Riješeni zadaci uz Predavanje 1

Dolje navedeni zadaci po svom sadržaju prate Predavanje 1.

1. Neka je dat bazen oblika kvadra dimenzije $a \times b \times c$ i keramičke pločice oblika kvadrata dimenzija $d \times d$. Napisati program koji prvo zahtijeva od korisnika da unese dužinu, širinu i dubinu bazena (tj. a, b i c) u *metrima*, kao i širinu keramičke pločice d u *centimetrima*. Program zatim treba da ispita da li je bazen moguće popločati takvim pločicama, a da se pri tome nijedna pločica ne treba lomiti (debljinu pločice zanemariti). Ukoliko to nije moguće, treba ispisati odgovarajući komentar. Ukoliko jeste, treba ispisati koliko je pločica potrebno za popločavanje (naravno, popločavaju se zidovi i dno bazena). Slijede primjeri dva dijaloga između programa i korisnika:

```
Unesite duzinu, sirinu i dubinu bazena u metrima: 5 15 3
Unesite sirinu plocice u centimetrima: 10

Za poplocavanje bazena dimenzija 5x15x3m s plocicama dimenzija 10x10cm
potrebno je 19500 plocica.
```

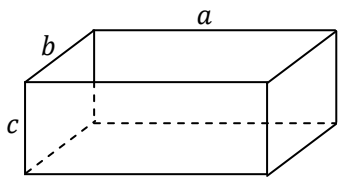
```
Unesite duzinu, sirinu i dubinu bazena u metrima: 5 15 3
Unesite sirinu plocice u centimetrima: 13

Poplocavanje bazena dimenzija 5x15x3m s plocicama dimenzija 13x13cm
nije izvodljivo bez lomljenja plocica!
```

U ispisu se ne koriste afrikati poput "č", "ć" itd. (to će vrijediti i ubuduće) zbog činjenice da nije sasvim jednostavno podesiti da se oni ispravno prikazuju na svim operativnim sistemima. Za unos podataka i ispis rezultata treba koristiti objekte "`cin`" i "`cout`" iz biblioteke "`iostream`". Također treba koristiti isključivo cjelobrojni tip podataka (tačnije tip "`int`"). Dozvoljeno je pretpostaviti da će ulazni podaci biti smisleni (tj. nije potrebno testirati da li je korisnik unio smislene podatke).

Rješenje:

Analizirajmo prvo ovaj zadatak s matematičkog aspekta. Površina koju treba popločati sastoji se od pet pravougaonika (četiri bočne stranice i dno). Dvije stranice su dimenzija $a \times c$, a dvije dimenzija $b \times c$, dok dno ima dimenziju $a \times b$, što se vidi sa sljedeće slike:



Površine ovih pravougaonika su ac, bc i ab respektivno, odakle slijedi da ukupna površina koju treba popločati iznosi

$$P = 2ac + 2bc + ab = 2c(a+b) + ab$$

Krajnji izraz smo malo preuredili da smanjimo neophodni broj množenja s 5 na 3. Kako su dimenzije bazena date u metrima, a dimenzije pločica u centimetrima, neophodno je prilagoditi mjerne jedinice, odnosno pretvoriti centimetre u metre, ili obratno. Međutim, pretvorba dužina zadanih u centimetrima u dužine izražene u metrima zahtijevala bi dijeljenje sa 100, tako da rezultati pretvorbe vjerovatno ne bi bili cijeli brojevi. Kako je u postavci rečeno da se koriste isključivo cjelobrojni tipovi podataka, pretvorićemo dužine zadane u metrima u dužine izražene u centimetrima. Ovo zahtijeva množenje sa 100, što neće narušiti cjelobrojnost. Stoga ćemo dužine a, b i c zadane u metrima pretvoriti u odgovarajuće dužine a', b' i c' izražene u centimetrima prema formulama

$$a' = 100\,a, \quad b' = 100\,b, \quad c' = 100\,c$$

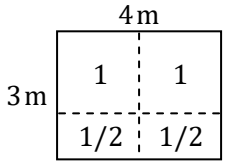
U skladu s tim, prilagodićemo i formulu za površinu u oblik

$$P' = 2c'(a'+b') + a'b'$$

da dobijemo površinu izraženu u kvadratnim centimetrima umjesto u kvadratnim metrima. Površina jedne pločice je naravno $P'' = d^2$, izražena također u kvadratnim centimetrima (s obzirom da je dužina stranice d zadana u centimetrima). Slijedi da broj pločica N koji je potreban da se poploča bazen možemo dobiti direktnim dijeljenjem površine bazena s površinom pločice, tj. kao

$$N = \frac{P'}{P''} = \frac{2c'(a'+b') + a'b'}{d^2}$$

Ostaje problem lomljenja pločica. Na prvi pogled, dovoljno je testirati da li je P' djeljivo s P'', jer ako jeste, N je cijeli broj, pa se koristi cijeli broj pločica. Međutim, to nije dovoljno, jer se može desiti da broj pločica ispadne cijeli broj, ali da su se ipak pločice morale lomiti (recimo da se upotrijebi dva puta po pola pločice da se pokrije površina jedne pločice). Čak nije dovoljno ni da površina svake stranice i dna zasebno bude djeljiva s površinom pločice. Kao primjer koji to ilustrira, uzmimo da su dužina i širina bazena 4 m i 3 m respektivno, a da je širina keramičke pločice 2 m (ovo je ogromna pločica, ali ovo je samo primjer za ilustraciju). Površina dna bazena je tada $4\,\text{m} \cdot 3\,\text{m} = 12\,\text{m}^2$, dok je površina pločice $(2\,\text{m})^2 = 4\,\text{m}^2$. Vidimo da je površina dna savršeno djeljiva s površinom pločice, odakle se zaključiti da su dovoljne $12/4 = 3$ pločice da se poploča dno. Ovo je istina, ali to nije moguće uraditi *bez lomljenja jedne pločice*, kao što se vidi na sljedećoj slici:



Nakon ukazivanja na ovu moguću brzopletu grešku, nije teško zaključiti da se popločavanje može izvesti bez lomljenja pločica ako i samo ako je *dužina svake od stranica bazena djeljiva s širinom pločice*, odnosno sva tri broja a', b' i c' moraju biti djeljivi s d (sretna je okolnost da je pločica oblika kvadrata, jer da i ona može biti proizvoljnog pravougaonog oblika, analiza bi bila neuporedivo komplikovanija). Djeljivost se lako provjerava nalaženjem ostatka pri dijeljenju i testiranja da li je taj ostatak jednak nuli. Dakle, ukoliko ostatak pri dijeljenju bilo kojeg od brojeva a', b' i c' s d nije jednak nuli, popločavanje nije moguće bez lomljenja. Provedena diskusija vodi ka sljedećem programu:

```
#include <iostream>

int main() {
  int a, b, c, d;
  std::cout << "Unesite duzinu, sirinu i dubinu bazena u metrima: ";
  std::cin >> a >> b >> c;
  std::cout << "Unesite sirinu plocice u centimetrima: ";
  std::cin >> d;
  std::cout << std::endl;
  int a1 = 100 * a, b1 = 100 * b, c1 = 100 * c;
  if(a1 % d != 0 || b1 % d != 0 || c1 % d != 0)
    std::cout << "Poplocavanje bazena dimenzija " << a << "x" << b << "x" << c
      << "m s plocicama dimenzija " << d << "x" << d << "cm\n"
      << "nije izvodljivo bez lomljenja plocica!\n";
  else
    std::cout << "Za poplocavanje bazena dimenzija " << a << "x" << b << "x" << c
      << "m s plocicama dimenzija " << d << "x" << d << "cm\n"
      << "potrebno je " << (2 * c1 * (a1 + b1) + a1 * b1) / (d * d) << " plocica.\n";
  return 0;
}
```

Treba primijetiti da za traženi broj pločica nismo rezervirali posebnu promjenljivu. Naime, kako nam ta informacija treba na samo jednom mjestu (unutar ispisa), umjesto korištenja dodatne promjenljive direktno smo unutar naredbe za ispis upotrijebili izraz kojim se taj broj računa. Pri tome, za računanje vrijednosti d^2 nismo koristili funkciju "`pow`", nego ponovljeno množenje (u formi podizraza "`d * d`"). Naime, funkcija "`pow`" posjeduje brojne nedostatke o kojima ćemo kasnije govoriti, tako da njenu upotrebu treba izbjeći po svaku cijenu, kad god je to moguće (a ovdje je to očito moguće), pogotovo kada se koristi cjelobrojna aritmetika. Također treba obratiti pažnju i na upotrebu zagrada koje obezbjeđuju da se u navedenom izrazu sve operacije izvedu u ispravnom redoslijedu. Recimo, ukoliko bismo zaboravili staviti podizraz "`d * d`" u zagradu, smatralo bi se da je

samo "d" u nazivniku, nakon čega se rezultat dijeljenja množi s "d" (efektivno bismo na kraju dobili samo vrijednost brojnika). Konačno, posljednja stvar na koju treba obratiti pažnju je kako je struktuirana naredba za ispis da bi se postiglo da ispis bude tačno onakav kako je to traženo u postavci zadatka.

Na kraju recimo još i ovo. Ukoliko nekome smeta stalno navođenje prefiksa "std::" ispred svih imena definiranih u standardnoj biblioteci, moguće je to izbjeći pomoću direktive "**using**". Međutim, ono što nikako nije dobro je uvesti kompletan imenik "std" koristeći konstrukciju "**using namespace** std", nego umjesto toga eksplicitno treba reći ispred kojih imena želimo da se podrazumijeva prefiks "std::". Tako se prethodni program mogao napisati i ovako:

```
#include <iostream>

using std::cout, std::cin, std::endl;

int main() {
  int a, b, c, d;
  cout << "Unesite duzinu, sirinu i dubinu bazena u metrima: ";
  cin >> a >> b >> c;
  cout << "Unesite sirinu plocice u centimetrima: ";
  cin >> d;
  cout << endl;
  int a1 = 100 * a, b1 = 100 * b, c1 = 100 * c;
  if(a1 % d != 0 || b1 % d != 0 || c1 % d != 0)
    cout << "Poplocavanje bazena dimenzija " << a << "x" << b << "x" << c
      << "m s plocicama dimenzija " << d << "x" << d << "cm\n"
      << "nije izvodljivo bez lomljenja plocica!\n";
  else
    cout << "Za poplocavanje bazena dimenzija " << a << "x" << b << "x" << c
      << "m s plocicama dimenzija " << d << "x" << d << "cm\n"
      << "potrebno je " << (2 * c1 * (a1 + b1) + a1 * b1) / (d * d) << " plocica.\n";
  return 0;
}
```

Treba napomenuti da je mogućnost da se jednom "**using**" direktivom navede više idenfikatora uvedena tek od standarda C++17. U ranijim standardima jezika C++, umjesto jedne direktive

```
using std::cout, std::cin, std::endl;
```

potrebno je koristiti posebnu direktivu za svaki identifikator, odnosno pisati nešto poput

```
using std::cout;
using std::cin;
using std::endl;
```

što je, bez ikakve dileme, naporno. Srećom, ovo je riješeno u standardu C++17.

