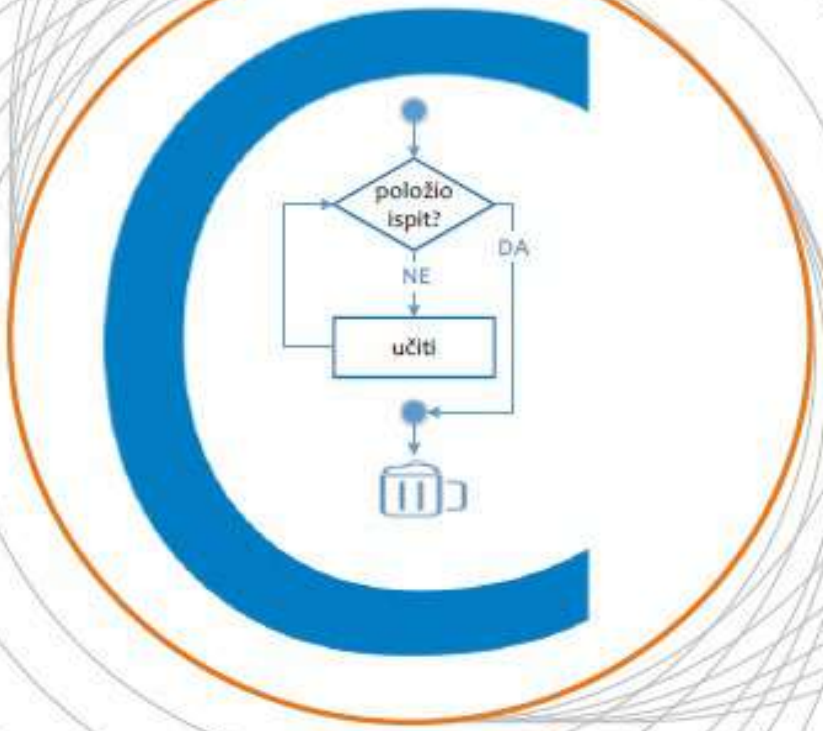
SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE U SPLITU



Ivo Marinić-Kragić
Milan Ćurković

INŽENJERSKA PRIMJENA PROGRAMSKOG JEZIKA C

Inženjerska primjena programskog jezika C

UDŽBENICI SVEUČILIŠTA U SPLITU
MANUALIA UNIVERSITATIS STUDIORUM SPALATENTIS

Nakladnik
FAKULTET ELEKTROTEHNIKE,
STROJARSTVA I BRODOGRADNJE

Urednici
prof. dr. sc. Vladan Papić

Autori
doc. dr. sc. Ivo Marinić-Kragić
izv. prof. dr. sc. Milan Ćurković

Recenzenti
izv. prof. dr. sc. Damir Sedlar
prof. dr. sc. Neven Pavković
doc. dr. sc. Andrijana Ćurković

Lektor
GNOSIS d.o.o.

Objavljivanje ovog udžbenika odobrio je Senat Sveučilišta u Splitu, rješenjem Klasa: 602-09/21-04/05, URBROJ: 2181-202-2-1/1-21-13 od 28. listopada 2021. god.

ISBN 978-953-290-125-2

Prvo izdanje objavljeno u listopadu 2023.g.

INŽENJERSKA PRIMJENA PROGRAMSKOG JEZIKA C

doc. dr. sc. Ivo Marinić-Kragić, izv. prof. dr. sc. Milan Ćurković

Split, 2023.

SADRŽAJ

PREDGOVOR

C je programski jezik opće namjene, dostupan na svim platformama. Često se koristi u inženjerskoj praksi, od programiranja mikrokontrolera (npr. Arduino) do upravljanja programima koji rješavaju složene inženjerske numeričke modele na superračunalima (npr. ANSYS Fluent). Osim što ima široku primjenu također postavlja izvrsne temelje za programiranje u bilo kojem drugom programskom jeziku. Programiranje u inženjerskoj praksi omogućava ubrzanje pojedinih poslova, a nerijetko zbog drugačijeg načina razmišljanja može dovesti i do inovativnih rješenja. Kao primjer redukcije radnih sati može se izdvojiti analiza terenskih mjerenja koja može biti vremenski zahtjevna (mjerenja rada motora ili kotla, mjerenje vibracija, termodinamičke karakteristike zgrade, …). Inženjer treba izdvojiti važne podatke, analizirati ih, donijeti zaključke te konačno izraditi izvještaj. Ako se ovakvi poslovi često ponavljaju, velika je šansa da se dio inženjerskog posla može automatizirati izradom jednostavnih računalnih programa. Dolaskom novih tehnologija može se očekivati da će programiranje postati sve potrebnije u inženjerskim primjenama.

Ovaj je udžbenik namijenjen studentima preddiplomskih i diplomskih studija strojarstva, brodogradnje i industrijskog inženjerstva koji se prvi put susreću s programiranjem. Gradivo ovog udžbenika pokriva uvod u programiranje za inženjere s naglaskom na inženjerske primjene. S obzirom na to da se radi o uvodu u programiranje, udžbenik ne ulazi u sve tehničke detalje programskog jezika C. Ipak, za one koji žele više, u prilozima su objašnjeni neki važniji tehnički detalji. Od predznanja, očekuje se poznavanje matematike na razini prve godine preddiplomskog studija.

Iako postoje slični udžbenici vezani uz primjenu programskog jezika C vezano za inženjerske primjene [1]–[4], doprinos ovog udžbenika je veći naglasak na inženjerske primjene iz područja strojarstva. Također, dodana su detaljnija i postupna objašnjenja najvažnijih pojmova u programiranju te velik broj novih vježbi za samostalan rad. U udžbeniku su kratko objašnjene važne numeričke metode potrebne za rješavanje inženjerskih problema i uz to su izrađeni programi i funkcije za njihovu primjenu. Riješeni su razni praktični problemi koji obuhvaćaju mehaniku, mehaniku fluida, termodinamiku i druge inženjerske probleme. Rješenju prethodi ispravna formulacija problema temeljena na teorijskim postavkama iz pripadnih grana inženjerstva. Ovaj udžbenik ne pokriva potrebno znanje (mehanika, mehanika fluida…) koje je potrebno za ispravnu formulaciju pojedinog inženjerskog problema, ali citirana je literatura prikladna za pojedini problem.

1 OSNOVE

1.1 Prvi program

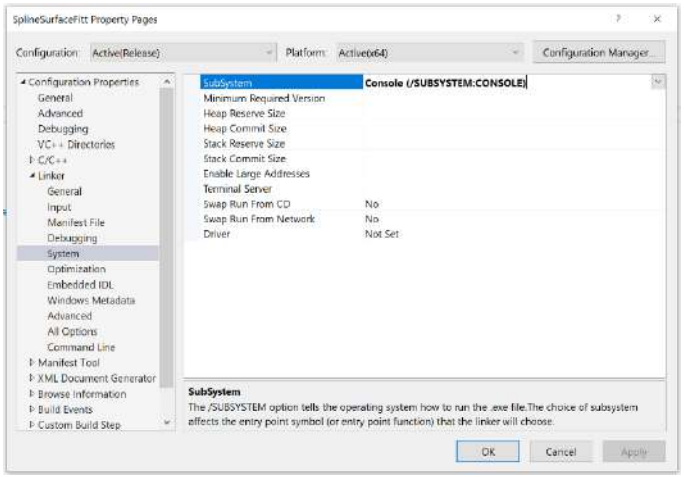
C programi mogu se pisati u bilo kojem programu za uređivanje teksta, te se trebaju pohraniti s nastavkom .c ili .cpp. Za pokretanje programa, potrebni su *prevoditelj* (engl. *compiler*) i *povezivač* (eng. *linker*) koji generiraju izvršnu datoteku (.exe na Windows opertivnom sistemu). Programi u ovoj skripti provjereni su korištenjem najnovije verzije programa *Mircosoft Visual Studio Comunity (2020)* koji možete besplatno preuzeti putem interneta. U nastavku se nalazi programski kôd prvog C programa.

```c
/*Ovo je program koji ispisuje poruku*/
#include<stdio.h>

int main()
{
	printf("Ovo je moj prvi");
	printf(" C program");
	return 0;
}
```

Pokretanjem ovog programa, u komandnom prozoru ispisuje se poruka:

```
Ovo moj prvi C program
Press any key to close this window ...
```

Ako koristite Visual Studio i komandni prozor nestane nakon pokretanja programa, tada unutar postavki projekta odaberite Console kao SubSystem:



Sastavni dijelovi koje ovaj program sadržava su: Komentar, uključivanje biblioteka, glavna (*main*) funkcija te naredba za ispis.

- U prvoj liniji programskog kôda nalazi se **komentar** koji počinje s ' /* ' te završava s '*/'. Komentari nisu obavezni, ali često se koriste zbog lakše razumljivosti programa. Programski kôd može imati proizvoljno mnogo komentara.
- U sljedećoj liniji nalazi se `#include<stdio.h>` što predstavlja uključivanje **biblioteke** sa standardnim funkcijama za unos i čitanje iz komandnog prozora. Ovo je potrebno da bi se mogle koristiti

funkcije kao što su *printf*, *printf_s*, *scanf* i *scanf_s* koje omogućavaju korisniku interakciju s programom.

- Funkcija ***main()*** predstavlja glavnu funkciju i označava početak svakog C programa. U ovoj se funkciji piše redom niz naredbi koje program treba izvršavati. Svaka naredba završava znakom točka-zarez.
- Prvi program sadržava samo naredbu za ispis poruke *„Ovo je moj prvi C program“*. Ova naredba izvršava se pozivom funkcije ***printf***. Funkcija *printf* ovdje služi samo za ispis tekstualnog niza znakova unutar navodnika.
- Program završava kada u glavnoj funkciji dođe do ***return 0;*** izraza.

Pitanja za provjeru znanja 1.1

- Što će se ispisati nakon što se napravi mala izmjena u *main* funkciji (u čemu je razlika?):

```
printf("Ovo je moj prvi");
printf("C program");
return 0;
```

- Što će se dogoditi u sljedećem slučaju:

```
printf("Ovo je moj prvi");
return 0;
printf("C program");
```

1.2 Varijable

Varijable se mogu zamisliti kao spremnici u koje možete pohraniti točno određeni tip podataka (npr. cijeli broj, tekst, matricu,…). Svaka varijabla ima svoj tip, a najčešće se koriste realni brojevi (tip *float* i *double*), cijeli brojevi (tip *int*) i znakovi (slova, znamenke…) (tip *char*).

NAZIV VARIJABLE Kod nazivanja varijabli potrebno je paziti na sljedeće:

- Nazivi su osjetljivi na mala i velika slova odnosno *mojbroj* i *mojBroj* su dvije različite varijable.
- Nazivi varijabli moraju početi sa slovom i ne smiju sadržavati razmake (*broj_1* može biti varijabla dok „*broj 1*“ ili „*1_broj*“ ne mogu).
- Broj znakova naziva može biti proizvoljno velik, ali u nekim je slučajevima ograničen na 31.
- Varijabla se ne može zvati isto kao pred-definirane C naredbe ili funkcije (npr. *printf* ne može biti naziv varijable).
- Zbog preglednosti programa (i preglednosti vašeg rješenja na **ispitu**!), potrudite se dati varijablama smisleno ime umjesto kratice. Na primjer, ako vam treba varijabla koja pohranjuje izračunato maksimalno naprezanje u grednom nosaču, naziv *maxNaprezanje* je puno bolji nego naziv *mn*.

PRIJAVLJIVANJE VARIJABLI Varijable je potrebno prijaviti prije korištenja. Varijable se prijavljuju na način da se prvo napiše željeni tip varijable, a zatim željeni naziv, primjer:

```
int brojKomada;
```

Nakon prijavljivanja, naredbi se **pridjeljuje** vrijednost. Naziv varijable uvijek se mora nalaziti na lijevoj strani, nakon čega slijedi znak “=” koji se naziva operator pridjeljivanja:

```
brojKomada = 23;
```

ISPIS VARIJABLI Za ispis vrijednosti varijabli, koristi se funkcija *printf*. Kod ispisa je potrebno paziti na tip varijable. Ako se želi ispisati cjelobrojna vrijednost, koristi se oznaka *%d ili %i*, a za realne tipa *float* koristi se *%f*, odnosno *%lf* za tip *double*:

```
/*Prijavljivanje varijabli:*/
int brojKomada;
double masa;
/*Pridjeljivanje vrijednosti:*/
brojKomada = 23;
jedinicnaMasa = 6.022;
/*Ispis:*/
printf("Broj komada: %d\n", brojKomada); /* "\n" - nova linija */
printf("Jedinicna masa: %lf kg/kom\n", jedinicnaMasa);
```

Rezultat programa je ispis:

```
Broj komada: 23
Jedinicna masa: 6.022000 kg/kom
```

Kod ispisa, oznaka '*\n*' predstavlja simbol za novu liniju teksta. Bez dodavanja nove linije cijeli bi ispis bio u jednoj liniji teksta.

