Zadaci za Laboratorijsku vježbu 5

NAPOMENA: Studenti bi trebali da razmisle o zadacima koji će se raditi na laboratorijskoj vježbi prije nego što dođu na vježbu, tako da već u startu imaju osnovne ideje kako riješiti zadatke. U suprotnom, rad na vježbi neće biti produktivan. Zadatke koje studenti ne stignu uraditi za vrijeme vježbe, trebali bi ih samostalno uraditi kod kuće.

1. Napišite program koji od korisnika traži da sa tastature unese rečenicu, ne dužu od 1000 znakova, a koji će zatim ispisati unesenu rečenicu *bez prve riječi* te rečenice, pri čemu ispis započinje od prvog znaka druge riječi (u slučaju da rečenica ima samo jednu riječ, ili da je "prazna" bez riječi, ne ispisuje se ništa). Pod riječi se ovdje smatra svaki niz znakova koji nisu razmaci a koji je s obje strane omeđen razmacima, osim na početku ili kraju kada ne mora biti razmaka lijevo odnodno desno od riječi. Vodite računa da može biti više razmaka između dvije riječi, kao i razmaka na početku ili kraju rečenice. Unesena rečenica se smješta u klasični niz znakova (dakle, ne u promjenljivu tipa "`string`"). Za realizaciju zadatka koristiti isključivo pokazivačku aritmetiku. Nije dozvoljena upotreba funkcija iz biblioteka "`cstring`" niti "`string`", kao ni upotreba indeksiranja (što uključuje i njegovu trivijalnu simulaciju koja podrazumijeva pisanje "`*(a + n)`" umjesto "`a[n]`"). Dijalog između korisnika i programa treba izgledati ovako:

```
Unesite recenicu:    Na   vrh   brda   vrba  mrda...
Recenica bez prve rijeci glasi: vrh   brda   vrba  mrda...
```

2. Napišite generičku funkciju "`RazmijeniBlokove`" koja prihvata iste parametre i obavlja isti zadatak kao funkcija "`swap_ranges`" iz biblioteke "`algorithm`". Funkcija treba koristiti potpunu dedukciju (tako da će da radi i sa pokazivačima i sa iteratorima) i treba je realizirati isključivo korištenjem pokazivačke aritmetike, bez korištenja indeksiranja i njegove trivijalne simulacije (tj. pisanja "`*(p + i)`" umjesto "`p[i]`"). Napišite i kratki testni program u kojem ćete testirati napisanu funkciju na jednom fiksnom paru nizova cijelih brojeva i na jednom fiksnom paru stringova (iste dužine).

3. Koristeći odgovarajuće funkcije iz biblioteke "`algorithm`", napišite program koji će za niz cijelih brojeva unesenih sa tastature (broj elemenata niza se prethodno također unosi sa tastature, a neće biti veći od 1000) ispisati:
   - najveći element niza;
   - koliko puta se u nizu pojavljuje najmanji element;
   - koliko u nizu cijelih brojeva unesenih sa tastature ima brojeva koji su potpuni kvadrati (tj. brojevi poput 1, 4, 9, 16, 25, 36, itd.);
   - element sa najmanjim brojem cifara (ili prvi od njih, ukoliko ima više elemenata koji imaju isti najmanji broj cifara).

   Nakon toga, program treba prepisati u drugi niz sve elemente koji nisu trocifreni, i ispisati elemente tako formiranog niza. Dijalog između korisnika i programa treba da izgleda poput sljedećeg (zanemarite probleme sa bosanskom gramatikom koji mogu dovesti do toga da se svi elementi ispisanih rečenica neće lijepo slagati po broju i padežu, što je vidljivo i u datom primjeru):

```
Unesite broj elemenata (max. 1000): 10
Unesite elemente: 1849 374 3327 -112 12544 -27 -112 998001 -112 49
Najveci element niza je 998001
Najmanji element niza se pojavljuje 3 puta u nizu
U nizu ima 4 brojeva koji su potpuni kvadrati
Prvi element sa najmanjim brojem cifara je -27
Elementi koii nisu trocifreni su: 1849 3327 12544 -27 998001 49
```

   Napomena: U programu nije uopće dozvoljeno koristiti petlje, već sve manipulacije treba ostvariti isključivo pozivima funkcija iz biblioteke "`algorithm`". Tamo gdje je potrebno da koristite funkcije kao parametre drugih funkcija, definirajte imenovane funkcije čije ćete ime slati kao parametar. Ako ne znate kako bez petlji dobiti broj cifara, prisjetiste se šta (ni)ste dosad naučili o logaritmima.