2. Napisati program koji traži da se sa tastature unesu tri realna broja *a*, *b* i *c*, i koji ispisuje da li ta tri broja mogu biti stranice nekog trougla. Ukoliko uneseni brojevi mogu predstavljati dužine stranica trougla, treba izračunati njegov obim, površinu i najmanji ugao, a zatim ispisati izračunate vrijednosti obima, površine i najmanjeg ugla, pri čemu ugao treba ispisati u stepenima, minutama i sekundama. Ukoliko uneseni brojevi ne mogu predstavljati dužine stranica trougla, treba ispisati odgovarajući komentar. Dijalozi koje formira program trebaju izgledati poput sljedećih (pri čemu je dozvoljeno zanemariti eventualne probleme sa padežima i ostalom "nezgodnom" gramatikom):

```
Unesite tri broja: 7 5 8
Obim trougla sa duzinama stranica 7, 5 i 8 iznosi 20.
Njegova povrsina iznosi 17.3205.
Njegov najmanji ugao ima 38 stepeni, 12 minuta i 47 sekundi.
```

```
Unesite tri broja: 5 15 7
Ne postoji trougao cije su duzine stranica 5, 15 i 7!
```

Za unos podataka i ispis rezultata treba koristiti objekte "cin" i "cout" iz biblioteke "iostream", a za odgovarajuća računanja funkcije iz biblioteke "cmath". Dozvoljeno je pretpostaviti da će korisnik zaista unositi brojeve (ne nužno cijele), a ne neko "smeće".

Rješenje:

Da bi tri realna broja a, b i c mogli biti stranice nekog trougla, potrebno je i dovoljno da su sve pozitivne i da je zbr dužina ma koje dvije stranice veći od dužine treće stranice. Ovo se može napisati u vidu složenog uvjeta

$$a>0 \wedge b>0 \wedge c>0 \wedge a+b>c \wedge a+c>b \wedge b+c>a$$

Slijedi da tri realna broja a, b i c ne mogu biti stranice nikakvog trougla ukoliko vrijedi negacija prethodnog uvjeta, odnosno ukoliko vrijedi

$$a\leq 0 \vee b\leq 0 \vee c\leq 0 \vee a+b\leq c \vee a+c\leq b \vee b+c\leq a$$

U slučaju da je moguće formirati trougao, njegov obim je naravno $O=a+b+c$, dok se površina može izračunati pomoću poznate Heronove formule prema kojoj je

$$P=\sqrt{s(s-a)(s-b)(s-c)}$$

gdje je $s=O/2$ (polovica obima). Za računanje uglova, koristićemo kosinusnu teoremu (popoćenje Pitagorine teoreme za nepravougle trouglove), prema kojoj je

$$c^2=a^2+b^2-2ab\cos\gamma$$

Odavde slijedi da je

$$\gamma=\arccos\frac{a^2+b^2-c^2}{2ab}$$

Analogne formule postoje za preostale kombinacije stranica i uglova, tako da je

$$\alpha=\arccos\frac{b^2+c^2-a^2}{2bc}$$

$$\beta=\arccos\frac{a^2+c^2-b^2}{2ac}$$

Za računanje funkcije arc cos koristićemo funkciju "`acos`" iz biblioteke s zaglavljem "`cmath`". U zadatku se traži da se izračuna samo najmanji ugao, ali nije moguće znati koji je najmanji dok ne izračunamo sva tri ugla α, β i γ i među njima poređenjem odaberemo najmanji. Naime, ukoliko je $\alpha<\beta$ i $\alpha<\gamma$, najmanji ugao je α. U suprotnom, ukoliko je $\beta<\gamma$, najmanji ugao je β (nema potrebe provjeravati i da li je $\beta<\alpha$, jer ako smo ranije ustanovili da ne vrijedi istovremeno $\alpha<\beta$ i $\alpha<\gamma$, to znači da sigurno nije $\alpha<\beta$, tako da bi taj test bio suvišan, iako ne bi bio neispravan). Konačno, ukoliko ne vrijedi nijedna od opisane dvije situacije, najmanji ugao je γ.

Funkcija "`acos`" daje rezultat izražen u radijanima, tako da će nađeni najmanji ugao također biti izražen u radijanima. Da bismo ga izrazili u stepenima, koristimo činjenicu da π radijana sadrži 180 stepeni, odnosno 1 radijan sadrži $180/\pi$ stepeni. Slijedi da rezultat u radijanima treba pomnožiti sa $180/\pi$ da dobijemo rezultat izražen u stepenima. Što se tiče broja π, možemo iskoristiti poznatu činjenicu da je $\operatorname{arctg} 1=\pi/4$, odakle slijedi da je $\pi=4\cdot\operatorname{arctg} 1$. Za računanje funkcije arc tg možemo koristiti funkciju "`atan`" iz biblioteke s zaglavljem "`cmath`". Ovo je neuporedivo bolje nego korištenje vrlo loših aproksimacija kao što je $\pi\approx 3.14$, ili aproksimacija poput $\pi\approx 3.14159265358979$ (ova posljednja aproksimacija je dobra, onoliko tačna koliko može stati u tip "**`double`**", samo ju je teško zapamtiti). Na osnovu činjenice da je $\pi=4\cdot\operatorname{arctg} 1$, slijedi da rezultat u radijanima treba pomnožiti s $45/\operatorname{arctg} 1$ da dobijemo rezultat izražen u stepenima.

Ovim još nismo riješili sve što se traži. Naime, opisanim postupkom dobićemo ugao izražen u stepenima kao realna vrijednost, recimo kao 3.3375 stepeni, a traži se da se taj rezultat izrazi posebno u vidu cijelog broja stepeni, minuta i sekundi (prethodni rezultat trebalo bi izraziti kao 3 stepena, 20 minuta i 15 sekundi, s obzirom da je minuta 60-ti dio stepena, a sekunda 60-ti dio minute). To razlaganje možemo obaviti ovako. Cijeli dio rezultata predstavlja traženi broj stepeni. Preostalu vrijednost, koju dobijemo kad odbacimo cijeli dio (u gornjem primjeru to je 0.3375 stepeni) treba pomnožiti sa 60 da dobijemo broj minuta izražen ne nužno kao cijeli broj (u gornjem primjeru to će biti 20.25). Ponovo, cijeli dio ovog rezultata daje broj minuta, a preostali dio (0.25 u

gornjem primjeru) treba pomnožiti sa 60 da dobijemo broj sekundi. Po potrebi bi mogli postupak nastaviti analogno da odredimo broj subsekundi (60-ti dio sekunde), ali to se ne traži. Kada sve što smo dosada rekli sastavimo u jednu cjelinu, dobijamo rješenje u vidu sljedećeg programa:

```
#include <iostream>
#include <cmath>

int main() {
  double a, b, c;
  std::cout << "Unesite tri broja: ";
  std::cin >> a >> b >> c;
  if(a <= 0 || b <= 0 || c <= 0 || a + b <= c || a + c <= b || b + c <= a)
    std::cout << "Ne postoji trougao cije su duzine stranica "
      << a << ", " << b << " i " << c << "!\n";
  else {
    double s = (a + b + c) / 2;
    double p = std::sqrt(s * (s - a) * (s - b) * (s - c));
    double alfa = std::acos((b * b + c * c - a * a) / (2 * b * c));
    double beta = std::acos((a * a + c * c - b * b) / (2 * a * c));
    double gama = std::acos((a * a + b * b - c * c) / (2 * a * b));
    double minugao;
    if(alfa < beta && alfa < gama) minugao = alfa;
    else if(beta < gama) minugao = beta;
    else minugao = gama;
    minugao *= 45 / std::atan(1);
    double minute = 60 * (minugao - int(minugao));
    double sekunde = 60 * (minute - int(minute));
    std::cout << "Obim trougla sa duzinama stranica " << a << ", " << b
      << " i " << c << " iznosi " << 2 * s << ".\n"
      << "Njegova povrsina iznosi "  << p << ".\n"
      << "Njegov najmanji ugao ima " << int(minugao) << " stepeni, "
      << int(minute) << " minuta i " << int(sekunde) << " sekundi.\n";
  };
  return 0;
}
```

Nakon prethodno provedene analize i diskusije, program bi trebao da bude sam po sebi dovoljno jasan. Primijetimo da se u njemu ne računa obim O u posebnoj promjenljivoj, nego samo poluobim s (koji se koristi na više mjesta u Heronovom obrascu), dok se na mjestu gdje treba prikazati obim on računa koristeći relaciju $O = 2s$.

3. Napisati program koji traži da se sa tastature unesu dva prirodna broja a i b, ne veća od 9999 i pri čemu je $a < b$, a koji zatim za sve prirodne brojeve u opsegu od a do b uključivo ispisuje tablicu njihovih kvadrata, kvadratnih korijena i prirodnih logaritama. Tačan izgled tablice vidljiv je iz dijaloga koji će biti prikazan. Uglavnom, kolona za prikaz brojeva široka je 9 polja. Brojeve treba ispisivati poravnate *ulijevo*, pri čemu je prvo polje uvijek razmak. Kolone za prikaz kvadrata i korijena široke su 10 polja, dok je kolona za prikaz logaritama široka 11 polja. Kvadrate, korijene i logaritme treba ispisivati poravnate *udesno*, pri čemu je posljednje polje uvijek razmak. Pored toga, korijene treba ispisivati fiksno na tri decimale, a logaritme fiksno na pet decimala. Slijedi primjer kako treba izgledati dijalog između korisnika i programa:

```
Unesite pocetnu i krajnu vrijednost: 95 103

+---------+----------+----------+-----------+
| Brojevi | Kvadrati | Korijeni | Logaritmi |
+---------+----------+----------+-----------+
| 95      |     9025 |    9.747 |   4.55388 |
| 96      |     9216 |    9.798 |   4.56435 |
| 97      |     9409 |    9.849 |   4.57471 |
| 98      |     9604 |    9.899 |   4.58497 |
| 99      |     9801 |    9.950 |   4.59512 |
| 100     |    10000 |   10.000 |   4.60517 |
| 101     |    10201 |   10.050 |   4.61512 |
| 102     |    10404 |   10.100 |   4.62497 |
| 103     |    10609 |   10.149 |   4.63473 |
+---------+----------+----------+-----------+
```

Za ispis treba koristiti objekat "cout" iz biblioteke "iostream" i odgovarajuće manipulatore iz iste biblioteke, kao i biblioteke "iomanip". U programu ne treba testirati da li su a i b zaista prirodni brojevi ne veći od 9999, i da li je $a \leq b$. Međutim, potrebno je ispitati kako će se program ponašati ukoliko se unesu brojevi koji ne zadovoljavaju ova ograničenja.