U jednoj naredbi za ispis, može se ispisivati i više različitih varijabli, a dopušteno je i korištenje aritmetičkih operacija:

```
printf("Ukupna masa svih (%d kom.): %lf kg\n", brojKomada,jedinicnaMasa*brojKomada);
```

Ispis prethodne linije u komandnom prozoru je:

```
Ukupna masa svih (23 kom.): 138.506000 kg
```

IZMJENA VRIJEDNOSTI Vrijednost varijable može se mijenjati u toku programa. Na primjer:

```
int mojBroj;
mojBroj = 3;
printf("mojBroj=%d\n", mojBroj);
mojBroj = 5;
printf("mojBroj=%d\n", mojBroj);
mojBroj = mojBroj-1;
printf("mojBroj=%d\n", mojBroj);
```

Rezultat programa je:

```
mojBroj=3
mojBroj=5
mojBroj=4
```

U prvoj naredbi, varijabli *mojBroj* dodijeljena je vrijednost *3*. U drugoj naredbi varijabli *mojBroj* dodijeljena je vrijednost *5* (stara vrijednost prebrisana). U trećoj liniji, varijabli *mojBroj* dodijeljena je vrijednost izraza „*mojBroj-1*“. Ovdje se prvo evaluira izraz s desne strane znaka jednakosti koji se zatim pridjeljuje u *mojBroj* . S obzirom na to da je trenutno *mojBroj=5*, evaluacija izraza daje *5-1=4*.

TIPOVI VARIJABLI Dva navedena tipa realnih brojeva (*float* i *double*) razlikuju se po preciznosti koju mogu pohraniti. Varijable tipa *float* koriste 32 bita u memoriji, dok varijable tipa *double* koriste 64 bita

čime imaju značajno veću preciznost. Ako se u varijablu pokuša pohraniti broj π s 20 decimalnih mjesta, program će u varijablu pohraniti najbliži mogući broj:

```
double xD = 3.14159265358979323846;
float xF = 3.14159265358979323846;
printf("pi=3.14159265358979323846\n");
printf("xD=%.20lf\n", xD);
printf("xF=%.20f\n", xF);
```

Ispis rezultata:

```
pi=3.14159265358979323846
xD=3.14159265358979311600
xF=3.14159274101257324219
```

U ispisu vidimo da je varijabla tipa *double* ispravno pohranila broj do 15. decimalnog mjesta dok je varijabla *float* samo do šestoga.

Kod istovremenog rada s cijelim i realnim brojevima potrebno je paziti na ograničenja varijabli koje su prijavljene kao cijeli broj. Ako pod cijeli broj pokušate pohraniti realni broj, rezultat će biti zaokružen na najbliži manji cijeli broj. Naravno, može se raditi i obratno, a pri tome se cijeli broj pretvara u realni:

```
int a = 10;
double b = 8.314;
a = b;
printf("a=%d\n", a);
b = a;
printf("b=%lf\n", b);
```

Ispis:

```
a=8
b=8.000000
```

Pitanja za provjeru znanja 1.2

- Sljedeći program ispisuje vrijednost varijabli *a*, *b* i *c*. Bez pokretanja programa zaključite što će biti ispisano za pojedinu varijablu: 0, 1, 1.333, 2, 5, 7, 9 ili GREŠKA? Program (tj. bitni dio programa):

```
int a, b = 2, c = 1;
a = 1 + 2 * 3;
b = 4 / 3;
c = c + 1;
printf("a=%d b=%d c=%d", a, b, c);
```

- Što ako su sve varijable realni broj tipa *double*?
- Za sljedeći program, zaključite što će biti ispisano za pojedinu varijablu:

```
int a = 1, b = 2, c = 3;
a + b = c;
printf("a=%d b=%d c=%d", a, b, c);
```

Pretvaranje tipa varijable ***Dodatak***

U C programiranju nije dopušteno mijenjati tip varijable u toku programa. Ako se ovo pokuša, prevoditelj će javiti grešku. Na primjer, ako se cijeli broj pretvara u realni:

```
int var = 10;
var = var / 3;
double var; //Greska!
```

Umjesto prethodnog, potrebno je dodati zasebnu varijablu za realni broj. Kod pretvorbe se zatim koristi operator pretvaranja. Za slučaj pretvorbe broja u realni, potrebno je prije izraza napisati operator pretvorbe `(double)`:

```
int suma = 10, brojac=3;
double ave;
ave = suma / brojac; //Bez pretvorbe cijelog broja u realni
printf("ave = suma / brojac = %lf\n", ave);
ave = (double) suma / brojac; //Dodan operator pretvorbe
printf("ave = (double) suma / brojac = %lf\n", ave);
```

Ispis pokazuje rezultat u slučaju bez pretvorbe i s pretvorbom u realni broj:

```
ave = suma / brojac = 3.000000
ave = (double) suma / brojac = 3.333333
```

Operator pretvaranja djeluje samo na najbližu varijablu, odnosno sumu (varijabla *suma*) pretvara u realni broj. Ako se prvo napravi dijeljenje dva cijela broja (10/3=3), a zatim pretvaranje u realni broj, rezultat neće biti isti:

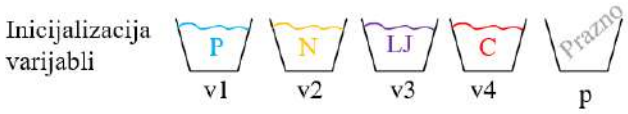
```
ave = (double) (suma / brojac) = 3.000000
```

Generalno, za pretvoriti jedan tip varijable u drugi koristi se operator pretvaranja, općeg oblika:

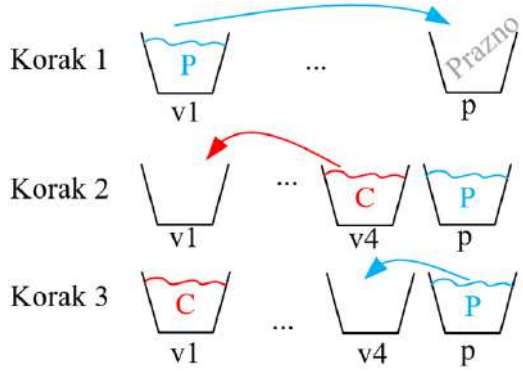
```
(željeniTip) varijabla
```

PRIMJER 1 Potrebno je napraviti program koji zamjenjuje vrijednosti dviju odabranih varijabli.

Rješenje problema može se prvo prikazati vizualno, pomoću posuda ispunjenih različitim bojama. Inicijalizacija vrijednosti varijabli tako se može za primjer s 5 varijabli (*v1-v4* i *p*) prikazati:



Ako zamislimo posude u stvarnosti, potrebno je nekoliko koraka da bismo zamijenili sadržaj (boju) dviju posuda. Uzmimo za primjer zamjenu sadržaja posuda *v1* i *v4*, gdje se *p* koristi kao privremena posuda, a strelice označavaju prelijevanje boje iz jedne posude u drugu:



Za zamjenu boje između posuda *v1* i *v4*, koristi se pomoćna posuda *p*. Boja se prvo prelije u pomoćnu posudu, čime se posuda *v1* isprazni. Sada se *v4* može preliti u *v1*, te konačno posuda *p* u *v4*. Pazite na malu razliku između ovoga i programiranja: operator pridjeljivanja zapravo kopira vrijednost, čime bi vrijednost nakon prelijevanja boje ostala nepromijenjena u izvornoj posudi. Program koji odgovara opisanom postupku prikazan je u nastavku:

```
#include<stdio.h>
int main()
{
	Int v1=1, v2=3, v3=7, v4=2, pomoc;
	printf("Prije zamjene: v1=%d v4=%d\n", v1, v4);
	pomoc = v1;
	v1 = v4;
	v4 = pomoc;
	printf("Nakon zamjene: v1=%d v4=%d\n", v1, v4);
	return 0;
}
```

Ispis rezultata:

```
Prije zamjene: v1=1 v4=2
Nakon zamjene: v1=2 v4=1
```

1.3 Interakcija s korisnikom

FIKSNI PROGRAM U prethodnim primjerima program je imao fiksne „ulazne“ vrijednosti varijabli. U tom slučaju, za promjenu rezultata programa *programer* treba izmijeniti programski kôd. Na ovom primjeru program računa ukupnu masu za uneseni broj komada (23) te jediničnu masu (6.022):

```
brojKomada = 23;
jedinicnaMasa = 6.022;
printf("Broj komada: %d\n", brojKomada);
printf("Jedinicna masa: %lf kg/kom\n", jedinicnaMasa);
printf("Ukupna masa svih (%d kom.): %lf kg\n", brojKomada,jedinicnaMasa*brojKomada);
```

Ispis programa je:

```
Broj komada: 23
Jedinicna masa: 6.022000 kg/kom
Ukupna masa svih (23 kom.): 138.50600 kg
```

Ako se želi broj komada promijeniti na npr. 41, izmjenu unosi *programer* izravno u programski kôd:

```
brojKomada = 41;
jedinicnaMasa = 6.022;
printf("Broj komada: %d\n", brojKomada);
printf("Jedinicna masa: %lf kg/kom\n", jedinicnaMasa);
printf("Ukupna masa svih (%d kom.): %lf kg\n", brojKomada,jedinicnaMasa*brojKomada);
```

Sada je i ispis programa izmijenjen:

```
Broj komada: 41
Jedinicna masa: 6.022000 kg/kom
Ukupna masa svih (41 kom.): 246.902000 kg
```

KORISNIČKI UNOS (scanf_s) Cilj programiranja je (uglavnom) napraviti program koji *korisnik* upotrebljava bez razmišljanja o programskom kôdu. *Korisnik* programa je osoba koja pokreće program, a od programa se očekuje da pruža mogućnost *interakcije*. Interakcija se u jednostavnom slučaju odnosi primjerice na unos nekog broja (npr. broja komada te jedinične mase). Ovo se postiže funkcijom *scanf_s*, koja zaustavlja program i čeka od korisnika programa traženi unos. Potrebno je definirati kakav tip unosa se očekuje (npr *%lf* za *double*, ili *%d* za *integer*), te u koju varijablu se pohranjuje (npr. *brKomada*). Uz naziv varijable, treba se dodati adresni operator & (operator je objašnjen na kraju poglavlja):

```
int brKomada;
double jedMasa;
printf("Unesite broj komada: ");
scanf_s("%d", &brKomada);
printf("Unesite jedinicnu masu (kg/kom): ");
scanf_s("%lf", &jedMasa);
printf("Ukupna masa svih (%d kom.): %lf kg\n", brKomada, jedMasa * brKomada);
```

Pokretanjem programa, ispisuje se:

```
Unesite broj komada: _
```

Program je sada zaustavljen sve dok korisnik ne unese željeni broj. Sada korisnik utipkava broj (42), te pritiskom na tipku *enter* potvrđuje unos; zatim slijedi unos jedinične mase, te konačno ispis rezultata:

```
Unesite broj komada: 42
Unesite jedinicnu masu (kg/kom): 6.022
Ukupna masa svih (42 kom.): 252.924000 kg
```

Korištenjem funkcije *scanf_s* može se i istom naredbom napraviti unos više različitih vrijednosti, na primjer korisnik unosi jediničnu masu proizvoda i broj komada u jednoj liniji:

```
printf("Unesite jedinicnu masu (kg/kom.) i broj komada: ");
scanf_s("%lf %d", &jedinicnaMasa,&brojKomada);
printf("Ukupna masa iznosi %lf kg\n", jedinicnaMasa* brojKomada);
```

Ovdje treba paziti da se prilikom unosa razmak stavi između dva broja. Primjer izvođenja programa:

```
Unesite jedinicnu masu (kg/kom.) i broj komada: 8.31 64
Ukupna masa iznosi 531.840000 kg
```

Kada se u istoj liniji traži unos više vrijednosti, lako nastane pogreška. Zbog toga se u ovom udžbeniku uglavnom izbjegava ovakav način unosa.

Adresni operatori ***Dodatak***

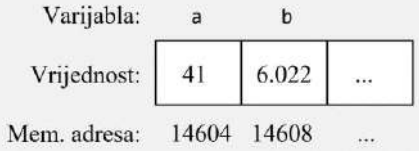
Znatiželjni student će negodovati zbog neobjašnjive pojave adresnog operatora „*&*“ kod funkcije *scanf_s*. Ovaj dio nije važan za razumijevanje programiranja na inženjerskoj razini, te ga možete slobodno preskočiti.

Radna memorija računala (RAM) može se gledati kao niz spremnika za pohranu brojčanih vrijednosti. Ovisno o tipu varijable (*int, double, ...*), pojedini spremnik može zauzimati manje ili više memorije. Uglavnom se radi o spremnicima veličine 4 ili 8 bajtova.

Primjer, prijavljene su dvije varijable *a* i *b*:

```
int a;
double b;
a=41;
b=6.022;
```

Stanje u radnoj memoriji može se vizualizirati na sljedeći način;



Svakoj se varijabli prilikom prijavljivanja dodijeli neka memorijska adresa / memorijski prostor. Na primjer, varijabla *a* nalazi se na adresi 14604, a varijabla *b* na adresi 14608. Varijabla *a* će zauzimati 4 bajta, varijabla *b* 8 bajtova. Adresa varijable ne mijenja se za vrijeme rada programa. Kada se vrijednost pridjeljuje varijabli, ova se vrijednost zapravo zapisuje na točno određeno mjesto u memoriji koje je povezano s tom varijablom.

Kod korištenja funkcije *scanf_s* jedan od argumenata funkcije je adresa varijable. Za saznati adresu varijable koristi se adresni operator *&*. U trenutnom primjeru *&a* nam kao rezultat daje broj 14604. Da bi funkcija *scanf_s* zapisala korisnički unesenu vrijednost na mjesto varijable *a*, potrebno je poznavati adresu na kojoj se ta varijabla nalazi.