4. Izmijenite prethodni program tako što ćete umjesto niza koristiti dek (pri tome uklonite i ograničenje na max. 1000 elementata kao i dio "(max. 100)" u prvoj rečenici dijaloga) i što ćete umjesto imenovanih funkcija kao parametre koje šaljete funkcijama iz biblioteke "`algorithm`" koristiti anonimne (lambda) funkcije. Prilagodite dijalog između korisnika i programa tako da se umjesto varijacija riječi "niz" javljaju varijacije riječi "dek".

5. 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. Napisanu funkciju testirajte na primjerima integrala funkcije sin x na intervalu $(0, \pi)$, zatim funkcije x^3 na intervalu $(0, 10)$, i funkcije $1/x$ na intervalu $(1, 2)$. Testiranje izvršite za različite vrijednosti n i uporedite rezultate sa tačnim rezultatima. Utvrdite kolike su vrijednosti za n bile potrebne da se dobije rezultat tačan na 5 decimala za sva tri primjera.

```
Unesite broj podintervala: 70
Za taj broj podintervala priblizne vrijednosti integrala iznose:
- Za funkciju sin x na intervalu (0,pi): 1.99966
- Za funkciju x^3 na intervalu (0,10): 2500.51
- Za funkciju 1/x na intervalu (1,2): 0.69316
```

6. Napišite program koji će tražiti da se sa tastature unese broj elemenata nekog vektora, a zatim i sami elementi tog vektora. Nakon toga, program treba da sortira elemente tog vektora u rastući poredak *po sumi cifara* (tj. brojevi sa većom sumom cifara treba da dođu iza elemenata sa manjom sumom cifara), i da ispiše tako sortirani niz. Ukoliko dva broja imaju istu sumu cifara, manji broj treba da dođe prije većeg broja. Konačno, nakon obavljenog sortiranja, program treba da korisnika pita za neki broj n i da ispiše na kojoj se poziciji taj broj nalazi u nizu nakon obavljenog sortiranja, ili činjenicu da se taj broj uopće ne nalazi u nizu, korištenjem binarne pretrage. Dijalog između programa i korisnika trebao bi izgledati poput sljedećeg:

```
Unesite broj elemenata: 10
Unesite elemente: 25 -174 2316 1111 99 0 -345 614 -1111 22222
Niz sortiran po sumi cifara glasi: 0 -1111 1111 25 22222 614 -345 -174 2316 99
Unesite broj koji trazite: 2316
Trazeni broj nalazi se na poziciji 8
```

U slučaju da traženog broja nema u nizu, posljednja rečenica u dijalogu treba da glasi "Trazeni broj ne nalazi se u nizu!". U programu se nigdje ne smiju koristiti petlje, nego samo odgovarajuće funkcije iz biblioteke "`algorithm`" (za računanje sume cifara, možete se poslužiti rekurzijom).

7. Napišite generičku funkciju "`SumaBloka`", koja slično kao funkcije iz biblioteke "`algorithm`" prima kao parametre dva pokazivača ili iteratora koji omeđuju neki blok (prvi pokazuje na početak bloka, a drugi tačno iza kraja bloka). Funkcija treba da kao rezultat vrati sumu svih elemenata u bloku, uz pretpostavku da su elementi bloka takvi da se mogu sabirati. U slučaju da je blok prazan, treba baciti izuzetak tipa "`range_error`" uz prateći tekst "Blok je prazan". Testirajte funkciju na jednom deku realnih brojeva čiji se elementi unose sa tastature.

Napomena 1: Funkciju koja radi tačno ovako kako je opisano nemoguće je napisati bez C++11 inovacija. Slična funkcija, pod imenom "`accumulate`", postoji u biblioteci "`numeric`", ali ona zahtijeva i jedan dodatni parametar koji služi da se prevaziđu problemi koji se nisu mogli riješiti do uvođenja standarda C++11.

Napomena 2: Vjerovatno će Vam trebati operator "**`decltype`**". Međutim, vodite računa da ovaj operator vraća *tip reference* ukoliko mu je parametar *l*-vrijednost, tako da će konstrukcija poput "**`decltype`**`(*p)`" vratiti tip reference, jer je izraz "`*p`", koji je zadan kao parametar operatora "**`decltype`**", *l*-vrijednost. Tip reference u ovom slučaju neće biti pogodan, tako da je potrebno nekako prevazići ovaj problem. Univerzalan (ali nažalost prilično rogobatan) način za rješavanje ovog problema je upotreba metafunkcije "`remove_reference`". Međutim, u ovom konkretnom primjeru moguće je i znatno jednostavnije rješenje. Naime, mada će "**`decltype`**`(*p)`" vratiti tip reference, "**`decltype`**`(*p + *p)`" *neće vratiti tip reference*, jer izraz "`*p + *p`" *nije l-vrijednost*. Pored toga, izraz "`*p + *p`" je *garantirano legalan*, jer smo pošli od pretpostavke da su elementi bloka takvi da se mogu sabirati (što ne bi moralo biti bez ove pretpostavke).