Rješenje:

U ovom zadatku jedino što je problematično je postići traženi format ispisa. Za tu svrhu, treba koristiti manipulatore "left" i "right" za poravnavanje ulijevo ili udesno u rezervirani prostor, zatim "setw" za podešavanje širine prostora rezerviranog za ispis, "setprecision" za podešavanje preciznosti ispisa, te "fixed" za forsiranje ispisa s fiksnim brojem decimala. Radi korištenja manipulatora "setw" i "setprecision" potrebno je u program uključiti biblioteku s zaglavljem "iomanip". Alternativno, umjesto njih možemo koristiti poziv funkcija "width" i "precision" nad objektom izlaznog toka "cout", što ne bi tražilo biblioteku s zaglavljem "iomanip", ali bismo morali prekidati tok prilikom svakog poziva tih funkcija. Prirodni logaritam možemo dobiti pomoću funkcije "log" iz biblioteke sa zaglavljem "cmath". Kada uzmemo u obzir sve što je rečeno, možemo napisati program poput sljedećeg:

```
#include <iostream>
#include <iomanip>
#include <cmath>

int main() {
  int xmin, xmax;
  std::cout << "Unesite pocetnu i krajnju vrijednost: ";
  std::cin >> xmin >> xmax;
  std::cout << "\n"
    << "+---------+----------+----------+-----------+\n"
    << "| Brojevi | Kvadrati | Korijeni | Logaritmi |\n"
    << "+---------+----------+----------+-----------+\n";
  for(int x = xmin; x <= xmax; x++)
    std::cout << "| " << std::left << std::setw(8) << x << "|"
      << std::right << std::setw(9) << x * x << " |" << std::fixed
      << std::setw(9) << std::setprecision(3) << std::sqrt(x) << " |"
      << std::setw(10) << std::setprecision(5) << std::log(x) << " |\n";
  std::cout << "+---------+----------+----------+-----------+\n";
  return 0;
}
```

Ostaje još da analiziramo šta će se dogoditi ukoliko uneseni brojevi a i b nisu prirodni brojevi ne veći od 9999 za koje vrijedi da li je $a \leq b$. U slučaju da je $a > b$, biće ispisano samo zaglavlje i podnožje tabele, jer se petlja u kojoj se obavlja račun i ispis provedenog računa neće izvršiti ni jedanput (uvjet pod kojim se petlja izvodi odmah će na početku biti netačan). Ukoliko je a negativan broj ili nula, imaćemo problema pri računanju kvadratnog korijena i logaritma, jer kvadratni korijeni nisu definirani za negativne brojeve (logaritam nije definiran ni za nulu), barem ne u skupu realnih brojeva. Ponašanje matematičkih funkcija u tom slučaju nije potpuno predviđeno standardom, ali kompajleri koji koriste tzv. IEEE 754 format zapisa realnih brojeva (a to su gotovo svi današnji kompajleri) u takvim slučajevima vraćaju kao rezultat specijalne objekte realnog tipa, koji se zovu ne-brojevi (engl. not-a-number, skraćeno NaN), koji se na ekranu tipično ispisuju kao riječ "nan" ili ponekad kao "ind" (od engl. indeterminate). Također, prema istom standardu, za logaritam od nule kao rezultat se dobija specijalni objekat realnog tipa koji predstavlja negativnu beskonačnost, koji se obično ispisuje kao "-inf" (od engl. infinity).

Najčudnije ponašanje je u slučaju da unesemo nešto što nije broj. Recimo, ukoliko za a probamo unijeti nešto što nije broj, ulazni tok će doći u neispravno stanje. Postavlja se pitanje kakav će biti sadržaj promjenljive u koju je pokušano izdvajanje u slučaju da se to dogodi. Tu je situacija dosta neodređena, ali generalno ne bi se trebalo osloniti na to kakav će biti njen sadržaj u tom slučaju (to je razlog zbog kojeg bi u ozbiljnim programima uvijek trebalo testirati stanje toka nakon pokušaja unosa). Konkretno, stariji standardi jezika C++ (prije C++11) ostavljaju njen sadržaj onakvim kakav je bio prije pokušaja izdvajanja, dok noviji standardi (od C++11 nadalje) u nju upisuju nulu. Dakle, vrlo vjerovatno će u odgovarajućoj promjenljivoj biti nula. U drugoj promjenljivoj će tada također vjerovatno biti nula, jer drugo čitanje neće ni biti pokušano (tok je došao u neispravno stanje, a nismo izvršili njegov oporavak pozivom funkcije "clear" nad objektom toka). Dakle, program će se vjerovatno ponašati kao da smo unijeli dvije nule. Ukoliko je prvi broj unesen korektno, a drugi ne,

ponašanje će vjerovatno biti kao da je drugi uneseni broj nula. Konačno, ako za prvi broj unesemo realni broj, npr. "`25.342`", ispravno će biti pročitan broj 25, nakon čega će izdvajanje stati na prvom znaku koji ne pripada cijelom broju, tj. znaku "`.`". Znakovi "`.342`" ostaju u spremniku toka, tako da sljedeće čitanje neće uspjeti (jer već prvi znak nije cifra). Vjerovatno ponašanje biće kao da je drugi uneseni broj nula. Kao što vidimo, ima dosta neodređenosti u ponašanju programa u takvom slučaju, što je još jedan razlog zbog kojeg u ozbiljnim programima uvijek treba testirati validnost unosa.

4. U numerologiji starih Grka, svi prirodni brojevi su se dijelili u tri kategorije, prema tome kakva im je suma svih njihovih djelilaca (ne računajući njega samog). Oni brojevi kod kojih je ta suma manja od samog broja nazivali su se *manjkavi* (engl. *deficient*), oni kod kojih je ta suma veća od samog broja nazivali su se *obilni* (engl. *abudant*), dok su oni brojevi koji su jednaki sumi svih svojih djelilaca (bez njega samog) nazivali *savršeni* (engl. *perfect*). Recimo, broj 49 je manjkav: njegovi djelioci su 1 i 7, pri čemu je $1+7=8<49$. Isto tako, broj 42 je obilan: njegovi djelioci su 1,2, 3, 6, 7, 14 i 21, pri čemu je $1+2+3+6+7+14+21=54>42$. Konačno, 28 je primjer savršenog broja: njegovi djelioci su 1, 2, 4, 7 i 14, a vrijedi $1+2+4+7+14=28$.

   Napisati program koji traži da se sa tastature unese prirodan broj *n*. U slučaju da korisnik unese nešto što nije prirodan broj (što uključuje i situaciju kada uneseni podatak uopće nije broj), treba ispisati poruku upozorenja, i ponoviti unos. Ukoliko je unos ispravan, program treba da ispita i ispiše da li je uneseni broj manjkav, obilan ili savršen. Nakon toga, program treba da traži unos novog broja i da ponavlja postupak sve dok se kao broj ne unese nula. Dijalog između programa i korisnika trebao bi izgledati poput sljedećeg:

```
Unesite prirodan broj ili 0 za kraj: 42
Broj 42 je obilan!
Unesite prirodan broj ili 0 za kraj: 28
Broj 28 je savrsen!
Unesite prirodan broj ili 0 za kraj: 49
Broj 49 je manjkav!
Unesite prirodan broj ili 0 za kraj: -1
Niste unijeli prirodan broj!
Unesite prirodan broj ili 0 za kraj: 2.15
Niste unijeli prirodan broj!
Unesite prirodan broj ili 0 za kraj: qwerty
Niste unijeli prirodan broj!
Unesite prirodan broj ili 0 za kraj: 0
Dovidjenja!
```

Rješenje:

S obzirom da se traži da program radi sve dok korisnik ne unese nulu, cijeli program ćemo izvesti u formi jedne velike do-while petlje, koja se izvodi sve dok je uneseni broj različit od nule. Prva stvar koju u toj petlji treba uraditi, pored samog unosa, je testiranje da li je unos validan. Situaciju u kojoj uopće nije unesen broj, možemo testirati ispitujući da li je ulazni tok u neispravnom stanju, koristeći operator negacije. Da bi broj bio prirodan, mora biti cijeli i pozitivan. Dakle, ukoliko je broj negativan, unos sigurno nije ispravan. Također, unos nije ispravan ni ako smo unijeli broj koji nije cijeli. Postavlja se pitanje kako razlikovati situaciju kada uneseni broj nije cijeli od situacije kada je unesen cijeli broj. Ukoliko bismo unos vršili u cjelobrojnu promjenljivu, ne bismo mogli lako detektirati da je unesen necijeli broj. Zaista, ukoliko bismo unijeli recimo "`253.72`", zbog načina kako radi operator izdvajanja "`>>`" bile bi pročitane cifre "`2`", "`5`" i "`3`", nakon čega bi izdvajanje stalo na znaku "`.`", koji ne pripada cijelom broju (ovo smo već nagovijestili prilikom analize prethodnog zadatka). Dakle, bio bi (ispravno) pročitan broj 253, a u spremniku toka bi ostali znakovi "`.72`", koji bi čekali eventualno kasnije čitanje. Drugim riječima, broj bi bio pročitan (do tačke), a ulazni tok bi bio u ispravnom stanju, tako da bi jedini način da testiramo da nešto nije u redu bio da provjerimo da li je ostalo nepročitanih znakova u spremniku, a to sa znanjem nakon samo prvog predavanja još nismo u stanju. Da bismo riješili ovaj problem, unos ćemo, umjesto u cjelobrojnu, unositi u realnu promjenljivu (tipa "`double`"), nakon čega ćemo testirati da li je njen sadržaj zaista cjelobrojan. Na taj način, ukoliko unesemo nešto poput "`253.72`", korektno će se u promjenljivu pročitati čitav broj 253.72, ali ćemo kasnijim testiranjem utvrditi da taj broj nije cijeli. Testiranje na cjelobrojnost sadržaja neke promjenljive najlakše možemo izvesti tako što ćemo uzeti njen cijeli dio (konverzijom u tip "`int`") i porediti je s njenim nekonvertovanim sadržajem. Ukoliko se to dvoje razlikuje, sadržaj nije cjelobrojan. Dakle, smatraćemo da unos nije bio korektan ukoliko je tok u neispravnom stanju,

ili ukoliko je uneseni broj negativan, ili ukoliko uneseni broj nije cjelobrojan. Primijetimo da, iako se traži samo unos prirodnih brojeva, ipak moramo dozvoliti i unos nule (koja nije pozitivan broj niti prirodan broj), s obzrom da po postavci zadatka nula služi za terminiranje programa.

U slučaju neispravnog unosa, prijavićemo grešku, oporaviti tok (pozivom funkcije "`clear`" nad objektom toka) i obrisati spremnik toka koji sadrži nepročitane znakove (pozivom funkcije "`ignore`" nad objektom toka), s obzirom da ne znamo šta je sve korisnik mogao unijeti. Međutim, treba obratiti pažnju na još jedan važan detalj. Naime, postavlja se pitanje kakav će biti sadržaj promjenljive u koju je pokušano izdvajanje u slučaju da tok dospije u neispravno stanje. Kao što smo nagovijestili u prethodnom zadatku, tu je situacija dosta neodređena, tako da se ne bismo trebali osloniti na to kakav će biti njen sadržaj u tom slučaju (rekli smo ranije da stariji standardi jezika C++ ostavljaju njen sadržaj onakvim kakav je bio prije pokušaja izdvajanja, dok noviji standardi u nju upisuju nulu). Dakle, postoji mogućnost da će u nju biti upisana nula. Nažalost, ukoliko dođe do upisa nule, to će nam prekinuti do-while petlju u kojoj se program izvodi, jer će se smatrati da smo zaista unijeli nulu, koja služi za terminiranje. Stoga ćemo, u slučaju pogrešnog unosa, vještački upisati neku nenultu vrijednost (recimo 1) u promjenljivu u koju vršimo unos, da obezbijedimo da do-while petlja neće biti prekinuta.

