Zadaci za Laboratorijsku vježbu 6

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 funkciju "`GenerirajStepeneDvojke`" sa jednim parametrom n, koja dinamički alocira niz od n brojeva, ali čiji tačan tip nije specificiran nego se zadaje naknadno, npr. speficikacijom poput "`GenerirajStepeneDvojke<unsigned long int>`". U slučaju da je $n \leq 0$, funkcija treba da baci izuzetak tipa "`domain_error`" uz prateći tekst "Broj elemenata mora biti pozitivan". a u slučaju da alokacija ne uspije, funkcija treba da baci izuzetak tipa "`runtime_error`" uz prateći tekst "Alokacija nije uspjela". Nakon alokacije, funkcija treba popuniti alocirani niz sa prvih n stepena broja 2 (stepeni dvojke su redom 1, 2, 4, 8, 16 itd.), pri čemu treba baciti izuzetak tipa "`overflow_error`" sa pratećim tekstom "Prekoracen dozvoljeni opseg" u slučaju da tom prilikom dođe do prekoračenja opsega tipa elemenata niza. Kao rezultat, funkcija treba da vrati pokazivač na prvi element tako alociranog niza. Napisanu funkciju iskoristite u testnom programu u kojem se sa tastature unosi broj n, zatim poziva napisana funkcija za kreiranje traženog niza, koji se potom ispisuje na ekran, i na kraju se vrši oslobađanje prostora koji je zauzeo niz. U testnom programu tip elemenata niza odaberite tako da program radi korektno za što je god moguće veće n, a da pri tome svi generirani stepeni dvojke budu prikazani potpuno egzaktno (odnosno bez ikakvog zaokruživanja). Također, u testnom programu treba predvidjeti hvatanje svih izuzetaka koji bi funkcija eventualno mogla baciti. Dijalozi između korisnika i programa trebali bi izgledati poput sljedećih:

```
Koliko zelite elemenata: 10
1 2 4 8 16 32 64 128 256 512
```

```
Koliko zelite elemenata: 100
Izuzetak: Prekoracen dozvoljeni opseg
```

2. Napišite generičku funkciju "`KreirajIzvrnutiNiz`" sa dva parametra koji predstavljaju pokazivače ili iteratore koji omeđuju neki blok elemenata (prvi parametar pokazuje na početak, a drugi tačno iza kraja bloka). Funkcija prvo treba da dinamički kreira niz čiji su elementi istog tipa kao i elementi razmatranog bloka i koji sadrži isti broj elemenata kao i razmatrani. U slučaju da alokacija ne uspije, funkcija treba baciti izuzetak tipa "`bad_alloc`", što je uobičajeno ponašanje u slučajevima kada alokacija ne uspijeva. Funkcija zatim treba prepisati sve elemente bloka u kreirani niz u obrnutom poretku (tj. prvi element bloka treba da postane posljednji element niza itd.) i vratiti pokazivač na prvi element tako kreiranog niza kao rezultat. Nad pokazivačima odnosno iteratorima koji se prosljeđuju kao parametri funkcije nije dozvoljeno koristiti niti jednu drugu operaciju osim dodjele ("`=`"), dereferenciranja ("`*`"), poređenja na jednakost i različitost ("`==`" i "`!=`"), te pomjeranja unaprijed ("`++`"). Razlog za ova ograničenja je da se funkcija učini univerzalnijom tako da može raditi i sa kontejnerima čiji iteratori nisu jednako moćni kao i pokazivači te podržavaju isključivo navedene operacije (kasnije ćemo vidjeti da ima i takvih kontejnera). Također, nije dozvoljeno ni koristiti funkcije iz biblioteke "`algorithm`", s obzirom da je cilj zadatka upravo da se one simuliraju i to uz posve ograničeni skup dozvoljenih operacija. Napišite i mali testni program u kojem ćete demonstrirati kako se upotrebljava napisana funkcija na vektoru realnih brojeva čiji se elementi unose sa tastature. Dijalog između programa i korisnika treba da izgleda poput sljedećeg:

```
Koliko zelite elemenata: 10
Unesite elemente: 2.5 -7 3.12 6 4 0 -1.111 4 2 7.39
Kreirani niz: 7.39 2 4 -1.111 0 4 6 3.12 -7 2.5
```

U slučaju nedostatka memorije, program treba ispisati tekst "Nedovoljno memorije!", bez obzira da li je do problema došlo prilikom kreiranja početnog vektora koji sadrži ulazne podatke, ili pri pokušaju dinamičke alokacije niza. Recimo, u prvom od ta dva navedena slučaja, dijalog između korisnika i programa mogao bi izgledati ovako:

```
Koliko zelite elemenata: 2000000000
Nedovoljno memorije!
```