PRIMJER 2 Zadatak je napraviti program za izračun prijenosnog omjera te maksimalnog okretnog momenta koji vozač bicikla može proizvesti.

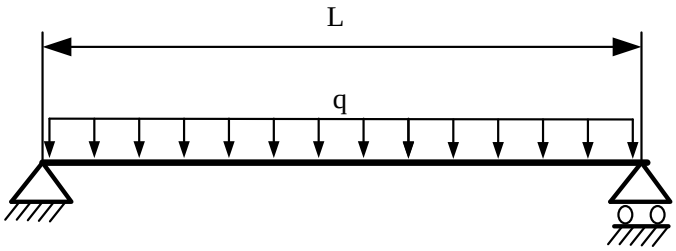
Rješenje Prijenosni omjer se računa kao omjer između broja zubi stražnjeg i prednjeg zupčanika: $R=N_{straznji}/N_{prednji}$. Ako su poznate masa vozača M i duljina pedale L, maksimalni okretni moment može se računati izrazom T=R*M*9.81*L [Nm].

```
#include<stdio.h>
int main()
{
	int brojZubiStraznji, brojZubiPrednji;
	double duljinaPedale, masaVozaca,R,T;
	printf("Unesite broj zubi prednjeg zubcanika: ");
	scanf_s("%d", &brojZubiPrednji);
	printf("Unesite broj zubi straznjeg zubcanika: ");
	scanf_s("%d", &brojZubiStraznji);
	R = (double)brojZubiStraznji / brojZubiPrednji;
	//Komentar: "(double)" pretvara cijele brojeve u realne

	printf("Unesite masu vozaca [kg] : ");
	scanf_s("%lf", &masaVozaca);
	printf("Unesite duljinu tj. radijus pedale [mm] : ");
	scanf_s("%lf", &duljinaPedale);
	duljinaPedale = duljinaPedale / 1000; //Pretvaranje mm u m
	T = R * masaVozaca * 9.81 * duljinaPedale;

	printf("Prjenosni omjer lancanog prijenosa: %lf\n", R);
	printf("Maksimalni moment vozaca: %lf Nm \n", masaVozaca * 9.81 * duljinaPedale);
	printf("Maksimalni okretni moment na straznjem kolu: %lf Nm \n", T);
	return 0;
}
```

Pokretanjem programa, unose se tražene vrijednosti, a zatim slijedi i ispis rezultata:

```
Unesite broj zubi prednjeg zubcanika: 45
Unesite broj zubi straznjeg zubcanika: 15
Unesite masu vozaca [kg] : 80
Unesite duljinu tj. radijus pedale [mm] : 150
Prjenosni omjer lancanog prijenosa: 0.333333
Maksimalni moment vozaca: 117.720000 Nm
Maksimalni okretni moment na straznjem kolu: 39.240000 Nm
```

PRIMJER 3 Zadatak je izraditi program za izračun maksimalnog naprezanja i progiba za gredu oslonjenu na dva oslonca sa zadanim kontinuiranim opterećenjem kako je shematski prikazano na donjoj slici. Greda je napravljena od konstrukcijskog čelika, a poprečni presjek je pravokutan. Korisnik unosi iznos kontinuiranog opterećenja, duljinu grede i dimenzije pravokutnog presjeka.



Rješenje Iz statike [5] i mehanike materijala [6], [7], poznati su izrazi za maksimalno naprezanje i maksimalni progib za ovakvu gredu:

$$\sigma_{max} = \frac{y_{max} \cdot q \cdot L^2}{8 \cdot I} \text{ i } \delta_{max} = \frac{5 \cdot q \cdot L^4}{384 \cdot E \cdot I} \tag{1.1}$$

gdje je σ_{max} maksimalno naprezanje, a δ_{max} maksimalni progib. Najveća udaljenost bilo koje točke poprečnog presjeka od neutralne linije je y_{max}, q je iznos kontinuiranog opterećenja, L je duljina grede, I je moment tromosti poprečnog presjeka i E je modul elastičnosti materijala. Za slučaj pravokutnog poprečnog presjeka, moment tromosti se računa prema $I=(bh^3)/12$, gdje je b širina poprečnog presjeka, a h visina. Riješeni program je u nastavku.

```c
#include<stdio.h>

int main()
{
	double q, L, I, E, y, b, h;
	double Sigma, maxProgib;

	printf("Ovo je program za proracun grede oslonjene na dva oslonca \n");
	printf("	q[N/mm]   \n");
	printf(" \\/\\/\\/\\/\\/\\/\\/\\/\\/\\/\n");
	printf(" _________________ \n");
	printf(" ^                 ^ \n");
	printf("Molim unesite trazene velicine... \n\n");

	printf("Unesi duljinu grede [mm]: \n");
	scanf_s("%lf", &L);
	printf("Unesi iznos kontinuiranog opterecenja [N/mm]: \n");
	scanf_s("%lf", &q);
	printf("Unesi modul elasticnosti materijala [N/mm^2]: \n");
	scanf_s("%lf", &E);
	printf("Unesi visinu poprecnog presjeka [mm]: \n");
	scanf_s("%lf", &h);
	printf("Unesi sirinu poprecnog presjeka [mm]: \n");
	scanf_s("%lf", &b);
	I = (b*h*h*h) / 12;
	y = h / 2;

	printf("\n Izracun u tijeku...\n");
	Sigma = y * q*L*L / (8 * I);
	maxProgib = 5 * q*L*L*L*L / (E*I * 384);
	printf("Ispis rezultata:\n");
	printf("Iznos maksimalnog naprezanja u gredi: %lf [N/mm^2] \n", Sigma);
	printf("Maksimalni progib iznosi: %lf [mm] \n", maxProgib);

	return 0;
}
```

Nakon pokretanja programa prvo se ispisuju informativne poruke o programu. Zatim se program zaustavlja na naredbi za unos duljine grede. Korisnik programa treba napraviti unos svih podataka, zatim slijedi izračun i ispis rezultata:

```
Ovo je program za proracun grede oslonjene na dva oslonca
       q[N/mm]
 \/\/\/\/\/\/\/\/\/
 ____________________
 ^                ^
Molim unesite trazene velicine...

Unesi duljinu grede [mm]:
1000
Unesi iznos kontinuiranog opterecenja [N/mm]:
1
Unesi modul elasticnosti materijala [N/mm^2]:
210000
Unesi visinu poprecnog presjeka [mm]:
30
Unesi sirinu visinu poprecnog presjeka [mm]:
20

 Izracun u tijeku...
Ispis rezultata:
Iznos maksimalnog naprezanja u gredi: 41.666667 [N/mm^2]
Maksimalni progib iznosi: 1.377866 [mm]
```

1.4 Aritmetički operatori i osnovne matematičke funkcije

Matematički izrazi mogu se stvarati kombinacijom aritmetičkih operatora, logičkih operatora, funkcija, varijabli i brojčanih vrijednosti. Aritmetički operatori su zbrajanje (+), oduzimanje (-), množenje (*) i dijeljenje (/). Osim toga, postoji velik broj ugrađenih matematičkih funkcija koji se dodaje uključivanjem biblioteke *math.h*: `#include<math.h>`. U ovo spadaju trigonometrijske funkcije sinus (*sin*), kosinus (*cos*) i tangens (*tan*), funkcija za izračun korijena broja (*sqrt*), funkcija za zaokruživanje broja na najbliži cijeli broj (*round*), funkcija za izračun apsolutne vrijednosti (*fabs*) itd. U osnovnim postavkama, matematičke funkcije rade s tipom varijable *double*. Funkcija se poziva na način da se napiše naziv funkcije, a nakon toga ulazni argument unutar zagrada. Na primjer, izračun korijena broja 2 te pohrana u varijablu *var*:

```
double var;
var = sqrt(2);
printf("var: %lf\n", var);
```

U ispisu se vidi da je u varijablu pohranjen broj:

```
var: 1.41421
```

PRIMJER 4 Potrebno je napraviti program za izračun vertikalne visine točke (npr. prozor na zgradi) za izmjerenu horizontalnu udaljenost (d = 20 m) i kut u odnosu na horizontalu (α=30°).

Rješenje Za poznatu udaljenost d i kut (α), visina se računa po izrazu koji pretpostavlja pravokutni trokut: $h=d \cdot tan(\alpha)$. Program s unesenim zadanim podacima je u nastavku:

```
#include<stdio.h>
#include<math.h>

int main()
{
	double pi = 3.14159;
	double kut,udaljenost,visina;

	kut = 30.0 / 180.0 * pi;
	udaljenost = 20.0;
	visina = udaljenost * tan(kut);
	printf("Izracunata visina: %lf\n", visina);
	return 0;
}
```

Rezultat programa:

```
Izracunata visina: 11.546994
```

Nakon prijavljivanja ostalih potrebnih varijabli, u prvoj se liniji vrijednost kuta od 30° pretvara u radijane, što je potrebno da bi se vrijednost mogla koristiti kao ulaz u trigonometrijske funkcije. Sljedeća naredba varijabli *udaljenost* pridjeljuje zadanu vrijednost od 20 m. Treća naredba računa *visinu* koristeći prethodne rezultate. Obratite pažnju da se cijeli izračun može napisati jednoj liniji:

```
printf("%lf“,20*tan(30/180*3.14))
```

Međutim, zbijanje puno operacija u jednoj liniji može umanjiti preglednost programa.

REDOSLIJED OPERATORA Kod složenih izraza koji sadržavaju više aritmetičkih operatora potrebno je paziti na redoslijed izvršavanja matematičkih operacija. Među aritmetičkim operacijama, prvo se izvršavaju operacije množenja i dijeljenja, a zatim zbrajanja i oduzimanja. Kada se koriste zagrade, prvo se izvršavaju operacije u zagradama, a zatim ostatak izraza. Na primjer za `var=2+2*2`, rezultat je var=6; u slučaju `var=(2+2)*2`, rezultat var=8.

OSTATAK DIJELJENJA Operator ostatka dijeljenja cjelobrojnih vrijednosti (*%*) često se koristi u primjerima za vježbu. Na primjer, broj 9 djeljiv je s brojem 3, te će rezultat biti nula (*9%3=0*). Ako broj nije djeljiv, rezultat je ono što preostane nakon dijeljenja, primjer:

```
int a = 4, b = 5, c = 12;
printf("Ostatak dijeljenja broja 4 s 2 je: %d\n", 4 % 2);
printf("Ostatak dijeljenja broja 5 s 2 je: %d\n", 5 % 2);
printf("Ostatak dijeljenja broja c s a je: %d\n", c % a);
printf("Ostatak dijeljenja broja c s b je: %d\n", c % b);
```

Ispis programa:

```
Ostatak dijeljenja broja 4 s 2 je: 0
Ostatak dijeljenja broja 5 s 2 je: 1
Ostatak dijeljenja broja c s a je: 0
Ostatak dijeljenja broja c s b je: 2
```

Pitanja za provjeru znanja 1.3

- Što će biti rezultat izvršavanja sljedećih naredbi:

```
int a = 4, b = 5, c = 12, d=13;
printf("c%d iznosi: %d\n", c % d);
printf("d%c iznosi: %d\n", d % c);
printf("a+b%c iznosi: %d\n", a+b%c);
printf("a+b%c iznosi: %d\n , a+b%c");
```

- Što će biti rezultat sljedećih operacija:

```
printf("Test 1: %lf\n", sin(90));
printf("Test 2: %lf\n", sin(0)/cos(0));
printf("Test 3: %lf\n", cos(0)/sin(0));
printf("Test 4: %lf\n", (sin((0))+(90)));
printf("Test 5: %lf\n", sqrt(-1*fabs(-1));
```

1.5 Usporedbe i logički operatori

Kod korištenja logičkih i relacijskih operatora, rezultat može biti samo istina ili laž.

RELACIJSKI OPERATORI Operatori usporedbe prikazani su u Tablici 1. skupa sa značenjem i primjerom usporedbe. Istina se označava brojem 1, a laž brojem 0. Obratite pažnju na operator jednakosti, koji se označava == za razliku od operatora pridjeljivanja koji se označava =.

Tablica 1. Operatori usporedbe

Operator	Značenje	Primjer	Rezultat
<	manje od	2<5	1
>	veće od	2>5	0
<=	manje ili jednako od	7<=7	1
>=	veće ili jednako od	5>=7	0
==	jednakost	3==5	0
!=	nejednakost	3!=5	1

U sljedećem primjeru varijabli *mojBroj* dodijeljena je vrijednost 2, a zatim se varijabla uspoređuje s vrijednosti 2 i rezultat je *istina*. Ako se zatim operatorom jednakosti varijabla *mojBroj* usporedi s brojem 3, rezultat će biti *laž*, a vrijednost pohranjena u varijabli neće se mijenjati:

```
mojBroj = 2;
printf("Rezultat za mojBroj==2: %d\n", mojBroj == 2);
printf("Rezultat za mojBroj==3: %d\n", mojBroj == 3);
printf("mojBroj: %d\n", mojBroj);
```

Ispis programa:

```
Rezultat za mojBroj==2: 1
Rezultat za mojBroj==3: 0
mojBroj: 2
```

Rezultati logičkih operatora mogu se također pohranjivati u varijable:

```
mojBroj = 2 < 5;
printf("Prvo, mojBroj: %d\n", mojBroj);
mojBroj = 2 > 5;
printf("Drugo, mojBroj: %d\n", mojBroj);
```

Ispis programa:

```
Prvo, mojBroj: 1
Drugo, mojBroj: 0
```

LOGIČKI OPERATORI Logički operatori su „i“, „ili“ i „ne“. Ovim se operatorima uspoređuju dvije logičke vrijednosti kao što je prikazano u Tablici 1.2. Logički se operator „i“ u C-u označava &&, „ili“ kao || dok je operatoru „ne“ dodijeljena oznaka „!“. Primjer primjene je provjera nalazi li se unutar nekih granica, odnosno je li broj istovremeno veći od donje granice i manji od gornje granice.

```
int broj,provjera,donjaGranica=10,gornjaGranica=20;
broj = 13;
provjera = (broj >= donjaGranica) && (broj <= gornjaGranica);
printf("Broj se nalazi u granicama (0-ne, 1-da): %d", provjera);
```

Ispis programa:

```
Broj se nalazi u granicama (0-ne, 1-da):1
```

U donjoj tablici logički se operatori primjenjuju na varijablama A i B, kojima je u svakom retku dodijeljena drugačija vrijednost, a rezultati su prikazani na desnoj strani tablice.