Razmotrimo sada šta treba raditi nakon obavljenog korektnog unosa. Naime, potrebno je naći sumu djelilaca unesenog broja, da bismo zaključili u koju kategoriju spada (pri čemu ovo procesiranje ne treba raditi ukoliko je uneseni broj nula, jer nula služi samo kao terminator). Djelioce broja ćemo naći tako što ćemo for petljom proći kroz sve brojeve od 1 do polovice unesenog broja, pri čemu ćemo testirati da li je uneseni broj djeljiv s njima. Ukoliko jeste, to je djelilac i treba ga dodati na sumu (koja je na početku 0). Petlja ne treba ići dalje od polovice unesenog broja, s obzirom da nijedan broj ne može imati neki djelilac koji je veći od njegove polovice (osim njega samog). Ovdje samo treba obratiti pažnju na jednu sitnicu. Uneseni broj se nalazi u realnoj promjenljivoj, a operator "`%`", koji nam treba za testiranje djeljivosti, ne radi s operandima realnog tipa (prijavljuje sintaksnu grešku). Stoga ćemo kao operand operatoru "`%`" poslati vrijednost te promjenljive konvertovanu u cjelobrojni tip (ta konverzija neće dovesti do gubitka informacija, jer će vrijednost promjenljive nakon ispravnog unosa biti cjelobrojna, iako je realnog tipa). Nakon što nađemo sumu djelilaca, poređenjem te sume sa vrijednošću broja lako ćemo ustanoviti u koju kategoriju on spada (manjkav, obilan, ili savršen).

Nakon provedene detaljne analize, lako je sastaviti i sam program. Slijedi njegov prikaz, koji je lako pratiti nakon prethodno datih objašnjenja:

```
#include <iostream>

int main() {
  double x;
  do {
    std::cout << "Unesite prirodan broj ili 0 za kraj: ";
    std::cin >> x;
    if(!std::cin || x < 0 || int(x) != x) {
      std::cout << "Niste unijeli prirodan broj!\n";
      std::cin.clear();
      std::cin.ignore(10000, '\n');
      x = 1;                          // Sprečava prekid petlje!
    }
    else if(x != 0) {
      int suma = 0;
      for(int i = 1; i <= int(x) / 2; i++)
        if(int(x) % i == 0) suma += i;
      std::cout << "Broj " << x << " je ";
      if(suma < x) std::cout << "manjkav!\n";
      else if(suma > x) std::cout << "obilan!\n";
      else std::cout << "savrsen!\n";
    }
  } while(x != 0);
  std::cout << "Dovidjenja!\n";
  return 0;
}
```

Ovaj program ipak ima i jedan nedostatak. Unesemo li nešto poput "`123xy`", biće ispravno pročitan broj 123, nakon čega će u spremniku toka ostati znakovi "`xy`". Program će ispravno zaključiti da je

broj 123 manjkav i ispisati komentar o tome, ali će preostali znakovi "xy", koji su ostali u spremniku toka, napraviti problem pri sljedećem unosu. Ovaj problem možemo djelimično riješiti tako što ćemo funkciju "ignore" također pozivati i u slučaju korektnog unosa (da obrišemo eventualni "višak") koji je preostao prilikom unosa. Ovo rješenje je "djelimično" zbog činjenice da se unos poput "123xy" tretira kao ispravan, samo što se "višak" u formi znakova "xy" prosto ignorira. Ukoliko želimo da se i ulaz poput "123xy" tretira kao neispravan, moramo nakon obavljenog čitanja testirati i da li je ostalo "viška" znakova u spremniku toka. Međutim, kao što smo već nagovijestili u prethodnom zadatku, to nije moguće samo sa znanjem stečenim nakon prvog predavanja.

5. Napisati program koji prvo traži da se s tastature unese prirodan broj n, a nakon toga, n prirodnih brojeva, takvih da mogu stati u promjenljive tipa "int". Nakon obavljenog unosa, program treba ispisati kako glase najmanji i najveći među unesenim brojevima. Za realizaciju programa ne koristiti nizove ili neke slične strukture podataka, nego samo individualne cjelobrojne promjenljive. Nije potrebno provjeravati ispravnost unosa, odnosno dozvoljeno je pretpostaviti da će korisnik zaista unijeti ispravan prirodan broj n i da će unositi ispravne cijele brojeve tamo gdje se to očekuje.

   Rješenje:

   U osnovi, ovo je jednostavan zadatak, ali je zanimljiv zbog činjenice da se može uraditi na veliki broj različitih načina, tako da ćemo razmotriti nekoliko interesantnih rješenja. Glavna ideja je da se u svakom trenutku pamtiti najmanji i najveći broj među dotada unesenim brojevima, recimo u dvije promjenljive koje ćemo nazvati "najmanji_dosad" i "najveci_dosad". Da bismo izbjegli upotrebu nizova, svaki broj ćemo unositi u jednu te istu promjenljivu, koju ćemo nazvati "tekuci_broj". Nakon unosa svakog od brojeva, poredićemo uneseni broj sa zapamćenom dotada najmanjom odnosno najvećom vrijednošću, i vršićemo ažuriranje ovih vrijednosti ukoliko je novouneseni broj manji odnosno veći od dotada najmanjeg odnosno najvećeg unesenog broja. Jedina je dilema kako postupati s prvim unesenim brojem, jer prije njega nije bilo "dotada najmanjeg i najvećeg broja", odnosno nemamo ga s čime porediti. Jedna ideja je da prvi broj tretiramo posebno, izvan petlje, odnosno da nakon unosa prvog broja iskoristimo njegovu vrijednost za inicijalizaciju promjenljivih "najmanji_dosad" i "najveci_dosad", a da tek nakon toga uđemo u petlju u kojoj ćemo procesirati ostale unesene brojeve na opisani način. To vodi ka rješenju poput sljedećeg:

```
#include <iostream>

int main() {
  int n;
  std::cout << "Koliko zelite unijeti brojeva? ";
  std::cin >> n;
  std::cout << "Unesite brojeve:" << std::endl;
  int tekuci_broj;
  std::cout << "1. broj: ";
  std::cin >> tekuci_broj;
  int najmanji_dosad = tekuci_broj, najveci_dosad = tekuci_broj;
  for(int i = 2; i <= n; i++) {
    std::cout << i << ". broj: ";
    std::cin >> tekuci_broj;
    if(tekuci_broj < najmanji_dosad) najmanji_dosad = tekuci_broj;
    if(tekuci_broj > najveci_dosad) najveci_dosad = tekuci_broj;
  }
  std::cout << "Najmanji medju unesenim brojevima je " << najmanji_dosad
    << ", a najveci medju unesenim brojevima je " << najveci_dosad << std::endl;
  return 0;
}
```

   Ovo rješenje nesporno radi, ali je isto tako njegova neelegancija očigledna, s obzirom da se prvi uneseni broj tretira posve drugačije od ostalih i izvan petlje u kojoj se obrađuju ostali brojevi. To, između ostalog, ima i neugodnu posljedicu da smo naredbu za unos "std::cin >> tekuci_broj" morali ponoviti na dva mjesta, kao i da smo promjenljivu "tekuci_broj", koja ima samo lokalnu ulogu, morali deklarirati ispred petlje, čime ona nastavlja "živjeti" i nakon petlje, odnosno znatno duže nego što je potrebno (što se generalno smatra lošom praksom, koja se posebno pokazuje štetnom u dužim programima). Prvi broj bismo također mogli procesirati unutar petlje ukoliko bismo pomoću "if" naredbe posebno provjeravali da li je u pitanju prvi uneseni broj, tj. prvi prolaz kroz petlju (testiranjem uvjeta "i == 1"). Ukoliko se radi o prvom broju, tada njegovu vrijednost koristimo za postavljanje početnih vrijednosti promjenljivih "najmanji_dosad" i "najveci_dosad", a

u suprotnom, postupamo kao i ranije. To vodi ka sljedećem rješenju, u kojem je za postavku početnih vrijednosti promjenljivih "najmanji_dosad" i "najveci_dosad" iskorišten izraz oblika "$x = y = z$" koji se izvšava zdesna nalijevo, odnosno prvo se vrijednost "z" kopira u "y", nakon čega se kopirana vrijednost također kopira i u "x", tako da će, na kraju, "x" i "y" imati iste vrijednosti kao i "z"):

```
#include <iostream>
int main() {
  int n;
  std::cout << "Koliko zelite unijeti brojeva? ";
  std::cin >> n;
  std::cout << "Unesite brojeve:" << std::endl;
  int najmanji_dosad, najveci_dosad;
  for(int i = 1; i <= n; i++) {
    int tekuci_broj;
    std::cout << i << ". broj: ";
    std::cin >> tekuci_broj;
    if(i == 1) najmanji_dosad = najveci_dosad = tekuci_broj;
    else {
      if(tekuci_broj < najmanji_dosad) najmanji_dosad = tekuci_broj;
      if(tekuci_broj > najveci_dosad) najveci_dosad = tekuci_broj;
    }
  }
  std::cout << "Najmanji medju unesenim brojevima je " << najmanji_dosad
    << ", a najveci medju unesenim brojevima je " << najveci_dosad << std::endl;
  return 0;
}
```

Dobra stvar kod ovog rješenja je što je sad promjenljiva "tekuci_broj" lokalizirana (život joj je ograničen do završetka petlje, ali i dalje se prvi uneseni broj tretira drugačije nego ostali, doduše unutar petlje, što je postignuto korištenjem "**if**" – "**else**" konstrukcije. Moguće je neznatno skratiti ovo rješenje, uz upotrebu karakteristične fraze "**else if**", kao u sljedećoj izvedbi

```
#include <iostream>
int main() {
  int n;
  std::cout << "Koliko zelite unijeti brojeva? ";
  std::cin >> n;
  std::cout << "Unesite brojeve:" << std::endl;
  int najmanji_dosad, najveci_dosad;
  for(int i = 1; i <= n; i++) {
    int tekuci_broj;
    std::cout << i << ". broj: ";
    std::cin >> tekuci_broj;
    if(i == 1) najmanji_dosad = najveci_dosad = tekuci_broj;
    else if(tekuci_broj < najmanji_dosad) najmanji_dosad = tekuci_broj;
    else if(tekuci_broj > najveci_dosad) najveci_dosad = tekuci_broj;
  }
  std::cout << "Najmanji medju unesenim brojevima je " << najmanji_dosad
    << ", a najveci medju unesenim brojevima je " << najveci_dosad << std::endl;
  return 0;
}
```

Naravno, u obje izvedbe, uloga ključne riječi "**else**" je sprečavanje da se poređenje unesenog broja s dotada najmanjim i najvećim brojem izvrši nakon unosa prvog broja, odnosno ukoliko je uvjet "`i == 1`" tačan. Međutim, interesantno je da će sve raditi i bez "**else**" (tj. samo s tri čiste "**if**" naredbe), iako je to logički "prljavije". Zaista, ukoliko uklonimo "**else**", poređenja će se vršiti i u prvom prolasku kroz petlju, tj. kada je uvjet "`i == 1`" tačan. Međutim, odmah nakon provjere tog uvjeta, promjenljive "najmanji_dosad" i "najveci_dosad" dobiće istu vrijednost kao i promjenljiva "tekuci_broj", tako da naredna dva uvjeta svakako neće biti tačna, odnosno neće se desiti ništa. Slijedi da neće biti nikakve štete od toga što će se oni testirati i u prvom prolasku kroz petlju.