3. Napišite funkciju "`KreirajTrougao`" (ili "`KreirajTrokut`", ovisno od Vaših jezičkih preferencija) sa jednim cjelobrojnim parametrom n koja vrši dinamičku alokaciju "grbave matrice" sa n redova u kojoj prvi red sadrži samo element 1, drugi red elemente 2, 1 i 2, treći red elemente 3, 2, 1, 2 i 3, četvrti red elemente 4, 3, 2, 1, 2, 3 i 4, itd. (elementi su tipa "`int`"). Kao rezultat, funkcija vraća dvostruki pokazivač preko kojeg se može pristupiti elementima tako kreirane grbave matrice. Ukoliko je n negativan ili 0, funkcija treba baciti izuzetak tipa "`domain_error`", uz prateći tekst "Broj redova mora biti pozitivan". U slučaju da alokacija ne uspije, funkcija treba baciti izuzetak tipa "`bad_alloc`", pazeći pri tome da ne dođe do curenja memorije ni u kakvim okolnostima. Za alociranje koristite postupak fragmentirane alokacije. Napisanu funkciju iskoristite u isječku programa koji traži da se sa tastature unese broj n, zatim kreira "grbavu matricu" traženih svojstava sa n redova, te na kraju ispisuje njene elemente red po red (uz jedan razmak između elemenata istog reda) nakon čega oslobađa prostor koji je zauzela ta matrica. Predvidite i hvatanje svih izuzetaka koji bi funkcija eventualno mogla baciti. U slučaju bacanja izuzetka tipa "`bad_alloc`", tekst koji se prikazuje na ekranu treba glasiti "Izuzetak: Nedovoljno memorije!". Dijalozi između korisnika i programa trebaju izgledati poput sljedećih:

```
Koliko zelite redova: 5
1
2 1 2
3 2 1 2 3
4 3 2 1 2 3 4
5 4 3 2 1 2 3 4 5
```

```
Koliko zelite redova: -3
Izuzetak: Broj redova mora biti pozitivan
```

```
Koliko zelite redova: 1000000000
Izuzetak: Nedovoljno memorije!
```

4. Ponovite prethodni zadatak, ali uz korištenje kontinualne umjesto fragmentirane alokacije.

5. Napišite generičku funkciju "`KreirajDinamickuKopiju2D`" koja kao svoj parametar prima neku dvodimenzionalnu strukturu nalik matrici, ali pri čemu broj elemenata u svakom redu ne mora nužno biti isti. Pretpostavlja se da je ta struktura predstavljena kao neki kontejnerski tip čiji su elementi ponovo nekog kontejnerskog tipa, ne nužno istog. Recimo, to može biti vektor vektōrâ, vektor dekōvâ, dek vektōrâ, dek dekōvâ, itd. Jedino ograničenje koje se postavlja na te kontejnerske tipove je da se mogu indeksirati i da podržavaju funkciju "`size`". Elementi "unutrašnjeg" kontejnerskog tipa mogu također biti proizvoljnog tipa. Funkcija prvo treba da dinamički alocira prostor za dvodimenzionalnu strukturu identičnog oblika kao i parametar, zatim da u nju prepiše elemente dvodimenzionalne strukture predstavljene parametrom i, konačno, da kao rezultat vrati dvojni pokazivač preko kojeg se može izvršiti pristup elementima ove strukture. Za alociranje koristite metod fragmentirane alokacije. U slučaju da dođe do problema sa alokacijom memorije, funkcija treba baciti izuzetak (tipa "`bad_alloc`"). Pri tome, ni u kom slučaju ne smije doći do curenja memorije. Napisanu funkciju testirajte u testnom programu koji sa tastature unosi elemente matrice formata $n \times n$ organizirane kao vektor dekōvâ cijelih brojeva (n se prethodno unosi sa tastature), a nakon toga poziva napisanu funkciju sa ciljem kreiranja odgovarajuće dinamičke matrice i, konačno, ispisuje elemente tako kreirane dinamičke matrice na ekran (svaki red matrice u posebnom redu, uz razmak između elemenata unutar jednog reda) i oslobađa zauzetu memoriju. U testnom programu predvidite i eventualno hvatanje bačenih izuzetaka. Dijalog između programa i korisnika treba izgledati poput sljedećeg (uz ispis teksta "Nedovoljno memorije" u slučaju bacanja izuzetka tipa "`bad_alloc`"):

```
Unesite broj redova kvadratne matrice: 3
Unesite elemente matrice: 3 12 5 123 -6 2 17 4 -31
3 12 5
123 -6 2
17 4 -31
```

6. Ponovite prethodni zadatak, ali uz korištenje kontinualne umjesto fragmentirane alokacije.