Tablica 1.2. Operatori usporedbe

A	**B**	**izraz:** **značenje:**	**A&&B** **A i B**	**A||B** **A ili B**	**!A** **ne A**
istina	laž	rezultati:	laž	istina	laž
laž	istina		laž	istina	istina
istina	istina		istina	istina	laž
laž	laž		laž	laž	istina

Za razliku od operatora „i“ i „ili“, operator *ne* (*!*)primjenjuje se na samo jednoj varijabli/izrazu i daje kao rezultat obrnutu vrijednost od trenutne.

Pitanja za provjeru znanja 1.4

- Ako je broj *a*=4, i znamo da je *a*/*b*=2, što će se dogoditi nakon sljedećih operacija:

```
a/b=2;
a/b==2;
```

- Što će biti rezultati u sljedećem primjeru s varijablama *a*, *b* i *c*:

```
a=1==2;
b=1==1;
c=a==b;
printf("a=%d b=%d c=%d",a,b,c);
```

- Što će biti rezultat sljedećih operacija:

```
printf("Test 1: %lf\n", 2>3 || 3>2);
printf("Test 2: %lf\n", 2>3 && 3>2);
printf("Test 3: %lf\n", !(2>3) && !(!(3>2)) );
printf("Test 4: %lf\n", 2>3 || 2!=3);
```

1.6 Operatori pridjeljivanja

Osnovni operator pridjeljivanja „=“, koristiti se kada rezultat naredbi s desne strane operatora želimo pridijeliti varijabli navedenoj s lijeve strane. Postoje specijalne varijante ovog operatora koje se mogu koristiti za jednostavniji zapis. Primjer česte potrebne operacije je uvećavanje vrijednosti varijable za 1. Korištenjem operatora pridjeljivanja ovo se postiže naredbom *var=var+1*, a isto se može postići i korištenjem operatora za uvećavanje varijable za 1 (*var*++).

Tablica 1.3. Operatori pridjeljivanja

Operator	**Izmjena varijable *var***	**Primjer za *a*=10**	**Vrijednost varijable *a***
var++	Uvećanje za 1	`a++;`	11
var --	Smanjenje za 1	`a--;`	9
var += *n*	Uvećanje za *n*	`a+=5;`	15
var -= *n*	Smanjenje za *n*	`a-=5;`	5
var *= *n*	Uvećanje za *n* puta	`a*=5;`	50
var /= *n*	Dijeljenje s *n*	`a/=5;`	2

Primjer uvećavanja cijelog i realnog broja:

```
#include<stdio.h>

int main()
{
	int a = 10;
	double b = 8.22;
	printf("Izvorne vrijednosti: a=%d b=%lf \n", a, b);
	a++;
	b++;
	printf("Nakon prve izmjene:  a=%d b=%lf \n", a, b);
	a += b;
	printf("Nakon druge izmjene: a=%d b=%lf \n", a, b);

	return 0;
}
```

Rezultat programa je:

```
Izvorne vrijednosti: a=10 b=8.220000
Nakon prve izmjene:  a=11 b=9.220000
Nakon druge izmjene: a=20 b=9.220000
```

1.7 Generiranje slučajnih brojeva

C funkcija ***rand()*** koristi se za generiranje slučajnog cijelog broja između 0 i nekog velikog broja *RAND_MAX* (broj ovisi o korištenoj verziji, a iznosi najmanje 32767). Za korištenje funkcije potrebno je osim biblioteke *stdio.h* dodati i *stdlib.h*.

```
#include<stdio.h>
#include <stdlib.h>

int main()
{
	int slucajniBroj;
	slucajniBroj = rand();
	printf("Slucajni broj: %d\n", slucajniBroj);
	return 0;
}
```

Ispis:

```
Slucajni broj: 41
```

BROJ U RASPONU U primjerima koje ćemo rješavati, uglavnom je potreban slučajni broj u točno određenom rasponu. Na primjer, za generiranje rezultata bacanja kockice potreban je slučajni broj između 1 i 6. Ovo skaliranje može se napraviti na različite načine, a jedan od njih je korištenje operatora ostatka dijeljenja. Ako se uzme ostatak dijeljenja slučajnog broja s brojem 6, rezultat će sigurno biti između 0 i 5:

```
printf("Slucajni broj: %d\n", rand() % 6);
printf("Slucajni broj: %d\n", rand() % 6);
printf("Slucajni broj: %d\n", rand() % 6);
printf("Slucajni broj: %d\n", rand() % 6);
printf("Slucajni broj: %d\n", rand() % 6);
```

Ispis:

```
Slucajni broj: 5
Slucajni broj: 4
Slucajni broj: 0
Slucajni broj: 0
Slucajni broj: 4
```

Za dobiti slučajne brojeve u rasponu od 1 do 6 potrebno je samo na ovaj rezultat zbrojiti jedinicu. U općem slučaju, slučajni broj između donje granice i gornje granice može se dobiti korištenjem izraza:

```
donjaGranica + rand() % (1 + gornjaGranica - donjaGranica);
```

FUNKCIJA ZA GENERIRANJE SLUČAJNIH BROJEVA Broj u rasponu može se generirati korištenjem funkcije `RandUGranicama(donjaGranica,gornjaGranica)`. Za koristiti ovu funkciju, potrebno je prije glavne funkcije dodati njenu definiciju. Funkcije će biti detaljno objašnjene u zasebnom poglavlju. Za sada, kod korištenja slučajnih brojeva samo prije glavne (*main)* funkcije, definirajte novu funkciju na sljedeći način:

```c
#include<stdio.h>
#include<stdlib.h>

int RandUGranicama (int donjaGranica, int gornjaGranica)
{
	return donjaGranica + rand() % (1 + gornjaGranica - donjaGranica);
}
int main()
{
	printf("Slucajni broj: %d\n", RandUGranicama(1, 9));
	printf("Slucajni broj: %d\n", RandUGranicama(1, 9));
	printf("Slucajni broj: %d\n", RandUGranicama(1, 9));
	return 0;
}
```

Ispis programa:

```
Slucajni broj: 6
Slucajni broj: 9
Slucajni broj: 8
```

Napomene o pseudo-slučajnim brojevima ***Dodatno***

Ako više puta koristite program sa slučajnim brojevima, primijetit ćete da se uvijek radi o potpuno istom nizu brojeva. Za promijeniti niz koristi se funkcija *srand(pocetniBroj)*. Ova funkcija inicijalizira novi niz slučajnih brojeva, koji će biti uvijek isti za istu inicijalizaciju:

```c
srand(0);
	printf("Tri broja: %d,%d,%d\n", rand(),rand(),rand());
	srand(9785);
	printf("Tri broja: %d,%d,%d\n", rand(), rand(), rand());
	srand(0);
	printf("Tri broja: %d,%d,%d\n", rand(), rand(), rand());
```

Ispis:

```
Tri broja: 21238,7719,38
Tri broja: 4876,28364,31992
Tri broja: 21238,7719,38
```

Za neke primjene može biti važno da niz treba biti uvijek drugačiji; tada se prethodni kôd treba još malo modificirati. Za inženjerske potrebe (npr. optimizacija) čak može biti i poželjno da je niz slučajnih brojeva uvijek isti tako da se s istim postavkama uvijek dobiju isti rezultati.

Također, potrebno je napomenuti da brojevi koje generira računalo nisu u potpunosti slučajni brojevi, već tzv. pseudo-slučajni brojevi. Ovo znači da niz brojeva samo sliči na slučajni niz, međutim niz se temelji na konkretnim numeričkim izrazima koji s istim postavkama uvijek daju identičan niz brojeva.

Vrijeme ***Dodatno***

Za automatsku inicijalizaciju slučajnih brojeva, može se koristiti vrijeme. Funkcija `time(0)` kao rezultat daje trenutno vrijeme u sekundama (najčešće mjereno od 01.01.1970), a funkcija se nalazi u biblioteci *time.h*. Primjerice, rezultat ispisa: `printf("Vrijeme=%d\n", time(0));` može biti, 1579013246, a ako ponovno pokrenete program tri sekunde kasnije, rezultat će biti: 1579013249.

Kod inicijalizacije na ovaj način potrebno je paziti na to da će dva niza pozvana jedan za drugim biti izvršena toliko brzo da će imati istu inicijalizaciju:

```
srand(time(0));
printf("Tri broja: %d,%d,%d\n", rand(), rand(), rand());
srand(time(0));
printf("Tri broja: %d,%d,%d\n", rand(), rand(), rand());
```

Rezultat:

```
Tri broja: 32138,17929,30998
Tri broja: 32138,17929,30998
```

Ako se isti program pokrene nešto kasnije, niz će biti drugačiji:

```
Tri broja: 281,23262,7341
```

1.8 Kratki pregled poglavlja

Sljedeća tablica ukratko prikazuje najvažnije naredbe, operatore i funkcije koje su se obradile u prvom poglavlju.

Tablica 1.4. Pregled uvodnog poglavlja

Naredba, operator ili funkcija	Opis
`#include<stdio.h>`	Dodavanje biblioteke sa standardnim funkcijama (*printf, scanf*…)
`#include<math.h>`	Dodavanje biblioteke s matematičkim funkcijama (*sin, cos*, sqrt, fabs…)
`int main(){}`	Izvođenje programa započinje glavnom (*main*) funkcijom
`return 0;`	Naredba označava kraj funkcije
`=`	Operator pridjeljivanja vrijednosti varijablama
`+, -, *, /`	Aritmetički operatori
`==, <=, >=, !=`	Logički operatori
`a%m`	Kada se koristi kao operator, daje ostatak dijeljenja broja *a* s brojem *m*
`;`	Označava kraj naredbe
`int, float, double`	Prijavljivanje brojčane varijable (cijeli broj, realni broj…)
`sin(x), cos(x), …`	Trigonometrijske funkcije
`abs(x) ili fabs(x)`	Funkcije za izračun apsolutne vrijednosti cijelog ili realnog broja
`printf("niz",a,…)`	Funkcija za ispis niza znakova s dodatnim mogućnostima ispisa varijabli pomoću oznaka (%d,%lf,%c,%s) ovisnih o tipu varijable
`scanf_s("format",&var)`	Učitava varijablu tipa označenog pod *format* (%d, %lf) u varijablu *var*
`#include<stdlib.h>`	Dodatna biblioteka sa standardnim funkcijama (*rand, srand…*)
`(double)varInt`	Pretvaranje cijelog broja *varInt* u realni broj
`srand(x0)`	Inicijalizacija niza slučajnih brojeva s cijelim brojem *x0*
`rand()`	Generiranje slučajnog broja
`time(0)`	Trenutno vrijeme u sekundama (najčešće mjereno od 01.01.1970.)

1.9 Inženjerski primjeri

PRIMJER 5 Za zid zgrade sastavljen od tri sloja izračunaj ukupni koeficijent prolaska topline – U [W/m^{2}K]. Koeficijenti toplinskog otpora slojeva mogu se pronaći u strojarskom priručniku [8], a zid se sastoji od:

- toplinske izolacije od mineralne vune debljine 3cm, toplinske provodljivosti λ=0.05 [W/mK]
- šuplje opeke debljine 20cm, λ=0.48 [W/mK]
- vapnene žbuke debljine 1.5 cm, λ=0.8 [W/mK].