Moguće je izvršiti i objedinjavanje uvjeta u manje nešto složenijih uvjeta. Recimo, uvjet "`i == 1`" može se objediniti s uvjetima "`tekuci_broj < najmanji_dosad`" i "`tekuci_broj > najveci_dosad`" u složene uvjete "`i == 1 || tekuci_broj < najmanji_dosad`" i "`i == 1 || tekuci_broj > najveci_dosad`" koristeći operator logičke disjunkcije "`||`". Time dobijamo sljedeće rješenje:

```
#include <iostream>

int main() {
  int n;
  std::cout << "Koliko zelite unijeti brojeva? ";
  std::cin >> n;
  std::cout << "Unesite brojeve:" << std::endl;
  int najmanji_dosad, najveci_dosad;
  for(int i = 1; i <= n; i++) {
    int tekuci_broj;
    std::cout << i << ". broj: ";
    std::cin >> tekuci_broj;
    if(i == 1 || tekuci_broj < najmanji_dosad) najmanji_dosad = tekuci_broj;
    if(i == 1 || tekuci_broj > najveci_dosad) najveci_dosad = tekuci_broj;
  }
  std::cout << "Najmanji medju unesenim brojevima je " << najmanji_dosad
    << ", a najveci medju unesenim brojevima je " << najveci_dosad << std::endl;
  return 0;
}
```

Ovo rješenje zasniva se na ključnoj ideji da se dodjele "`najmanji_dosad = tekuci_broj`" odnosno "`najveci_dosad = tekuci_broj`" izvrše ne samo onda kada su uvjeti "`tekuci_broj < najmanji_dosad`" odnosno "`tekuci_broj > najveci_dosad`" tačni, nego i u prvom prolasku kroz petlju (tj. kada je uvjet "`i == 1`" tačan), bez obzira na ostale uvjete, što će obezbijediti ispravnu početnu vrijednost za promjenljive "`najmanji_dosad`" i "`najveci_dosad`". Naime, uvjet oblika "x `||` y" je tačan kad god je poduvjet "x" tačan (u našem slučaju, to je pod uvjet "`i == 1`"), bez obzira na tačnost poduvjeta "y". Neko bi mogao prigovoriti da su u prvom prolasku kroz petlju promjenljive "`najmanji_dosad`" i "`najveci_dosad`" neinicijalizirane, što znači da su uvjeti "`tekuci_broj < najmanji_dosad`" odnosno "`tekuci_broj > najveci_dosad`" nedefinirani, odnosno ne zna se da li su tačni ili netačni. Međutim, kod operatora "`||`" postoji garancija da će se u uvjetima oblika "x `||` y" prvo testirati poduvjet "x", pa ukoliko se ispostavi da je on tačan, čitav će se uvjet smatrati tačnim, bez da se poduvjet "y" uopće pokuša testirati (to recimo znači da tada, ukoliko poduvjet "y" sadrži recimo poziv neke funkcije, ta funkcija uopće neće biti pozvana). Stoga se u prvom prolasku kroz petlju uopće neće ni testirati sporni poduvjeti koji sadrže neinicijalizirane promjenljve. Tek ukoliko je poduvjet "x" netačan, testira se i poduvjet "y", čija će tačnost odrediti tačnost čitavog uvjeta. Slično ponašanje postoji i kod operatora logičke konjunkcije "`&&`". Naime, u uvjetima oblika "x `&&` y", prvo se testira poduvjet "x", pa ukoliko se ispostavi da je on netačan, čitav će se uvjet smatrati netačnim, bez da se poduvjet "y" uopće pokuša testirati. Tek ukoliko je poduvjet "x" tačan, testira se i poduvjet "y", čija će tačnost odrediti tačnost čitavog uvjeta. Ovo specifično ponašanje operatora "`||`" i "`&&`" u složenim uvjetima često se naziva *kratkospojno izračunavanje* odnosno *izračunavanje kratkog spoja* (engl. *short-circuit evaluation*) i često omogućava brojne olakšice u programiranju.

U svim dosada prikazanim rješenjima, na direktan ili indirektan način prvi uneseni broj tretira se drugačije od ostalih, bilo što se on obrađuje izvan petlje, bilo što se unutar petlje testira da li se radi o prvom unesenom broju ili ne. Mnogo elegantnije rješenje bismo dobili kada bismo mogli sve unesene brojeve tretirati na isti način. To se može postići pogodnom inicijalizacijom promjenljivih "`najmanji_dosad`" i "`najveci_dosad`". Naime, kada bismo na početku promjenljivu "`najmanji_dosad`" inicijalizirali tako da na početku bude veća od svih brojeva koje planiramo unijeti, tada bi već prvi uneseni broj bio manji od njene početne vrijednosti, i ona bi dobila vrijednost prvog unesenog broja, kao i u svim dosadašnjim rješenjima (i kao što treba). Slično će se desiti ukoliko na početku promjenljivu "`najveci_dosad`" postavimo da bude manja od svih brojeva koje planiramo unijeti. Jedino je pitanje *kako da znamo opseg brojeva koji ćemo unositi*. Često se viđaju loša rješenja u kojima programer prosto *pretpostavi* da brojevi koje ćemo unositi nisu po modulu veći od neke vrijednosti, recimo 10000 (mada to nigdje nije rečeno u postavci zadatka), tako da promjenljive "`najmanji_dosad`" i "`najveci_dosad`" inicijalizira recimo na vrijednosti 10000 i -10000. Jasno je da takvo rješenje neće raditi ispravno ukoliko su svi uneseni brojevi veći od 10000 ili ukoliko su svi uneseni brojevi manji od -10000. Često se susreću i još gora rješenja u kojima neko za početnu vrijednost promjenljive "`najveci_dosad`" uzme nulu. Jasno je da takvo rješenje neće raditi korektno ukoliko su svi uneseni brojevi negativni, a nigdje nije rečeno da se to ne može desiti. Jedino što je u postavci zadatka rečeno je da će uneseni brojevi biti takvi da mogu stati u promjenljive tipa "`int`". Stoga je jedino korektno rješenje da promjenljive "`najmanji_dosad`" i "`najveci_dosad`" na početku inicijaliziramo na najveću odnosno najmanju vrijednost koja može stati u promjenljive tipa "`int`". Nažalost, koje su to tačne vrijednosti ovisi od konkretnog kompajlera. Na većini kompajlera to su

vrijednosti $2^{31}-1 = 2147483647$ i $-2^{31} = -2147483648$, ali te vrijednosti nisu garantirano. Sretna je okolnost što se u biblioteci sa zaglavljem "climits" nalaze konstante nazvane "INT_MAX" i "INT_MIN", koje predstavljaju najveću odnosno najmanju vrijednost koja na konkretnom kompajleru može stati u promjenljive tipa "**int**". Upotreba ovih konstanti, ne samo da smanjuje ovisnost od toga koji kompajler koristi, nego nam omogućava da ne moramo pamtiti koliko ove vrijednosti iznose, čak i ukoliko znamo njihove vrijednosti na konkretnom kompajleru koji koristimo). Pri tome, treba istaknuti da, zbog nekih tehničkih razloga, ove konstante nisu u imeniku "std", tako da bi navođenje prefiksa "std::" ispred njih bilo pogrešno i dovelo bi do prijave greške. To nam daje sljedeće, vrlo elegantno rješenje:

```
#include <iostream>
#include <climits>

int main() {
  int n;
  std::cout << "Koliko zelite unijeti brojeva? ";
  std::cin >> n;
  int najmanji_dosad = INT_MAX, najveci_dosad = INT_MIN;
  std::cout << "Unesite brojeve:" << std::endl;
  for(int i = 1; i <= n; i++) {
    int tekuci_broj;
    std::cout << i << ". broj: ";
    std::cin >> tekuci_broj;
    if(tekuci_broj < najmanji_dosad) najmanji_dosad = tekuci_broj;
    if(tekuci_broj > najveci_dosad) najveci_dosad = tekuci_broj;
  }
  std::cout << "Najmanji medju unesenim brojevima je " << najmanji_dosad
    << ", a najveci medju unesenim brojevima je " << najveci_dosad << std::endl;
  return 0;
}
```

Mada je ovo rješenje jako elegantno, njemu se još uvijek mogu prigovoriti neke stvari. Prvo, konstante "INT_MAX" i "INT_MIN" prilagođene su isključivo tipu podataka "**int**". Ukoliko bismo umjesto tipa "**int**" za unos brojeva koristili neki drugi cjelobrojni tip drugačijeg opsega mogućih vrijednosti, recimo tip "**unsigned long int**", morali bismo umjesto konstanti "INT_MAX" i "INT_MIN" koristiti neke druge (za ovaj primjer, to bi konkretno bile "ULONG_MAX" i "ULONG_MIN"). Još bi gora situacija bila ukoliko bismo umjesto cijelih brojeva unosili realne brojeve, recimo tipa "**double**". Nije iznenađenje da postoje konstante koje i za realne tipove određuju njihove dozvoljene opsege, ali je iznenađenje da se one ne nalaze u biblioteci sa zaglavljem "climit", nego u biblioteci sa zaglavljem "cfloat". Tako bismo, ukoliko želimo koristiti tip "**double**" umjesto tipa "**int**" za unos brojeva, prvo morali zamijeniti zaglavlje "climit" sa zaglavljem "cfloat". Nakon toga bismo trebali zamijeniti i konstante "INT_MAX" i "INT_MIN". Konkretno, umjesto konstante "INT_MAX" trebali bismo koristiti konstantu "DBL_MAX", što nije preveliko iznenađenje. Mnogi bi, po analogiji, pomislili da umjesto konstante "INT_MIN" treba koristiti konstantu "DBL_MIN". Nažalost, bez obzira što konstanta s takvim imenom zaista postoji, ona ne predstavlja najmanji broj koji se može zapisati u promjenljivu tipa "**double**", nego predstavlja *najmanji pozitivni broj koji se može zapisati u promenljivu tipa* "**double**", odnosno svi brojevi koji su po modulu manji od tog broja tretiraju se kao nule (to je neki po modulu vrlo mali broj, obično reda veličine oko 10^{-308}). Stoga, za ono što nam je potrebno, umjesto konstante "DBL_MIN" treba koristiti konstrukciju "-DBL_MAX", odnosno negiranu vrijednost konstante "DBL_MAX". Stvar je u tome što su, zbog nekih razloga u koje nećemo ulaziti, dozvoljeni opsezi promjenljivih realnih tipova uvijek simetrični (tj. ukoliko je maksimalna dozvoljena vrijednost M, minimalna dozvoljena vrijednost je $-M$), dok kod cjelobrojnih tipova najčešće postoji asimetrija između maksimalne i minimalne dozvoljene vrijednosti, što se može vidjeti ako pogledamo tipične vrijednosti konstanti "INT_MAX" i "INT_MIN" (koje iznose 2147483647 i −2147483648). Dakle, ukoliko bismo prethodni program željeli prepraviti da se umjesto cijelih brojeva sa tastature unose realni brojevi, trebali bismo promjenljive "najmanji_dosad", "najveci_dosad" i "tekuci_broj" deklarirati da budu tipa "**double**" umjesto "**int**", zaglavlje "climit" trebalo bi zamijeniti zaglavljem "cfloat", dok bi konstante "INT_MAX" i "INT_MIN" trebalo zamijeniti s "DBL_MAX" i "-DBL_MAX".

