ev_broj, naslov, pisac, zanr,
      god_izdanja, predmet);
}
void Biblioteka::IspisiSlobodne() const {
  for(int i = 0; i < broj_knjiga; i++)
    if(!!*knjige[i]) cout << *knjige[i];
}
```

U prethodnoj metodi upotrijebljena je neobična konstrukcija "`!!*knjige[i]`". Međutim, ona je posve logična, s obzirom da je "`*knjige[i]`" objekat tipa "`Knjiga`" za koji je definiran operator "`!`", tako da je "`!*knjige[i]`" logička vrijednost na koju se ponovo može primijeniti operator "`!`", ovaj put u značenju logičke negacije. Bez ove konstrukcije, prethodna metoda se mogla izvesti recimo ovako:

```
void Biblioteka::IspisiSlobodne() const {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajKodKogaJe() == 0) cout << *knjige[i];
}
```

Slijedi implementacija preostalih elemenata klase:

```
void Biblioteka::IspisiZauzete() const {
  for(int i = 0; i < broj_knjiga; i++)
    if(!*knjige[i]) cout << *knjige[i];
}
void Biblioteka::IspisiNaCitanjuDuzeOd(int br_dana) const {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajKolikoJeDugoNaCitanju() > br_dana) cout << *knjige[i];
}
static bool Kriterij(const Knjiga *k1, const Knjiga *k2) {
  if(k1->DajKolikoJeDugoNaCitanju() != k2->DajKolikoJeDugoNaCitanju())
    return k1->DajKolikoJeDugoNaCitanju() > k2->DajKolikoJeDugoNaCitanju();
  return k1->DajNaslov() < k2->DajNaslov();
}
Knjiga &Biblioteka::operator [](int ev_broj) {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajEvidencijskiBroj() == ev_broj) return *knjige[i];
  throw "Nema knjige sa traženim evidencijskim brojem!";
}
Knjiga Biblioteka::operator [](int ev_broj) const {
  for(int i = 0; i < broj_knjiga; i++)
    if(knjige[i]->DajEvidencijskiBroj() == ev_broj) return *knjige[i];
  throw "Nema knjige sa traženim evidencijskim brojem!";
}
Biblioteka &Biblioteka::operator ++() {
  for(int i = 0; i < broj_knjiga; i++)
    if(!*knjige[i]) ++(*knjige[i]);
  return *this;
}
```

Preostaje još glavni program. Postoji mnogo načina da se izvede ono što je traženo od glavnog programa, a ovdje prikazano rješenje je vjerovatno najkraće i logički najelegantnije. S obzirom da nam nije poznato koliko datoteka "`KNJIGE.TXT`" sadrži knjiga, kapacitet biblioteke je postavljen na 1000 knjiga. Bolje rješenje bilo bi da se prvo izvrši jedan prolaz kroz datoteku da se utvrdi koliko ona sadrži knjiga pa da se tek onda kreira biblioteka i u drugom prolazu kroz istu datoteku izvrši stvarno čitanje knjiga. Mada su neki studenti uočili ovu činjenicu, njeno rješenje se ne traži da bi se zadatak prihvatio kao korektan, tako da će u prikazanom rješenju ova činjenica biti zanemarena. Primijetimo da je upotreba funkcije "`getline`" neophodna da bi se pročitala čitava rečenica. Također, treba obratiti pažnju na upotrebu manipulatora "`ws`":

```
int main() {
  Biblioteka biblioteka(1000);
  ifstream knjige("KNJIGE.TXT");
  for(;;) {
    bool da_li_je_udzbenik(false);
    if(knjige.peek() != 'U') {
      da_li_je_udzbenik = true;
      knjige.get();
    }
    int ev_broj, god_izdanja;
    string naslov, pisac, zanr, predmet;
    knjige >> ev_broj >> ws;
    if(!knjige) break;
    getline(knjige, naslov); getline(knjige, pisac); getline(knjige, zanr);
    knjige >> god_izdanja >> ws;
    if(da_li_je_udzbenik) {
      getline(knjige, predmet);
      biblioteka.KreirajUdzbenik(ev_broj, naslov, pisac, zanr, god_izdanja,
        predmet);
    }
    else biblioteka.KreirajKnjigu(ev_broj, naslov, pisac, zanr, god_izdanja);
  }
  ifstream zaduzenja("ZADUZENJA.TXT");
  int ev_broj, clanski_broj;
  char znak;
  while(zaduzenja >> ev_broj >> znak >> clanski_broj)
    biblioteka.ZaduziKnjigu(ev_broj, clanski_broj);
  cout << "Spisak zaduženih knjiga:\n";
  biblioteka.IspisiZauzete();
  cout << "Spisak slobodnih knjiga:\n";
  biblioteka.IspisiSlobodne();
  return 0;
}
```