Rješenje Izrazi za izračun ukupnog koeficijenta prolaska topline su poznati iz termodinamike [9], [10]. Toplinski otpor pojedinog sloja računa se prema $R=d/\lambda$ gdje je d debljina sloja. Ukupni toplinski otpor R_T računa se kao suma svih otpora, a koeficijent prolaska topline je $1/R_T$. Uzmi u obzir i toplinski otpor uzrokovan konvekcijom (plošni otpor prolaska topline):

- zid prema vani, R=0.04 [m^{2}K/W]
- zid prema unutra, R=0.13 [m^{2}K/W]

Riješeni program prikazan je u nastavku. Prvi dio zadatka je unos svojstava svih zidova i plošnih otpora prolaska topline. Zatim se prema navedenim izrazima računa i ispisuje koeficijent prolaska topline.

```c
#include<stdio.h>

int main()
{
	double lMV = 0.05; // Toplinska provodljivost mineralne vune[W / mK]
	double dMV = 0.03; // Debljina u metrima
	double lSO = 0.48, dSO = 0.2; //Šuplja opeka
	double lVZ = 0.8, dVZ = 0.015; // Vapnena žbuka
	double Ruk = dMV / lMV + dSO / lSO + dVZ / lVZ; // Toplinski otpor zida[m2K / W]
	double Rvanjski = 0.04, Runutarnji = 0.13;// Plošni otpori prolaza topline
	double Uuk; //Ukupni koeficjent prolaza topline[W / m2K]
	Ruk = Ruk + Rvanjski + Runutarnji;
	Uuk = 1 / Ruk;
	printf("Ukupni koeficjent prolaza topline zida: %.2lf [W/m2K]\n", Uuk);

	return 0;
}
```

Rezultat programa je izračunati koeficijent prolaza topline:

```
Ukupni koeficijent prolaza topline zida: 0.83 [W/m2K]
```

PRIMJER 6 Prethodni je primjer pokazao kako izračunati koeficijent prolaza topline za zid. Za stan u stambenoj zgradi s takvim (U=0.83 W/m^{2}K) vanjskim zidom, površine 45 m^2 , izračunaj potrebnu snagu za grijanje u slučaju vanjske temperature T_{out}=-5°C. Potrebno je uzeti u obzir i gubitke zbog prirodne ventilacije. Volumen zraka u stanu je 180 m^3, pretpostavi jednu izmjenu cijelog volumena zraka na sat i toplinski kapacitet zraka od 1.2 kJ/m^{3}K. Pretpostavi učinkovitost sustava grijanja od 70% (gubici zbog nejednolike raspodjele temperature…).

Rješenje Potrebno je izračunati površinske te ventilacijske gubitke topline. Površinski gubici računaju se umnoškom ukupne površine zidova s koeficijentom prolaza topline i razlikom temperatura. Ventilacijski gubici dobiju se umnoškom toplinskog kapaciteta zraka s faktorom broja izmjena zraka svedeno na m^3/s. Uz to, sve se dijeli s učinkovitosti sustava grijanja. Riješeni program je u nastavku.

```
#include<stdio.h>

int main()
{
	double Uzid = 0.83;//koeficjent prolaza topline vanjskog zida[W / m2K]
	double Azid = 45;//Površina vanjskog zida[m2]
	double topKapZrak = 1200;//Toplinski kapacitet zraka po volumenu[J / m3]
	double volZraka = 180;//Ukupni volumen zraka u stanu
	double izmjene = 1;//Broj izmjena zraka na sat
	double Tvanjska = -5;//Vanjska projektna temperatura[°C]
	double Tunut = 20;//Unutarnja projektna temperatura[°C]
	double eta = 0.7;//Učinkovitost sustava grijanja je 70 %
	double Pzidovi, izmjenaZrakaPoSec, Pv;

	// Toplinski gubici kroz zidove[W]:
	Pzidovi = Uzid * Azid * (Tunut - Tvanjska) / eta;
	// Top.gubici zbog ventilacije[W]:
	izmjenaZrakaPoSec = izmjene * volZraka / 3600;
	Pv = topKapZrak * izmjenaZrakaPoSec * (Tunut - Tvanjska) / eta;

	printf("Toplinski gubici kroz zidove iznose %.2f[kW]\n", Pzidovi / 1000);
	printf("Toplinski gubici zbog ventilacije %.2f[kW]\n", Pv / 1000);
	printf("Potrebna snaga za grijanje %.2f[kW]\n", (Pv + Pzidovi) / 1000);

	return 0;
}
```

Ispis:

```
Toplinski gubici kroz zidove iznose 1.33[kW]
Toplinski gubici zbog ventilacije 2.14[kW]
Potrebna snaga za grijanje 3.48[kW]
```

1.10 Zadaci za vježbu

Zadaci za vježbu poredani su od lakših prema težima. Zadacima označenima znakom $^+$ prikazano je rješenje u narednom poglavlju.

1. Napravite program koji ispisuje veliko slovo c, koristeći samo znak #. Primjer izgleda ispisa:

```
   ######
 ##      ##
#
#
#
#
#
 ##      ##
   ######
```

2. Izračunajte i ispišite rezultate sljedećih matematičkih izraza:

$\sqrt{2^2+2^2}$

$\frac{2^{0.5}}{\sqrt{2}}$

$\sin\frac{\pi}{4}$

$\arctan 0.5$

$\log_2 1024$

3$^+$. Napišite program koji računa sumu tri broja, a brojevi se unose u jednoj liniji odvojeni zarezom.
Očekivani tijek programa:

```
Unesi tri broja odvojeno zarezom: 3,6,7
Suma unesenih brojeva iznosi 17
```

4$^+$. Korisnik unosi koordinate točke (*xT*,*yT*), koordinate središta kružnice (*xS*,*yS*) i radijus kružnice (*r*). Program treba ispisati udaljenost točke od kružnice.

5$^+$. Korisnik unosi proizvoljan cijeli broj. Program treba ispisati 1 ako je uneseni broj pozitivan, a nula ako nije pozitivan (nije potrebno koristiti *ako* kontrolu toka).
Očekivani tijek programa:

```
Unesi broj: 3
Broj je (0-negativan ili 1-pozitivan): 1
```

6. Korisnik unosi tri cijela broja. Program treba ispisati koliko unesenih brojeva su pozitivni.
Očekivani tijek programa:

```
Unesi tri broja odvojeno zarezom: 3,-1,2
Uneseno je ukupno 2 pozitivna broja
```

7. Korisnik unosi proizvoljan cijeli broj. Program treba provjeriti radi li se o pozitivnom broju manjem od 100 i djeljivom s 5. Ako je ovo istina, program treba ispisati jedinicu, a ako ne, treba ispisati nulu.

8. U programu prijavite varijable:

```
int ia=2 ib=5;
float fa=2.6, fb=5.8, fc;
double da=2.6, db=5.8, dc;
char z='A';
```

Napravite program koji ispisuje rezultat sljedećih operacija:

`ia+ib`, `ia+fc`, `fa+da`, $\texttt{fa}^{15}$, $\texttt{da}^{15}$, `ia+z`, `fa+ib`, `(int)fa+ib`.

Varijablama pridijelite vrijednosti i zatim ispišite vrijednost varijable:

`fc=`10^{15}, `dc=`10^{15}, `fc=ib/ia`, `fc=(float)ib/ia`, `fc=fb/ia`.

Prilikom ispisa pazite na formatiranje (`%d`, `%f` ili `%1f`). Pogledajte i objasnite rezultate.

9. Napišite program gdje korisnik programa unosi troznamenkasti cijeli broj, a program treba ispisati zasebno pojedine znamenke (stotine, desetice i jedinice). Na primjer, ako korisnik unese broj 356, program treba ispisati: *Broj sadržava 3 stotine, 5 desetica i 6 jedinica*. (Pomoć: Ako je varijabla cijeli broj (*integer*), što će biti rezultat operacije 356/100?).

10. Korisnik unosi željeni iznos eura kao cijeli broj. Potrebno je razbiti taj iznos na najmanji mogući broj novčanica. Pojedina novčanica može vrijediti: 500, 100, 50, 20, 10, 5, 2 ili 1 eura. Očekivani tijek programa:

```
Unesi iznos: 43
Potrebne novčanice su:
2 novčanica od 20 eura
1 novčanica od 2 eura
1 novčanica od 1 eura
```

11. Korisnik unosi koordinate dviju točaka koje definiraju liniju *AB* (*xA, yA, xB, yB*), koordinate središta kvadrata (*xK, yK*) i duljinu stranice kvadrata (*a*), a stranice su paralelne s koordinatnim osima. Program treba ispisati koliko puta linija *AB* siječe stranice kvadrat.

12. Kao 11, ali korisnik unosi i kut zakreta kvadrata (stranice ne trebaju biti paralelne s koordinatnim osima).

1.11 Riješeni primjeri

ZADATAK 3 Korisnik unosi tri varijable. Potrebno je ispisati njihovu sumu.
Rješenje je prikazano ispod.

```
#include<stdio.h>

int main()
{
	double a, b, c;
	printf("Unesi tri broja odvojeno zarezom: ");
	scanf_s("%lf,%lf,%lf", &a, &b, &c);
	printf("Suma unesenih brojeva iznosi %lf\n", a + b + c);

	return 0;
}
```

ZADATAK 4 Korisnik unosi koordinate točke (*xT*, *yT*), koordinate središta kružnice (*xS*, *yS*) i radijus kružnice (*r*). Program treba ispisati udaljenost točke od kružnice.
Rješenje je prikazano ispod.

```
#include<stdio.h>
#include<math.h>

int main()
{
	double xT, yT, xS, yS, r, d;

	printf("Unesite koordinate tocke (xT,yT): ");
	scanf_s("%lf,%lf", &xT, &yT);
	printf("Unesite koordinate sredista kruznice (xT,yT): ");
	scanf_s("%lf,%lf", &xS, &yS);
	printf("Unesite radijus  kruznice: ");
	scanf_s("%lf", &r);
	d = fabs( sqrt(pow(xT - xS, 2) + pow(yT - yS, 2)) - r );
	printf("Udaljenost od tocke T do kruznice iznosi %lf\n", d);

	return 0;
}
```

Test programa:

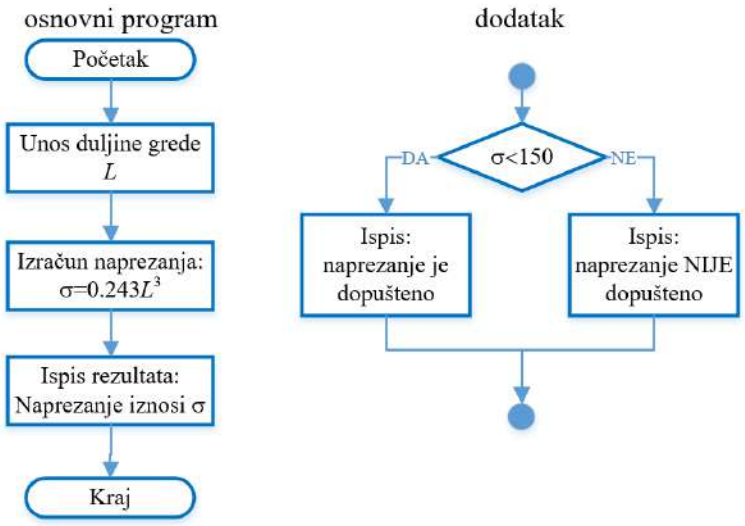
```
Unesite koordinate tocke (xT,yT): 5,4
Unesite koordinate sredista kruznice (xT,yT): 2,1
Unesite radijus  kruznice: 2.82
Udaljenost od tocke T do kruznice iznosi 1.422641
```

ZADATAK 5 Korisnik unosi proizvoljan cijeli broj. Program treba ispisati 1 ako je uneseni broj pozitivan, a nula ako nije pozitivan (nije potrebno koristiti *ako* kontrolu toka).
Rješenje Kada se koristi operator usporedbe „>“ ili „<“, rezultat operatora je „0“ ili „1“.

2 UVJETNA KONTROLA TOKA

Do sada se program izvršavao liniju po liniju, redoslijedom kojim su napisane. Međutim, često je potrebno dio programa izvršavati samo ako su zadovoljeni određeni uvjeti ili je potrebno neki dio programa ponavljati sve dok je ispunjen određeni uvjet. U tim slučajevima koriste se uvjetna kontrola toka, odnosno petlje opisane u nastavku.

Jednostavni primjer potrebe za uvjetnom kontrolom prikazan je na sljedećem primjeru. Osnovni program za proračun grednog nosača (iz prethodnog poglavlja) računa i ispisuje maksimalno naprezanje. Ako želimo napraviti dodatak da nam program ispisuje je li naprezanje dopušteno ili nije, potrebno je koristiti uvjetnu kontrolu toka programa. Ovo je shematski, u obliku dijagrama toka, prikazano na slici ispod.



2.1 Ako uvjetna kontrola

Uvjetna kontrola toka koristi se u slučajevima kada je dio programa potrebno izvršiti samo *ako* je neki uvjet zadovoljen. Općenito, *if* kontrola piše se u obliku:

```
if (uvjet)
{
    radnje;
}
```

Uvjet može biti bilo koji logički izraz (poglavlje 1.5), kojemu rezultat može biti ili *istina* ili *laž*. Ako je uvjet jednak *istini*, tada se izvršavaju radnje koje se nalaze između vitičastih zagrada.

PRIMJER 1 Napravi program koji za varijablu provjerava predznak broja.

Rješenje je prikazano u nastavku.

```
var = -4;
if (var < 0)
{
      printf("Varijabla je negativan broj!\n");
}
```

Korisno je znati da vitičaste zagrade nisu potrebne ako se pod uvjetnim radnjama nalazi samo jedna radnja, tj. ako se blok naredbi sastoji od samo jedne naredbe:

```
var = -4;
if (var < 0)
	printf("Varijabla je negativan broj!\n");
```

PRIMJER 2 Napravi program koji za korisnički unesenu varijablu provjerava je li negativna. Ako je negativna, potrebno ju je pretvoriti u pozitivnu i ispisati poruku.