Pomenute konstante za određivanje opsega poput "INT_MAX", "DBL_MAX" itd. naslijeđene su iz jezika C. Međutim, na predavanjima je rečeno da jezik C++ nudi jednoobrazniji i fleksibilnihi način za saznavanje raznih informacija o pojedinim numeričkim tipovima podataka, uključujući informacije o maksimalnoj i minimalnoj dozvoljenoj vrijednosti. Za tu svrhu potrebno je u program uključiti biblioteku sa zaglavljem "limits" (bez "c" na početku). Sâm pristup ovim informacijama vrši se

pomoću konstrukcija za koje je karakteristično da sadrže frazu "`numeric_limits<tip>`" gdje je "*tip*" ime tipa za kojeg nam trebaju informacije. Konkretno, najveću i najmanju vrijednost koja može stati u promjenljivu tipa "*tip*" možemo dobiti pomoću konstrukcija "`std::numeric_limits<tip>::max()`" odnosno "`std::numeric_limits<tip>::lowest()`" (prefiks "`std::`" nije potreban ukoliko smo prethodno pomoću "**`using`**" deklaracije"uvezli" ime "`numeric_limits`" iz imenika "`std`", ili čak i cijeli imenik "`std`"). Tako dobijamo sljedeću verziju programa, u kojoj je, u slučaju da želimo raditi s nekim drugim tipom podataka a ne s tipom "**`int`**", dovoljno samo promijeniti oznaku tipa prilikom deklaracije promjenljivih "`najmanji_dosad`", "`najveci_dosad`" i "`tekuci_broj`" kao i oznaku tipa u frazi "`numeric_limits<int>`" u željeni tip:

```
#include <iostream>
#include <limits>

int main() {
  int n;
  std::cout << "Koliko zelite unijeti brojeva? ";
  std::cin >> n;
  int najmanji_dosad = std::numeric_limits<int>::max(),
    najveci_dosad = std::numeric_limits<int>::lowest();
  std::cout << "Unesite brojeve:" << std::endl;
  for(int i = 1; i <= 10; i++) {
    int tekuci_broj;
    std::cout << i << ". broj: ";
    std::cin >> tekuci_broj;
    if(tekuci_broj < najmanji_dosad) najmanji_dosad = tekuci_broj;
    if(tekuci_broj > najveci_dosad) najveci_dosad = tekuci_broj;
  }
  std::cout << "Najmanji medju unesenim brojevima je " << najmanji_dosad
    << ", a najveci medju unesenim brojevima je " << najveci_dosad << std::endl;
  return 0;
}
```

Ovo je vjerovatno najbolje i najfleksibilnije rješenje postavljenog problema, koje se može napisati samo sa stvarima koje su dosada rađene na kursu i kursevima koji su prethodili ovom kursu (uz upotrebu nekih stvari koje nisu još rađene, moguće je napisati i još bolja rješenja). Obratimo pažnju da smo za dohvatanje minimalne dozvoljene vrijednosti koristili funkciju "`lowest`", a ne "`min`". Naime, mada u ovom kontekstu postoji i funkcija "`min`" (koja se upotrebljava koristeći istu sintaksu kao i funkcije "`max`" i "`lowest`"), ona za realne tipove podataka ne daje ispravnu stvar. Mada za cjelobrojne tipove podataka ona radi isto kao i funkcija "`lowest`", za realne tipove podataka ona daje najmanju pozitivnu vrijednost koju je moguće predstaviti u navedenom tipu, poput konstante "`DBL_MIN`" o kojoj smo govorili ranije.

6. Napisati program koji traži da se s tastature unese visina trougla kao prirodan broj, a zatim iscrtava na ekranu trougao zadane visine sastavljen od zvjezdica. Na primjer, ukoliko se za željenu visinu unese 8, prikaz na ekranu treba da izgleda poput sljedećeg:

```
       *
      ***
     *****
    *******
   *********
  ***********
 *************
***************
```

Nije potrebno provjeravati ispravnost unosa, odnosno dozvoljeno je pretpostaviti da će uneseni podatak o visini biti ispravan prirodan broj, dovoljno mali da iscrtani trougao može stati na ekran.

Rješenje:

Ovo je još jedan lagan zadatak, ali koji se može riješiti na mnogo različitih načina, pri čemu se iz svakog od mogućeg načina rješavanja može mnogo naučiti, tako da će ovdje biti prikazano nekoliko rješenja. U svakom slučaju, jasno je da trougao trebamo iscrtavati red po red, u nekoj petlji. Stoga je potrebno razmotriti kako teče iscrtavanje jednog reda. U svakom redu prvo je potrebno ispisati određeni broj razmaka, zatim određeni broj zvjezdica, te preći u novi red. Postoji mnogo načina kako možemo ispisati određeni broj razmaka odnosno zvjezdica, a u prvom rješenju za tu svrhu

koristićemo petlje, u kojima ćemo jednostavno ispisivati po jedan razmak odnosno zvjezdicu, a petlja će se ponavljati onoliko puta koliko nam razmaka odnosno zvjezdica treba. U prvom pristupu, uvešćemo promjenljive nazvane "broj_razmaka" i "broj_zvjezdica", koje govore koliko razmaka odnosno zvjezdica treba ispisati u tekućem redu. Jasno je da, bez obzira na visinu trougla, za početni red broj zvjezdica je uvijek 1, dok je broj razmaka za 1 manji od visine trougla. Još jedino treba primijetiti da se, nakon iscrtavanja svakog reda, broj zvjezdica uvećava za 2, a broj razmaka umanjuje za 1. Iz ove analize slijedi rješenje poput sljedećeg:

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  int broj_zvjezdica = 1, broj_razmaka = visina - 1;
  for(int red = 1; red <= visina; red++) {
    for(int i = 1; i <= broj_razmaka; i++) std::cout << " ";
    for(int i = 1; i <= broj_zvjezdica; i++) std::cout << "*";
    std::cout << std::endl;
    broj_zvjezdica += 2; broj_razmaka--;
  }
  return 0;
}
```

Ovo rješenje radi, ali se može značajno poboljšati. Prvo je pitanje da li su nam uopće potrebne promjenljive "broj_razmaka" i "broj_zvjezdica". Ako pažljivo logički razmotrimo problem, možemo vidjeti da nam je, prilikom iscrtavanja r-tog reda po redu (pri čemu brojanje počinje od jedinice), potrebno ispisati $h-r$ razmaka (h je visina trougla), te $2r-1$ zvjezdicu. Na ovaj način, dolazimo do sljedećeg rješenja, u kojem se ne koriste promjenljive "broj_razmaka" i "broj_zvjezdica", nego se potrebni broj ispisivanja razmaka odnosno zvjezdica računa u skladu s provedenim razmatranjem (uvjet "i <= 2 * red - 1", zbog cjelobrojnosti promjenljivih, ekvivalentan je uvjetu "i < 2 * red":

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) {
    for(int i = 1; i <= visina - red; i++) std::cout << " ";
    for(int i = 1; i < 2 * red; i++) std::cout << "*";
    std::cout << std::endl;
  }
  return 0;
}
```

Petlju za ispis razmaka možemo izbjeći tako što ćemo pomoću manipulatora "setw" postaviti širinu ispisa na onoliku koliko želimo razmaka, a zatim pokušati ispisati prazan niz znakova (ništa između navodnika). To će dovesti do ispisa željenog broja razmaka, jer će se "prazno" raširiti na zadanu širinu, i popuniti razmacima. Treba voditi računa da sam manipulator "setw" ništa ne ispisuje, nego samo zadaje širinu za sljedeću stavku koja će biti ispisana (to je razlog zbog kojeg je potrebno poslati prazan niz znakova nakon manipulatora). Korištenje ovog manipulatora traži da u program uključimo i biblioteku sa zaglavljem "iomanip". To vodi ka sljedećem rješenju:

```
#include <iostream>
#include <iomanip>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) {
    std::cout << std::setw(visina - red) << "";
    for(int i = 1; i < 2 * red; i++) std::cout << "*";
    std::cout << std::endl;
  }
  return 0;
}
```

Možemo postupiti ovako. Umjesto da postavimo širinu ispisa na željeni broj razmaka i pošaljemo prazan tekst, možemo postaviti širinu ispisa za jedan veću od željenog broja razmaka bez ikakvog slanja ičega na izlazni tok. Nakon toga, kada naiđe petlja za ispis zvjezdica, prva zvjezdica koja se bude ispisivala biće, zbog postavljene širine ispisa, dopunjena ispred s tačno onoliko razmaka koliko treba. Kako postavljanje širine vrijedi samo za prvi ispis koji uslijedi, sve ostale zvjezdice biće ispisane bez ikakvih dodatnih razmaka ispred, kao što i treba da bude. S obzirom da nakon postavljanja širine ispisa nećemo ništa slati na tok u istoj naredbi, umjesto manipulatora "`setw`" za postavljanje širine možemo koristiti funkciju "`width`", pozvanu nad objektom toka. Stoga nam neće biti potrebno ni uključivanje biblioteke sa zaglavljem "`iomanip`". Time dobijamo sljedeće rješenje:

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) {
    std::cout.width(visina - red + 1);
    for(int i = 1; i < 2 * red; i++) std::cout << "*";
    std::cout << std::endl;
  }
  return 0;
}
```

Konačno, moguće je izbjeći i petlju za štampanje zvjezdica. Naime, kad dođe vrijeme za štampanje zvjezdica, možemo postaviti širinu ispisa da bude jednaka željenom broju zvjezdica, ali pomoću manipulatora "`setfill`" zadati da znak za ispunu novododanog prostora bude zvjezdica, a ne razmak. Naravno, nakon toga treba nešto konkretno poslati na izlazni tok (recimo prazan tekst) da bi do ispisa zaista došlo. Ima još samo jedan sitni detalj. S obzirom da znak za popunu koji se zadaje pomoću manipulatora "`setfill`" vrijedi sve dok se ne promijeni, moraćemo prije dijela u kojem treba ispisati razmake postaviti da znak za popunu bude razmak. U suprotnom, nakon što se znak za popunu postavi na zvjezdicu, tako bi ostalo do daljneg, i u svim narednim prolascima kroz petlju sve bi se popunjavalo zvjezdicama, uključujući i tamo gdje bi trebali biti razmaci. U skladu s tim, dolazimo do rješenja poput sljedećeg:

```
#include <iostream>
#include <iomanip>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++)
    std::cout << std::setfill(' ') << std::setw(visina - red) << ""
      << std::setfill('*') << std::setw(2 * red - 1) << "" << std::endl;
  return 0;
}
```

Ovo je vjerovatno najelegantnije rješenje koje se može napisati samo sa znanjem stečenim na prvom predavanju i predznanju s kurseva koji prethode ovom kursu. Nakon što se na Predavanju 3_b upoznamo s tipom "`string`", shvatićemo da je moguće napisati još kraća i elegantnija rješenja.

Na kraju je neophodno napomenuti nešto što nema direktne veze s ovim zadatkom, ali itekako ima s kvalitetnim programiranjem. Tiče se manipulatora "`setfill`", koji ima osobinu da postavljeni znak za popunu ostaje važeći sve dok ga ne promijenimo (isto vrijedi i za srodnu funkciju "`fill`", kao i za većinu drugih manipulatora i funkcija za formatiranje ispisa). Problem nastaje ukoliko upotrijebimo taj manipulator unutar nekog potprograma (funkcije). Sâmo po sebi, to nije problem sve dok tu funkciju koristimo samo unutar programa za koji je predviđena. Međutim, problemi nastaju ukoliko tu istu funkciju pokušamo upotrijebiti u nekom drugom programu, ili je damo nekome drugome da je upotrijebi u svom programu (uostalom, jedan od glavnih razloga zbog kojeg se funkcije uopće pišu je mogućnost *ponovna iskoristivost*, odnosno mogućnost da se ista funkcija upotrijebi na više mjesta, i eventualno, unutar više različitih programa). Tako, ukoliko u funkciji koja nešto radi (nebitno šta) upotrijebimo manipulator "`setfill`" da postavimo znak za ispunu na zvjezdicu, taj znak će ostati važeći i nakon što se funkcija završi. Vrlo je vjerovatno da onaj ko poziva funkciju neće biti svjestan tog pratećeg efekta (osim ukoliko je eksplicitno upozoren na to), odnosno poziv

te funkcije ostavlja neželjene posljedice. Drugim riječima, funkcija radi više od onoga što se od nje očekuje, jer osim onoga što bi trebala da radi, ona ostavlja zvjezdicu kao znak za popunu. Kao primjer, pretpostavimo da smo prethodni program htjeli da učinimo malo "modluarnijim", tako što ćemo napraviti posebnu funkciju "CrtajRed" koja iscrtava jedan red trougla, a koju ćemo u glavnom programu samo pozivati unutar petlje (ova funkcija će kao parametar primati redni broj reda, što je očigledno, ali će morati primati i visinu trougla, jer je neophodno znati visinu da bi se znalo koliko treba ispisati razmaka na početku reda):

```
#include <iostream>
#include <iomanip>

void CrtajRed(int red, int visina) {
  std::cout << std::setfill(' ') << std::setw(visina - red) << ""
    << std::setfill('*') << std::setw(2 * red - 1) << "" << std::endl;
}

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) CrtajRed(red, visina);
  return 0;
}
```

Nije uopće sporno da ovako napisan program radi ispravno. Međutim, funkcija "CrtajRed" radi više od onoga što tvrdi da radi (da crta jedan red trougla): ona do daljneg postavlja znak ispune na zvjezdicu. Ukoliko bi neko pokušao ovu funkciju da upotrijebi u nekom drugom kontekstu (nebitno od toga zašto bi to radio), recimo ukoliko bi napisao isječak kôda poput

```
CrtajRed(3, 10);
std::cout << std::setw(10) << 2 + 3;
```

vjerovatno bi bio iznenađen kada bi ispred broja 5 (rezultata računanja izraza 2+3) ugledao devet zvjezdica, a ne devet razmaka. Pouka je da nijedna dobro napisana funkcija ne bi trebala ostavljati iza sebe neočekivane posljedice koje mogu izmijeniti smisao programa koji slijedi iza njihovog poziva. Stoga bi funkcija "CrtajRed" trebala, prije završetka, da vrati znak za ispunu kakav je bio prije njenog poziva. Ne smijemo pretpostaviti da je to nužno bio razmak, s obzirom da je neko prije poziva ove funkcije možda postavio "tarabu" (tj. znak "#") kao znak za popunjavanje, i vjerovatno će očekivati da nakon poziva funkcije "CrtajRed" taj znak i dalje vrijedi, a ne da bude promijenjen na razmak. Kako ovo postići? Ukoliko funkciju "fill" pozovemo nad objektom toka bez parametara, ona će kao rezultat vratiti tekući (trenutno važeći) znak za popunu, koji možemo sačuvati u nekoj promjenljivoj tipa "**char**" (nazvaćemo je "stari_znak_za_popunu"). Na kraju, zapamćeni znak možemo iskoristiti da vratimo znak za popunu na prvobitnu vrijednost. Funkcija "fill" također omogućava da odjednom možemo preuzeti tekući znak za popunu i postaviti novi znak, čime se dobija nešto kraća izvedba. Stoga bi korektna izvedba funkcije "CrtajRed", koja ne ostavlja za sobom nikakve neočekivane posljedice, mogla izgledati recimo ovako:

```
void CrtajRed(int red, int visina) {
  char stari_znak_za_popunu = std::cout.fill(' ');
  std::cout << std::setw(visina - red) << "" << std::setfill('*')
    << std::setw(2 * red - 1) << "" << std::setfill(stari_znak_za_popunu) << std::endl;
}
```

Mnogi čitaoci ovu digresiju vezanu za "korektnu" izvedbu funkcije "CrtajRed" neće shvatiti ozbiljno, jer će njima biti važno samo da "program radi". Međutim, disciplinovano pisanje funkcija koje ne ostavljaju neočekivane posljedice nego rade samo tačno ono što se od njih očekuje od presudnog je značaja pri razvoju složenih programa, a pogotovo u timskom radu. Svaka funkcija treba da radi tačno ono što se od nje očekuje, ni manje ni više od toga. Funkcije čije pozivanje ima neočekivane posljedice koje se mogu nekada odraziti posve daleko od mjesta poziva, čest je uzrok bagova pri razvoju softvera. Situacija je slična kao kada poručujete hranu putem nekog od servisa za dostavu: kada naručite neko jelo putem servisa, vi od dostavljača očekujete tačno to: da Vam dostavi jelo, ni manje ni više. Ukoliko dostavljač zaista dostavi jelo, ali prilikom dostave udari u Vaš automobil koji je parkiran u blizini Vašeg ulaza i napravi veliku štetu na njemu, da li ćete reći da je dostavljač korektno obavio svoj posao, bez obzira što je on zapravo izvršio ono što ste od njega tražili (dostavio jelo)? Problem je upravo u tome što je izvršio i nešto štetno, što sigurno niste tražili…

7. Napisati program koji traži da se s tastature unese visina trougla kao prirodan broj, a zatim iscrtava na ekranu "šuplji" trougao zadane visine sastavljen od zvjezdica, na način da se zvjezdice nalaze samo duž ivica trougla, dok njegova unutrašnjost ostaje prazna. Na primjer, ukoliko se za željenu visinu unese 8, prikaz na ekranu treba da izgleda poput sljedećeg:

```
       *
      * *
     *   *
    *     *
   *       *
  *         *
 *           *
***************
```

Nije potrebno provjeravati ispravnost unosa, odnosno dozvoljeno je pretpostaviti da će uneseni podatak o visini biti ispravan prirodan broj, dovoljno mali da iscrtani trougao može stati na ekran.

Rješenje:

Ovaj zadatak je teži od prethodnog, i može se riješiti na još više različitih načina, pri čemu se iz svakog od mogućeg načina rješavanja ponovo može mnogo naučiti, tako da će ovdje biti prikazano nekoliko najzanimljivijih rješenja. Mada je i ovdje jasno da trougao trebamo iscrtavati red po red, otežavajuća okolnost je što se svi redovi ne tretiraju na isti način. Naime, nije teško vidjeti da se prvi i posljednji red trebaju tretirati drugačije: u svim redovima osim prvog i posljednjeg imamo tačno po dvije zvjezdice, dok u prvom redu imamo samo jednu zvjezdicu, a u posljednjem redu mnoštvo zvjezdica. Prvo ćemo kreirati rješenja u kojima ćemo posebno razmatrati prvi i posljednji red, a kasnije ćemo vidjeti kako je moguće dobiti jednoobrazniji tretman. Što se tiče prvog reda, nije teško vidjeti da prvo treba ispisati jedan manje razmak nego što je visina trougla, a zatim jednu zvjezdicu i preći u novi red. Za sve redove od drugog do pretposljednjeg, ispisujemo određeni broj razmaka, zatim jednu zvjezdicu, zatim ponovo određeni broj razmaka i nakon toga jednu zvjezdicu, te na kraju prelazimo u novi red. U prvim izvedbama, razmake ćemo ispisivati ispisujući jedan po jedan razmak unutar petlje. Također ćemo, u prvoj izvedbi, uvesti dvije promjenljive nazvane "`broj_razmaka_1`" i "`broj_razmaka_2`", koje govore koliko razmaka treba ispisati prije prve zvjezdice, odnosno nakon prve, a prije druge zvjezdice. Kako počinjemo od drugog reda, početni broj razmaka prije prve zvjezdice je za 2 manji od visine trougla, dok je početni broj razmaka nakon prve zvjezdice uvijek 1, bez obzira na visinu trougla. Nakon iscrtavanja svakog reda, broj razmaka prije prve zvjezdice umanjuje se za 1, a broj razmaka nakon prve zvjezdice uvećava se za 2. Konačno, što se tiče posljednjeg reda, u njemu treba iscrtati onoliko zvjezdica koliko bi bilo razmaka nakon prve zvjezdice kada bi se taj red tretirao identično kao i ostali, samo još dodatno uvećano za 2 (zbog početne i krajnje zvjezdice). Ovo vodi ka sljedećem rješenju (u kojem je na mnogim mjestima, radi praktičnosti, prelazak u novi red ostvaren ubacivanjem sekvence "`\n`" u tekst koji se ispisuje):

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int i = 1; i < visina; i++) std::cout << " ";
  std::cout << "*\n";
  int broj_razmaka_1 = visina - 2, broj_razmaka_2 = 1;
  for(int red = 2; red < visina; red++) {
    for(int i = 1; i <= broj_razmaka_1; i++) std::cout << " ";
    std::cout << "*";
    for(int i = 1; i <= broj_razmaka_2; i++) std::cout << " ";
    std::cout << "*\n";
    broj_razmaka_1--; broj_razmaka_2 += 2;
  }
  for(int i = 1; i <= broj_razmaka_2 + 2; i++) std::cout << "*";
  std::cout << std::endl;
  return 0;
}
```

Prvi korak ka boljem rješenju je eliminirati potrebu za promjenljivim "`broj_razmaka_1`" odnosno "`broj_razmaka_2`". Ako pažljivo logički razmotrimo problem, možemo vidjeti da nam je, za sve

redove osim prvog i posljednjeg, prilikom iscrtavanja r-tog reda po redu (pri čemu ovdje brojanje počinje od dvojke), potrebno ispisati $h-r$ razmaka prije prve zvjezdice (h je visina trougla), te $2r-3$ razmaka nakon prve zvjezdice. Što se tiče posljednjeg reda, kako bi u njemu trebalo biti $2h-3$ razmaka između prve i druge zvjezdice kada bi se on tretirao identično kao i ostali redovi, zbog dodatne dvije zvjezdice to će biti ukupno $2h-1$ zvjezdica. Na ovaj način, dolazimo do sljedećeg rješenja, u kojem se ne koriste promjenljive "broj_razmaka_1" i "broj_razmaka_2", nego se potrebni broj ispisivanja razmaka odnosno zvjezdica računa u skladu s provedenim razmatranjem:

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int i = 1; i < visina; i++) std::cout << " ";
  std::cout << "*\n";
  for(int red = 2; red < visina; red++) {
    for(int i = 1; i <= visina - red; i++) std::cout << " ";
    std::cout << "*";
    for(int i = 1; i <= 2 * red - 3; i++) std::cout << " ";
    std::cout << "*\n";
  }
  for(int i = 1; i < 2 * visina; i++) std::cout << "*";
  std::cout << std::endl;
  return 0;
}
```

Sljedeći korak je eliminirati petlje za ispis razmaka. To možemo lako uraditi tako što ćemo prije ispisa zvjezdice (bilo prve, bilo druge) u svim redovima osim posljednjeg, postaviti širinu ispisa pomoću manipulatora "setw" za 1 veću nego što je željeni broj razmaka, nakon čega će ispis tačno jedne zvjezdice nakon toga dovesti do popunjavanja ispisa s tačno onoliko razmaka koliko treba. Pri tome, ovaj put neće biti dobro stavljati sekvencu "\n" u tekst koji se ispisuje radi prelaska u novi red, zbog toga što će ubacivanje te sekvence promijeniti dužinu teksta i poremetiti računanje koliko razmaka treba dodati da se postigne željena širina. Zbog toga ćemo prelazak u novi red vršiti slanjem manipulatora "endl". Slično se možemo osloboditi i posljednje petlje za ispis zvjezdica, tako što ćemo pomoću manipulatora "setfill" postaviti znak za popunu na zvjezdicu. To daje rješenje poput sljedećeg:

```
#include <iostream>
#include <iomanip>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  std::cout << std::setw(visina) << "*" << std::endl;
  for(int red = 2; red < visina; red++)
    std::cout << std::setw(visina - red + 1) << "*"
      << std::setw(2 * red - 2) << "*" << std::endl;
  std::cout << std::setfill('*') << std::setw(2 * visina - 1) << "" << std::endl;
  return 0;
}
```

Ovo rješenje je prilično elegantno. Jedina eventualna neelegancija u ovom rješenju uzrokovana je činjenicom da se prvi i posljednji red procesiraju drugačije nego ostali redovi, izvan petlje. Sada ćemo razmotriti da li je to moguće donekle unificirati. Krenućemo od verzije rješenja koje ne koristi manipulatore. Ako razmotrimo po ćemu se prvi red razlikuje od ostalih, vidjećemo da u njemu nakon ispisa prve zvjezdice odmah prelazi u novi red, dok se u svim ostalim redovima osim posljednjeg nakon toga ispisuje još jedna serija razmaka, još jedna zvjezdica, i tek onda novi red. Slijedi da bismo pomoću "if" naredbe mogli izvesti da se dodatne radnje, koje trebaju u redovima od drugog do pretposljednjeg, jednostavno ne izvedu ukoliko se radi o prvom redu. Isto tako, posljednji red se od ostalih redova osim prvog razlikuje jedino po tome što se između prve i krajnje zvjezdice ispisuju također zvjezdice, a ne razmaci. Slijedi da pomoću "if" naredbe možemo testirati da li se radi o posljednjem redu ili ne, te ako se radi o posljednjem redu ispisujemo zvjezdice u petlji, a u suprotnom ispisujemo razmake. To vodi ka sljedećem rješenju, u kojem se procesiranje svih redova izvodi unutar jedinstvene petlje:

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) {
    for(int i = 1; i <= visina - red; i++) std::cout << " ";
    if(red != 1) {
      std::cout << "*";
      for(int i = 1; i <= 2 * red - 3; i++)
        if(red == visina) std::cout << "*";
       else std::cout << " ";
    }
    std::cout << "*\n";
  }
  return 0;
}
```

Nedostatak ovog rješenja je preveliko oslanjanje na "`if`" naredbu. Mada su "`if`" naredbe ponekad neizbježne, često se kaže da je kvalitet nekog rješenja obrnuto proporcionalan broju "`if`" naredbi koje se u njemu javljaju, jer prevelika upotreba "`if`" naredbi ukazuje na to da rješavač problema nije sagledao opću sliku, nego pokušava problem riješiti slučaj po slučaj, čak i u slučaju kada se sve moglo riješiti jednoobrazno. Nažalost, ovdje nema previše mogućnosti za izbjegavanje "`if`" naredbi. Ipak, naredba "`if`" unutar unutrašnje petlje za ispis razmaka ili zvjezdica može se izbjeći uz pomoć ternarnog operatora "`?:`". Podsjetimo se da izraz oblika "x `?` y `:` z" ima vrijednost "y" ukoliko je "x" tačan (odnosno, različit od nule), a inače ima vrijednost "z". Ovo možemo iskoristiti da formiramo izraz koji će predstavljati zvjezdicu ukoliko je razmatrani red posljednji, a inače će predstavljati razmak. Takav izraz glasi "`red == visina ? "*" : " "`". Pri tome, kako operator "`?:`" ima veoma nizak prioritet (niži prioritet imaju jedino operatori dodjele poput "`=`", "`+=`", itd. te zarez-operator), kad god se ovaj operator koristi unutar nekog složenijeg izraza, potrebno je konstrkuciju oblika "x `?` y `:` z" staviti unutar zagrada. To daje sljedeće, neznatno skraćeno rješenje:

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) {
    for(int i = 1; i <= visina - red; i++) std::cout << " ";
    if(red != 1) {
      std::cout << "*";
      for(int i = 1; i <= 2 * red - 3; i++) std::cout << (red == visina ? "*" : " ");
    }
    std::cout << "*\n";
  }
  return 0;
}
```

Prve unutrašnje petlje, koja ispisuje seriju razmaka, možemo se osloboditi na sličan način kao i ranije, koristeći funkciju "`width`", ili manipulator "`setw`". Time dobijamo sljedeće rješenje:

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) {
    std::cout.width(visina - red + 1);
    if(red != 1) {
      std::cout << "*";
      for(int i = 1; i <= 2 * red - 3; i++) std::cout << (red == visina ? "*" : " ");
    }
    std::cout << "*" << std::endl;
  }
  return 0;
}
```

Preostalu unutrašnju petlju možemo također eliminirati korištenjem "`setw`" manipulatora, ali ćemo ovaj put koristiti i manipulator "`setfill`", pri čemu ćemo, ovisno od toga da li se radi o posljednjem redu ili ne, znak za popunu praznog prostora postaviti na zvjezdicu ukoliko se radi o posljednjem redu, a na razmak u suprotnom (ovdje ćemo ponovo iskoristiti operator "`?:`"). Treba samo paziti da manipulator "`setfill`" očekuje *znak* (između apostrofa), a ne tekst između navodnika, tako da treba voditi računa i o tome (inače ćemo dobiti prijavu greške od strane kompajlera). Na taj način dobijamo sljedeće optimizirano rješenje:

```
#include <iostream>
#include <iomanip>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) {
    std::cout.width(visina - red + 1);
    if(red != 1)
      std::cout << "*" << std::setw(2 * red - 3)
        << std::setfill(red != visina ? ' ' : '*') << "";
    std::cout << "*" << std::endl;
  }
  return 0;
}
```

Da li je bolje ovo rješenje, ili ranije napisano optimizirano rješenje u kojem se prvi i posljednji red tretiraju posebno, pretežno je stvar ukusa. Činjenica je da je takvo rješenje neznatno kraće i ne sadrži nijednu "`if`" naredbu, ali je prednost posljednjeg napisanog rješenja veća jednoobraznost, jer se svi redovi procesiraju unutar jedinstvene petlje.

Čisti matematičari će možda prigovoriti da nijedno od predloženih rješenja nije "elegantno". Postoji velika šansa da bi čisti matematičari mogli rezonovati ovako. Možemo kreirati dvije ugniježdene petlje, pri čemu vanjska petlja ide red po red, a unutrašnja petlja ide znak po znak unutar tekućeg reda (tako da indeks ove petlje odgovara rednom broju kolone u kojoj se trenutno nalazimo). Zatim možemo formirati matematički uvjet koji će, zavisno od rednog broja reda r i rednog broja kolone k, donijeti odluku da li na tom mjestu treba da bude zvjezdica ili znak. Pogledajmo kako bi taj uvjet trebao da izgleda. Nije teško vidjeti da za sve zvjezdice duž lijeve ivice trougla vrijedi jednakost $k = h - r + 1$, pri čemu je h visina trougla. Isto tako, za sve zvjezdice duž desne ivice trougla vrijedi jednakost $k = h + r - 1$. Konačno, za donju ivicu trougla vrijedi $r = h$, neovisno od k. Traženi uvjet stoga glasi $k = h - r + 1 \vee k = h + r - 1 \vee r = h$. Ostaje još samo da vidimo do koje granice treba ići unutrašnja petlja. Sigurna granica je širina trougla, koja iznosi $2h - 1$ (broj zvjezdica u posljednjem redu). Međutim, nema nikakve potrebe da idemo dalje od pozicije posljednje zvjezdice u redu, koja iznosi $h + r - 1$, s obzirom da iza nje slijede samo razmaci. Na taj način dolazimo do sljedećeg rješenja, koje je s aspekta efikasnosti lošije od optimiziranih rješenja koja koriste manipulatore (zbog stalnog testiranja prilično složenog uvjeta i unutrašnje petlje, za koju smo vidjeli da uopće nije bila nužna), ali je vjerovatno najelegantnije s matematičkog načina posmatranja:

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) {
    for(int kolona = 1; kolona < 2 * visina; kolona++)
      if(kolona == visina - red + 1 || kolona == visina + red - 1  || red == visina)
        std::cout << "*";
      else std::cout << " ";
    std::cout << std::endl;
  }
  return 0;
}
```

Ovdje također možemo izvesti malu optimizaciju, ukoliko umjesto "`if`" naredbe upotrijebimo ternarni "`?:`" operator. To daje sljedeće rješenje:

```
#include <iostream>

int main() {
  int visina;
  std::cout << "Unesite visinu: ";
  std::cin >> visina;
  for(int red = 1; red <= visina; red++) {
    for(int kolona = 1; kolona < 2 * visina; kolona++)
      std::cout << ((kolona == visina + 1 - red || kolona == visina - 1 + red
        || red == visina) ? "*" : " ");
    std::cout << std::endl;
  }
  return 0;
}
```

Strogo rečeno, unutrašnji par zagrada prije znaka "?" oko složenog uvjeta i nije bio potreban, s obzirom da svi operatori upotrijebljeni u tom uvjetu imaju veći prioritet od operatora "?:", ali ih je dobro staviti, jer se time povećava čitljivost.

Koje će Vam se od ponuđenih rješenja najviše svidjeti, pretežno zavisi od Vaših sklonosti i načina razmišljanja (različiti profili studenata preferiraju različita rješenja). Međutim, ključno pitanje je da li je zaista bilo vrijedno truda razmatrati toliko mnogo različitih rješenja? Vjerovatno je najbolji odgovor na to pitanje dao jedan od najvećih metodičara nastave matematike koji su ikada živjeli, mađarsko-američki matematičar George Pólya. Njegova čuvena izjava je da je daleko korisnije jedan matematički zadatak riješiti na 10 suštinski različitih načina, nego riješiti 10 posve različitih zadataka, ali na suštinski isti način. U programiranju, to vrijedi još više nego u matematici.