Rješenje U slučaju da korisnik za vrijeme korištenja programa slučajno unese negativnu vrijednost (*var<0*), potrebno je upozoriti korisnika da se radi o pogrešnom unosu i ispraviti pogrešku. Kontrola s uvjetom (*var<0)* je često potrebna za provjeru korisnički unesenih dimenzija (udaljenosti, visina poprečnog presjeka itd.), jer dimenzija ne bi smjela biti negativna. Ovdje je pogreška ispravljena na jednostavan način – ako je broj negativan, uzima se njegova apsolutna vrijednost. Rješenje je prikazano u nastavku.

```
#include<stdio.h>
#include<math.h>
int main()
{
	double dim;
	printf("Unesi dimenziju [mm]: ");
	scanf_s("%lf", &dim);
	if (dim < 0)
	{
		printf("Unesena dimenzija (%.2f [mm]) nije ispravna\n", dim);
		dim = fabs(dim);
		printf("Ispravak: dimenzija sada iznosi; %.2f [mm]\n", dim);
	}
	return 0;
}
```

Samo u slučaju unosa negativnog broja program će ispisati poruku o automatskom ispravku unesene vrijednosti:

```
Unesi dimenziju [mm]: -5.65
Unesena dimenzija (-5.65 [mm]) nije ispravna
Ispravak: dimenzija sada iznosi; 5.65 [mm]
```

2.2 Uvjetna kontrola*: if-else*

Nešto složenija uvjetna kontrola od prethodne je ***if-else*** (hrvatski *ako-inače*) kontrola, čiji je opći oblik:

```
if ( uvjet )
{
	radnje1;
}
else
{
	radnje2;
}
```

U slučaju da je *uvjet* ispunjen, izvršit će se *radnje1*, a ako *uvjet* nije ispunjen, izvršit će se *radnje2*.

PRIMJER 3 Napravi program koji za uneseni postotak ostvaren na ispitu provjerava je li ocjena prolazna i ispisuje primjerenu poruku.

Rješenje je prikazano u nastavku.

```
double postotak;
printf("Unesite ostvareni postotak: ");
scanf_s("%lf", &postotak);
if(postotak > 50)
	printf("Ocjena je prolazna, Cestitamo");
else
	printf("Nazalost, morate ponovo pisati ispit");
```

Pokrenite program i unesite broj 50, ispis je:

```
Unesite ostvareni postotak: 50
Nazalost, morate ponovo pisati ispit
```

Pitanja za provjeru znanja 2.1

- Slijedi program koji ispisuje je li broj pozitivan ili negativan. Ima li razlike između sljedeća dva kôda? Ako ima razlike, koji je kôd točniji i/ili kako popraviti neispravni dio?

  Kôd 1:

```
if (broj>0)
	printf("Broj je pozitivan ");
else
	printf("Broj je negativan ");
```

  Kôd 2:

```
if (broj>0)
	printf("Broj je pozitivan ");
if (broj<0)
	printf("Broj je negativan ");
```

2.3 Uvjetna kontrola: *else-if*

Ako je potrebno napraviti program koji za uneseni postotak ispisuje ocjenu (1 do 5), tada se koristi *else-if* (inače-ako) uvjetna kontrola. Opći oblik *else-if* kontrole je:

```
if (uvjet1)
	{radnje1;}
else if (uvjet2)
	{radnje2;}
else if (uvjet3)
	{radnje3;}
/*itd*/
else
	{radnjeZadnje;}
```

U slučaju kada prvi uvjet (*if*) nije zadovoljen (samo u tom slučaju), testira se drugi uvjet (*elseif*); u slučaju da ni drugi uvjet nije ispunjen, ispituje se treći; i tako redom ovisno o broju *elseif* uvjeta. Zatim se na kraju može nalaziti *else* skupina radnji, koja se izvršava samo ako ni jedan od prethodnih uvjeta nije zadovoljen.

PRIMJER 4 Napravi program koji za uneseni postotak ostvaren na ispitu, ispisuje ostvarena ocjena.

Rješenje je prikazano u nastavku.

```
double postotak;
printf("Unesite ostvareni postotak: ");
scanf_s("%lf", &postotak);
if (postotak >= 85 && postotak <= 100)
printf("Cestitamo, dobili ste ocjenu odlican(5)");
else if (postotak >= 75)
       printf("Cestitamo, dobili ste ocjenu vrlo dobar(4)");
else if (postotak >= 60)
       printf("Cestitamo, dobili ste ocjenu dobar(3)");
else if (postotak >= 50)
       printf("Cestitamo, dobili ste ocjenu dovoljan(2)");
else
       printf("Nazalost, morate ponovo pisati ispit");
```

Pokretanjem programa i unosom postotka, ispisuje se ostvarena ocjena:

```
Unesite ostvareni postotak: 60
Cestitamo, dobili ste ocjenu dobar(3)
```

Pitanja za provjeru znanja 2.2

- Sljedeći kôd provjerava je li broj djeljiv s 2,3 ili 5 i ispisuje poruku. Ima li razlike između sljedeća dva kôda? Ako ima razlike, koji je kôd točniji i/ili kako popraviti neispravni dio?

  Kôd 1:

```
if (broj%5==0)
       printf("Broj je djeljiv s 5\n");
else if (broj%3==0)
       printf("Broj je djeljiv s 3\n");
else if (broj%2==0)
       printf("Broj je paran\n");
else
       printf("Broj nema djeljitelja manjeg od 7\n");
```

  Kôd 2:

```
if (broj%5==0)
       printf("Broj je djeljiv s 5\n");
if (broj%3==0)
       printf("Broj je djeljiv s 3\n");
if (broj%2==0)
       printf("Broj je paran\n");
if ( !( broj%5==0 && broj%3==0 && broj%2==0) )
       printf("Broj nema djeljitelja manjeg od 7\n");
```

2.4 Ugniježđene uvjetne kontrole

Uvjetne kontrole toka mogu biti i ugniježđene jedna unutar druge. To znači da je na mjestu *radnje* unutar kontrole toka postavljena još jedna kontrola toka. Na primjer, verzija programa za ispis ocjene:

```
if ( postotak>50 ) //Vanjska uvjetna kontrola
{
	printf("Cestitamo, polozili ste ispit!\nOcjena: ");
	if ( postotak > 85) //Unutarnja uvjetna kontrola
		printf("Izvrstan!\n");
	else if ( postotak > 70)
		printf("Vrlo dobar!\n");
	else
		printf("Dovoljno dobar.\n");
}
else
{
	printf("Potrebno je dopuniti znanje!\n");
}
```

Kontrola toka može biti ugniježđena i više puta.

PRIMJER 5 Napravi program koji za uneseni postotak na ispitu, ispisuje je li ispit položen ili nije. Ako je ispit položen, ispišite ostvarenu ocjenu. Ako nedostaju samo 2 boda za ocjenu više, ispišite dodatnu poruku.

Rješenje Na primjer, za kontrolu je li ocjena granična:

```
int main()
{
	int postotak;
	printf("Unesite ostvareni postotak: ");
	scanf_s("%d", &postotak);

	if (postotak > 50) //Vanjska uvjetna kontrola
	{
		printf("Cestitamo, polozili ste ispit!\nOcjena: ");
		if (postotak > 85) //Prva unutarnja uvjetna kontrola
			printf("Izvrstan!\n");
		else if (postotak > 70)
		{
			printf("Vrlo dobar!\n");
			if (postotak > 83) //Druga unutarnja uvjetna kontrola
				printf("Ali malo nedostaje za izvrstan!\n");
		}
		else
		{
			printf("Dovoljno dobar.\n");
			if (postotak > 68)
				printf("Ali malo nedostaje za vrlo dobar!\n");
		}
	}
	else
	{
		printf("Potrebno je dopuniti znanje!\n");
	}
	return 0;
}
```

Pokrenite program, i unesite ostvareni postotak na ispitu:

```
Unesite ostvareni postotak: 69
Cestitamo, polozili ste ispit!
Ocjena: Dovoljno dobar.
Ali malo nedostaje za vrlo dobar!
```

Uredno pisanje programa ***Dodatno***

Za lakše razumijevanje programa programski kôd treba biti uredan. Velik dio urednosti postiže se *tab* razmakom. Načelno, na svaku razinu ugniježđenosti dodaje se po jedan dodatni *tab* razmak. Na primjer, prethodni program za ispis ocjene može se jednako ispravno napisati ovako:

```
if ( postotak>50 ) //Vanjska_uvjetna_kontrola
                {
printf("Cestitamo, polozili ste ispit!\nOcjena: ");
       if ( postotak > 85) //Unutarnja_uvjetna_kontrola
       printf("Izvrstan!\n");
else if ( postotak > 70)
printf("Vrlo dobar!\n");
                else
printf("Dovoljno dobar.\n");       }
       else   {
printf("Potrebno je dopuniti znanje!\n"); }
```

Ovdje je potreban veliki napor za shvatiti koji dio kôda je ugniježđen i unutar koje kontrole se nalazi. Ako vezano za ovakav kôd upitate nekoga za pomoć (npr. na **ispitu**), moguća je neželjena reakcija.

Pitanja za provjeru znanja 2.3

- Dio programa treba ispisati parnost broja i predznak. Ima li razlike između sljedeća dva kôda? Ako ima razlike, koji je kôd točniji i/ili kako popraviti neispravni dio?

  Kôd 1:

```
if (A%2==0)
{
       printf("A je paran ");
       if (A>0)
              printf("i pozitivan ");
       else
              printf("i negativan ");
}
```

  Kôd 2:

```
if (A%2==0 && A>0)
       printf("A je paran i pozitivan");
if ( !(A%2==0 && A>0) )
       printf("A je paran i negativan");
```

- Napišite kontrolu toka koja ispisuje parnost broja (paran ili neparan) i predznak (pozitivan ili negativan).

2.5 Tipične pogreške

Točka-zarez iza naredbe *if* (npr. *if (n == 0);*) Potrebno je napraviti program koji provjerava je li broj jednak nuli, i ako je, tada ispisuje poruku. Program će ispod uvijek ispisati poruku *'Broj je jednak 0*', jer znak ';' znači kraj naredbe, tj. blok naredbi naredbe *if* nije očekivani blok *{ printf("Broj je jednak 0\n");}* nego je to prazan blok naredbi *{}*.

```
int main()
{
	int n;
	printf("Unesi broj: ");
	scanf_s("%d", &n)

	if (n == 0);
	{
		printf("Broj je jednak 0\n");
	}
	return 0;
}
```

Isti kôd kao i gore uvjetovan pogrešnim dodavanjem znaka ';' iza *if* naredbe *if (n == 0)*;

```
int main()
{
	int n;
	printf("Unesi broj: ");
	scanf_s("%d", &n);

	if (n == 0)
	{

	}
	{
		printf("Broj je jednak 0\n");
	}
	return 0;
}
```

Uspoređivanje vrijednosti s jednim znakom jednakosti '=' umjesto dva znaka '=='. Unutar uvjeta *if* naredbe, umjesto uspoređivanja vrijednosti s dva znaka jednakosti imamo pridruživanje s jednim znakom jednakosti. To pridruživanje *n* pretvori u 0, što dovodi do konstantnog uvjeta *if(0) {...}*. 0 kao argument naredbe *if* je *laž*, tj. uvjet nije ispunjen, a bilo koja druga vrijednost različita od 0 (1, -5, 3.14...) je *istina*.

```
int main()
{
	int n;
	printf("Unesi broj: ");
	scanf_s("%d", &n);

	if(n = 0)
	{
		printf("Broj je jednak 0\n");
	}
	return 0;
}
```

Switch case* operator** ***Dodatno

Za određene primjere, umjesto mnogih *elseif* funkcija, moguće je koristiti *switch case* operator koji pojednostavljuje pregled i pisanje kontrole toka. Operator se definira na sljedeći način:

```
switch (izraz)
{
case (konstanta1):
	// radnje
	break;
case (konstanta2):
	//radnje
	break;
	…
case (konstantaN):
	//radnje
	break;
}
```

case čija se *konstanta* podudara sa *switch* izrazom izvršava se pri čemu *break* operator osigurava da se samo traženi *case* izvrši. Također, *izraz* i *konstanta* moraju biti isti tip podatka radi uspoređivanja (*int, double, …*). Za odabir određenog *case,* aritmetički i relacijski operatori ne mogu se koristiti jer se isključivo traži konstanta određenog tipa podatka.

Primjer jednostavnog kalkulatora dvaju brojeva:

```
int main()
{
	float a, b, rezultat;
	int operacija;
	printf("Unesite brojeve a i b\n");
	scanf_s("%f %f", &a, &b);
	printf("Unesite sifru za zeljenu aritmeticku operaciju (1-zbrajanje, 2-oduzimanje, 3-
dijeljenje i 4-mnozenje)\n");
	scanf_s("%d", &operacija);

	switch (operacija)
	{
	case(1):
		rezultat = a + b;
		break;
	case(2):
		rezultat = a - b;
		break;
	case(3):
		rezultat = a / b;
		break;
	case(4):
		rezultat = a * b;
		break;
	}
	printf("Trazeni rezultat operacije %d je %f\n", operacija, rezultat);
	return 0;
}
```

2.6 Primjeri za vježbu

1. Napravi program za unos temperature u stupnjevima Celzija, program pretvara temperaturu u stupnjeve Kelvin i Fahrenheit (TF=9/5*TC+32 i TK=TC+273.15). Ako je unesena temperatura manja od apsolutne nule, ispiši da se radi o pogrešci i upitaj korisnika za ponovni unos.
Napomena: program se može riješiti na različite načine.

2⁺. Napiši program koji rješava kvadratnu jednadžbu $f(x)=ax^2 + bx + c$. Korisnik unosi koeficijente jednadžbe. Prije izračuna rješenja, potrebno je provjeriti postoji li realno rješenje.

3⁺. Napravi program za izračun naprezanja u gredi. Greda je čelična, poprečni presjek je 10x20. Korisnik unosi opterećenje [N/mm] i duljinu grede [mm]. Program računa i ispisuje naprezanje. Potrebno je ispisati poruku:
- "Naprezanje je dopusteno" ako je manje od 100 MPa,
- "Naprezanje je granicno" ako je između 100 i 120 MPa,
- "Naprezanje je na gornjoj granici" ako je između 120 i 130 MPa,
- "Naprezanje nije dopusteno" ako je vise od 130 MPa.

4. Sljedeći program treba usporediti vrijednosti dvaju brojeva te ispisati jesu li jednaki ili različiti. Program sadržava nekoliko pogrešaka, u svakoj liniji po barem jednu. Pokušajte ih pronaći.

```
#include<studio.h>
int main{
       int a,b,c,
       prinf("Unesi brojeve a i b:");
scanf_s("&a","&b",a b);
if (a = b);
{      prinf("Brojevi %a i %b su jednaki",a,b);  }
else;
{      prinf("Brojevi su razliciti",a,b);  }
return 0 }
```

5. Program za uneseni broj provjerava je li djeljiv s 3 ili 7, ali ne oboje.

6. Napravite program koji provjerava je li unesena točna lozinka. Ako je točna, program ispisuje „Točna lozinka“, a ako je pogrešna, program ispisuje „Pogrešna lozinka“. Za primjer uzmite da je točna lozinka 1234.

7. U programu prijavite tri cjelobrojne varijable (v1, v2 i v3). Korisnik unosi vrijednosti, a program treba pronaći i ispisati najveću.

8. Napravite program u koji se učitava 4 cijela broja (b1, b2, b3 i b4). Program treba ispisati koliko je brojeva pozitivnih i koliko je negativnih. Zatim je potrebno ispisati srednju vrijednost svih pozitivnih brojeva.

9. Napiši program u koji se unose 4 realna broja, a zatim se ispisuje razlika između najvećeg i najmanjeg.

10. Korisnik unosi tri cijela broja (b1, b2 i b3). Program provjerava jesu li dva od tri broja jednaka te ispisuje koji su to brojevi. Očekivani tijek programa:

```
Unesi tri broja: 3 5 3
Provjera: broj b1 je jednak broju b3
```

11. Napravi program koji računa površinu kruga, elipse, kvadrata ili pravokutnika. Korisnik na početku programa bira željeni geometrijski oblik, a zatim slijedi pripadni unos podataka i izračun.
Očekivani tijek programa:

```
Program računa površinu 1-kruga, 2-elipse 3-kvadrata ili 4-pravokutnika
Unesi željeni geometrijski lik: 4
Unesi dimenzije pravokutnika (a i b): 2 3.6
Površina iznosi: 7.2
```

2.7 Riješeni primjeri

Zadatci označeni znakom $^{+}$ su riješeni.

ZADATAK 2 Napiši program koji rješava kvadratnu jednadžbu $f(x)=ax^2 + bx + c$. Korisnik unosi koeficijente jednadžbe. Prije izračuna rješenja potrebno je provjeriti postoji li realno rješenje.

Rješenje je prikazano ispod.

```
#include<stdio.h>
#include<math.h>

int main()
{
	double a, b, c, x1, x2, D;

	printf("Koeficijent a: ");
	scanf_s("%lf", &a);
	printf("Koeficijent b: ");
	scanf_s("%lf", &b);
	printf("Koeficijent c: ");
	scanf_s("%lf", &c);
	D = b * b - 4 * a*c;

	if (fabs(D) < 1e-12 /*D == 0*/)
	{
		printf("X1=X2 = %.5lf\n", -b / (2 * a));
	}
	else if (D > 0)
	{
		x1 = (-b + sqrt(D)) / (2 * a);
		x2 = (-b - sqrt(D)) / (2 * a);
		printf("X1=%.5lf, x2=%.5lf\n", x1, x2);
	}
	else
	{
		printf("Nema realnih rjesenja!\n");
	}

	return 0;
}
```

ZADATAK 3 Proračun grednog nosača s kontrolom naprezanja (Laboratorijska vježba 2)
Napravi program za izračun naprezanja u gredi. Greda je čelična, poprečni presjek je 10x20. Korisnik unosi opterećenje [N/mm] i duljinu grede [mm]. Program računa naprezanje. Potrebno je ispisati poruku:
- "Naprezanje je dopusteno" ako je manje od 100 MPa,
- "Naprezanje je granicno" ako je između 100 i 120 MPa,
- "Naprezanje je na gornjoj granici" ako je između 120 i 130 MPa,
- "Naprezanje nije dopusteno" ako je vise od 130 MPa.

Rješenje je prikazano ispod.

```c
#include<stdio.h>

int main()
{
	float L, q, b, h, I, E, sigma;
	b = 10;
	h = 20;
	I = b * h*h*h / 12;
	E = 210000;
	printf("Unesi duljinu grede [mm]: ");
	scanf_s("%f", &L);
	printf("Unesi iznos opterecenja [N/mm]: ");
	scanf_s("%f", &q);
	sigma = h / 2 * q*L*L / (8 * I);
	printf("Naprezanje iznosi sigma=%f MPa\n", sigma);

	if (sigma<100.0)
	{
		printf("Naprezanje je dopusteno\n");
	}
	else if (sigma<120.0)
	{
		printf("Naprezanje je granicno\n");
	}
	else if (sigma<130.0)
	{
		printf("Naprezanje je na gornjoj granici\n");
	}
	else
	{
		printf("Naprezanje nije dopusteno\n");
	}

	return 0;
}
```

3 PETLJE

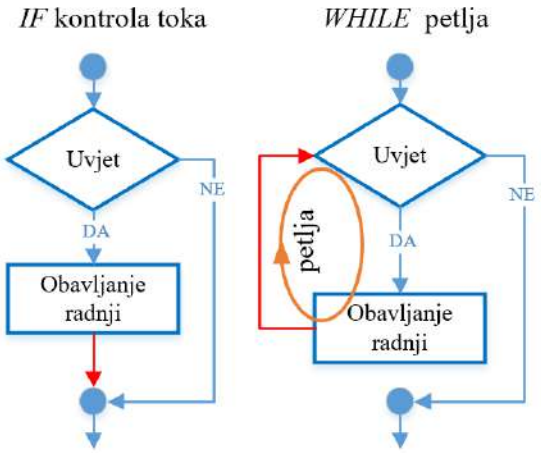
Petlje se koriste kada je potrebno jedan dio programa ponavljati više puta, odnosno sve dok je ispunjen zadani uvjet. Petlje se mogu podijeliti na dvije vrste: brojane petlje i uvjetne petlje. Brojane petlje (***for*** petlja) su one koje dio programa ponavljaju točno određen broj puta*. Uvjetna petlja (***while*** petlja) se koristi kada unaprijed nije poznato koliko puta neki dio programa treba ponavljati, već je samo poznat uvjet kada je potrebno ponoviti taj dio programa (ili zaustaviti ponavljanje). U programskom jeziku *C* petlje se mogu prikazati jedna preko druge, dok korisnik (programer) bira najprikladniju opciju.

3.1 Dok (*while*) petlja

While petlja ponavlja potrebne radnje sve dok je zadovoljen zadani uvjet. Opći zapis *while* petlje je:

```
while (uvjet)
{
        radnje;
}
```

Jednako kao i kod ***if*** kontrole, *radnje* je običaj pisati uvučeno u odnosu na ostatak kôda zbog bolje preglednosti. Uvjet može biti bilo koji logički izraz, isto kao i kod *if* kontrole toka. Također kao kod *if* petlje, radnje će se odvijati samo ako je uvjet ispunjen. Znači, ako uvjet prije ulaska u petlju nije ispunjen radnje se neće obaviti ni jedan put. U dijagramu toka, *while* petlja i *if* kontrola toka razlikuju se samo po smjeru strelice nakon radnji petlje.



PRIMJER 1 Napravi program koji inicijalizira vrijednost varijable *i* na 3. Varijablu zatim umanjuj sve dok je veća od nule.

Rješenje ovog jednostavnog primjera korištenja *while* petlje je u nastavku.

```
int i = 3;
while (i > 0)
{
        i = i - 1;
        printf("Varijabla 'i' trenutno iznosi: %d\n", i);
}
```

Rezultat programa je ispis brojeva od broja dva do nule:

* Najčešće se *for* petlja koristi kada se unaprijed zna koliko je puta potrebno nešto ponavljati. Međutim, postoje i druge mogućnosti, što je objašnjeno u zasebnom poglavlju o *for* petljama.

```
Varijabla 'i' trenutno iznosi: 2
Varijabla 'i' trenutno iznosi: 1
Varijabla 'i' trenutno iznosi: 0
```

PRIMJER 2 *While* petlje često se koriste kod provjere ispravnosti unosa podataka. U sljedećem programu unosi se temperatura u stupnjevima Celzija, program pretvara temperaturu u stupnjeve Kelvina i Fahrenheita. Ako je unesena temperatura manja od apsolutne nule, program pita korisnika za ispravak unosa.

Rješenje Korisnik prvo unosi temperaturu u °C. Sada je potrebno provjeriti je li unesena temperatura manja od apsolutne nule (-273.15°C). Ako je manja, ispisuje se poruka i ponovno ide unos sa *scanf_s*. Za razliku od obične kontrole toka *if*, ovdje će se kontinuirano provjeravati ispravnost unosa, sve dok se ne unese ispravna temperatura.

```
int main()
{
	double tC,tF,tK;
	printf("Unesi temperaturu u stupnjevima Celzija: ");
	scanf_s("%lf", &tC);
	while (tC <= -273.15)
	{
		printf("Greska u unosu, unesite fizikalnu temperaturu: ");
		scanf_s("%lf", &tC);
	}
	tF = 9/ 5*tC + 32;
	tK = 273.15 + tC;
	printf("Unesena temperatura (%lf C) iznosi %lf F odnosno %lf K \n",tC,tF,tK);
	return 0;
}
```

Test programa:

```
Unesi temperaturu u stupnjevima Celzijusa: -400
Greska u unosu unesite fizikalnu temperaturu: -300
Greska u unosu, unesite fizikalnu temperaturu: -200
Unesena temperatura (-200.000000 C) iznosi -168.000000 F odnosno 73.150000 K
```

3.2 Petlja: *do-while*

Druga verzija *while* petlje je ***do while*** petlja. Razlika je u tome što se kod *do while* prvo obavljaju *radnje*, a tek zatim se provjerava ispunjavanje *uvjeta*. Opći zapis *do while* petlje je:

```
do
{
        radnje;
} while (uvjet);
```

USPOREDBA *DO-WHILE* I *WHILE* Za usporedbu s *while*, pogledajte rezultate sljedeća dva testa. U prvome, koristi se *do-while* petlja, u kojoj se prvo vrši ispis poruke, a zatim se provjerava uvjet *i==0*, koji nije istinit jer je prije petlje inicijalizirana vrijednost *i=1*.

```
int i = 1;
do
{
        printf("Petlja je pokrenuta\n");
} while (i == 0);
printf("Kraj programa!\n");
```

Ispis sada sadržava obje poruke:

```
Petlja je pokrenuta
Kraj programa!
```

U drugome, koristi se *while* petlja:

```
int i = 1;
while (i == 0)
{
        printf("Petlja je pokrenuta\n");
}
printf("Kraj programa!\n");
```

Sada se u ispisu neće pojaviti ispis koji se nalazi unutar petlje:

```
Kraj programa!
```

Pitanja za provjeru znanja 3.1

- Što će program raditi u prethodna dva slučaja ako je na početku varijabla *i* inicijalizirana na nulu (`int i = 0;`)?

PRIMJER 3 Potrebno je napraviti program koji provjerava predznak brojeva sve dok korisnik ne unese broj nula.

Rješenje Program se sastoji od jedne *do-while* petlje, unutar koje se nalazi unos i provjera predznaka:

```
int broj;
do
{
        printf("Unesite broj: ");
        scanf_s("%d", &broj);
        if (broj > 0)
                printf("Broj je pozitivan\n");
        if (broj < 0)
                printf("Broj je negativan\n");
} while (broj!=0);
```

Test programa:

```
Unesite broj: 5
Broj je pozitivan
Unesite broj: -5
Broj je negativan
Unesite broj: 0
Press any key to close this window . . .
```

Nakon što je unesena nula, program završava.

Pitanja za provjeru znanja 3.2

- Prethodni primjer koji koristi *do-while* petlju, na što sve treba paziti ako se umjesto *do while* koristi *while* petlja?

Prekidanje rada programa kod beskonačnih petlji ***Dodatno***

Kod rada s petljama često se može dogoditi da program „zapne“ unutar petlje. Ako je program zauzet dulje vrijeme, može se ručno obustaviti rad programa pomoću kratice na tipkovnici *ctrl+c*. Na primjer, napravite program:

```
while(1)
	printf("Zapelo je!\n");
```

Ova se petlja ponavlja beskonačno dugo jer je uvjet petlje uvijek istina (1). Sada za prekinuti rad programa pritisnite zajedno *ctrl+c*, dok vam je označen komandni prozor (ako nije označen, potrebno je mišem prvo kliknuti bilo gdje unutar prostora komandnog prozora).

3.3 Petlja: *for*

EKVIVALENTNA *WHILE* PETLJA Korištenjem *while* petlje ispisat će se brojevi od 1 do 4. Varijabla *brojač* se inicijalizira na 1, a zatim se korištenjem *while* petlje *brojač* uvećava dok je manji od broja 5.

```
int brojac;
brojac = 1;
while(brojac<5)
{
	printf("Brojac=%d\n",brojac);
	brojac = brojac + 1;
}
```

Ispis:

```
Brojac=1
Brojac=2
Brojac=3
Brojac=4
```

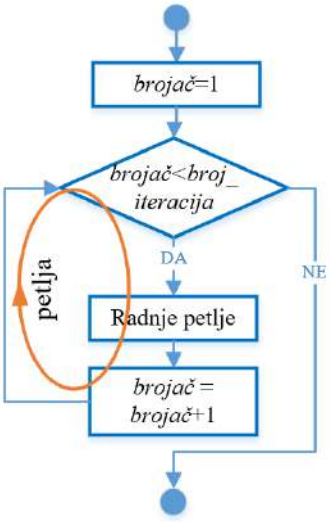
FOR* PETLJA** Prethodni primjer sadržava sve dijelove kao i ***for petlja. Sastavni dijelovi *for* petlje su *brojač*, inicijalizacija, uvjet ponavljanja i „uvećavanje“ *brojača*. Kada je potrebna petlja koja ima sve ove dijelove, umjesto *while* petlje, jednostavnije je koristiti *for* petlju, općeg oblika:

```
for ( inicijalizacija ; uvjetPonavljanja ; modifikacijaVarijable )
{
        radnje;
}
```

Inicijalizacija je jedna ili više naredbi koje se izvršavaju jednokratno prije petlje, a *modifikacija varijable* je naredba koja se izvršava svaki put nakon što su sve radnje petlje obavljene. U općem slučaju, inicijalizacija i modificiranje varijable mogu biti bilo koji izraz. Međutim, *for* petlja se najčešće koristi u slučaju kada se neki brojač inicijalizira na 1, a zatim se radnje obavljaju točno određeni broj puta dok brojač ne dostigne željeni broj iteracija. Prethodni primjer sada je riješen korištenjem *for* petlje:

```
int brojac;
for(brojac = 1 ; brojac<5 ; brojac = brojac + 1)
{
        printf("Brojac=%d\n",brojac);
}
```

Dijagram toka za *for* petlju prikazan je na sljedećoj slici.



MODIFIKACIJA VARIJABLE Uvećavanje varijable petlje ne treba uvijek biti za 1. Varijabla se može modificirati proizvoljnom operacijom (umanjivati, množiti…). Primjer, uvećavanje za pet:

```
for(brojac = 0 ; brojac<15 ; brojac = brojac + 5)
{
        printf("Brojac=%d\n",brojac);
}
```

Ispis:

```
Brojac=0
Brojac=5
Brojac=10
Brojac=15
```

Pitanja za provjeru znanja 3.3

- Prethodni primjer koristi *for* petlju. Može li se isto napraviti korištenjem *do-while* petlje?

JEDNOSTAVNO PONAVLJANJE Varijabla petlje uopće se ne treba koristiti unutar petlje, na primjer *for* petlje mogu se koristiti ako želimo neku radnju ponoviti točno određen broj puta:

```
for (brojac = 1; brojac <= 144; brojac = brojac + 1)
{
	printf("around the world, ");
}
```

Ispis:

```
around the world, around the world, around the world, ...
```

Višestruka inicijalizacija ***Dodatno***

Petlja *for* može se koristiti i s višestrukom inicijalizacijom, gdje se svaka inicijalizirana varijabla odvaja zarezom. Također se uvećavanje varijabli može zasebno raditi za obje varijable odvojeno zarezom.

```
for (i = 1, j = 2; i =< 4 && j <= 10; i++, j=j+2)
	printf("%d, %d\n", i, j);
```

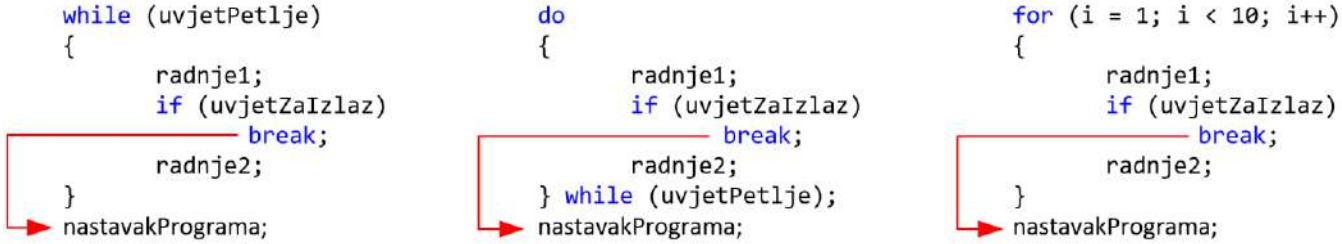
Rezultat:

```
1, 2
2, 4
3, 6
4, 8
```

Obratite pažnju da je uvjet ponavljanja još uvijek samo jedan, gdje su dva relacijska operatora $i \leq 4$ i $j \leq 10$ povezani s logičkim operatorom && (*i*). U prethodnom primjeru, petlja se ponavljala dok su oba uvjeta zadovoljena, što znači da se petlja zaustavlja čim jedan od njih nije zadovoljen. Ako želimo da se petlja ponavlja dok je barem jedan uvjet zadovoljen, koristi se logički operator || (*ili*).

3.4 Prekidanje i nastavljanje petlje

PREKIDANJE PETLJE Za petlje *for*, *while* i *do-while* postoji opcija za izlaz bez obzira na zadovoljavanje uvjeta petlje. Ovo se radi naredbom ***break***. Tijek programa za sva tri slučaja:



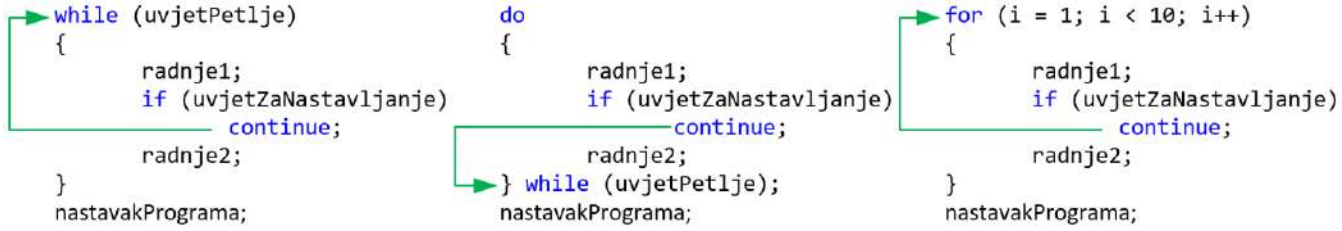
Kada se postigne *uvjetZaIzlaz*, program odmah izlazi iz petlje. U ugniježdenim petljama, naredba *break* izlazi samo iz najbliže petlje.

PRIMJER 4 Korisnik unosi broj po broj, sve dok ne unese 0.

Rješenje Primjer je riješen korištenjem *while(1)* petlje. Ova bi se petlja beskonačno ponavljala, ali dodana je naredba za prekid petlje unutar *if* kontrole. Ako je uneseni broj jednak nuli, program dolazi do *break* naredbe, te izlazi iz petlje. (Da bi se *while* petlja ponavljala, važno je da *uvjet* ima konstantnu vrijednost različitu od 0. Vrijednost 0 je *laž*, tj. uvjet nije ispunjen, a bilo koja druga vrijednost različita od 0 (1, -5, 3.14...) je *istina*.)

```
#include<stdio.h>

int main()
{
	int n;

	while (1)/* Konstantni uvjet (1). Može i (1 > 0), (true),..., (bilo
koja vrijednost ili izraz čiji je rezultat različit od 0)*/
	{
		printf("Unesi broj: ");
		scanf_s("%d", &n);

		if (n == 0)
			break; // Izlaz iz while petlje
	}
	return 0;
}
```

NASTAVLJANJE PETLJE Naredba srodna naredbi *break* je naredba ***continue***. Također se može koristiti kod bilo kojeg tipa petlje, a kada program dođe do *continue*, to označava preskakanje ostatka naredbi u petlji (*radnje2*) te vraćanje na provjeru uvjeta petlje. U općem obliku, *continue* se koristi slično kao i *break*:



PRIMJER 5 Napravite program koji ispisuje brojeve redom od 1 do 10, ali preskače parne brojeve. **Rješenje** je u nastavku.

```
int i;
for (i = 1; i <= 10; i++)
{
     if (i % 2 == 0)
     {
          continue;
     }
     printf("%d ", i);
}
```

Ispis:

```
1 3 5 7 9
```

Petlja varira vrijednost *i* od 1 do 10, a u petlji se nalazi naredba za ispis brojeva. Prije naredbe za ispis broja nalazi se provjera je li trenutni broj paran. Ako je broj paran, program prelazi na sljedeći element petlje.

Pitanja za provjeru znanja 3.4

- Kodovi u nastavku trebali bi ispisivati cijele brojeve redom, dok im je suma manja od 100. Ima li razlike između sljedeća dva kôda? Ako ima razlike, koji kôd je točniji i/ili kako popraviti neispravni dio?

  Kôd 1:

```
sum=0;
printf("Brojevi cija je suma<100: ", i);
for (i = 1; i <= 100; i++)
{
       sum=sum+i;
       if (sum>100)
              break;
       printf("%d ", i);
}
```

  Kôd 2:

```
i=0;
sum=0;
printf("Brojevi cija je suma<100: ", i);
do
{
       sum=sum+i;
       printf("%d ", i);
}while (sum>100);
```

3.5 Ugniježđene petlje

Ako se unutar petlje nalazi i druga (unutarnja) petlja, tada je riječ o ugniježđenoj petlji. Primjer:

```
for (i = 1; i < 10; i++) //Vanjska petlja
{
        radnje1;
        for (j = 1; j < 10; j++) //Unutarnja petlja
        {
                radnje2;
        }
}
```

U ovakvim petljama, za svaki od brojeva u prvom rasponu brojeva, unutarnja *for* petlja će izvršiti *radnje2* za cijeli drugi raspon brojeva.

Primjer koji pokazuje redoslijed radnji ugniježđene petlje:

```
int main()
{
    int i, j;
    for (i = 1; i <= 3; i++)
    {
        printf("o");
        for (j = 1; j <= 4; j++)
        {
            printf("x");
        }
        printf(".");
    }
       return 0;
}
```

Pokretanjem programa dobije se rezultat:

```
oxxxx.oxxxx.oxxxx.
```

Program dakle ide sljedećim redoslijedom. Prvi put ulazi u vanjsku petlju, a zatim ispisuje jedno slovo *o*. Zatim program dolazi do unutarnje petlje koja ponavlja svoje radnje četiri puta, odnosno ispisuje četiri slova *x*. Program zatim izlazi iz unutarnje petlje u vanjsku i ispisuje ostatak radnji u prvoj iteraciji odnosno ispisuje točku. Zatim slijedi drugo ponavljanje vanjske petlje, i tako dalje.

Primjer gdje se unutarnja petlja mijenja za vrijeme iteracija u vanjskoj petlji:

```
for (i = 1; i <= 5; i++)
{
    printf("o");
    for (j = 1; j <= i; j++)
    {
        printf("x");
    }
    printf(".");
}
```

Rezultat programa:

```
ox.oxx.oxxx.oxxxx.oxxxxx.
```

U svakoj iteraciji vanjske petlje mijenja se raspon brojeva unutarnje petlje. U prvoj iteraciji vanjske petlje raspon je jednak samo jednom broju (*i=1*, slijedi *j=1 do 1*). Za vrijeme druge iteracije raspon brojeva unutarnje petlje sadržava dva broja (*i=2*, slijedi *j=1 do 2*), itd. Ovo se vidi po ispisu *x*-eva u rezultatu programa. Prvi je put ispisano jedno slovo *x*, a zatim je u svakoj iteraciji vanjske petlje ispisano po jedno *x* slovo više.

Pitanja za provjeru znanja 3.5

- Kako će izgledati ispis ako se na prethodni primjer nadoda unutar petlje jedna linija s ispisom novog retka:

```
for (i = 1; i <= 5; i++)
{
	printf("o");
	for (j = 1; j <= i; j++)
	{
		printf("x");
	}
	printf(".");
	printf("\n");
}
```

- Koliko iznose *sum1*, *sum2*, *sum3* i *sum4* u sljedećem primjeru (pokušajte izračunati bez pokretanja programa):

```
sum1=0;
sum2=0;
sum3=0;
sum4=0;
for (i = 1; i <= 3; i++)
{
	for (j = 1; j <= 4; j++)
	{
		sum1=sum1+i;
		sum2=sum2+j;
	}
	sum3=sum3+i;
	sum4=sum4+j;
}
```

- Koliko iznosi *sum2* u sljedećem primjeru:

```
sum2=0;
for (i = 1; i <= 5; i++)
{
	for (j = 1; j <= i; j++)
	{
		sum2=sum2+j;
	}
}
```

3.6 Inženjerski primjeri

PRIMJER 6 Pretvaranje temperature s kontrolom (Laboratorijska vježba 3)

Napravi program za pretvorbu temperature iz °C u K ili °F. Korisnik za vrijeme izvođenja programa odabire koju pretvorbu želi (F ili K). Potrebno je osigurati fizikalan unos temperature tako da se temperatura ispravlja sve dok nije unesena ispravna.

Rješenje je prikazano ispod.

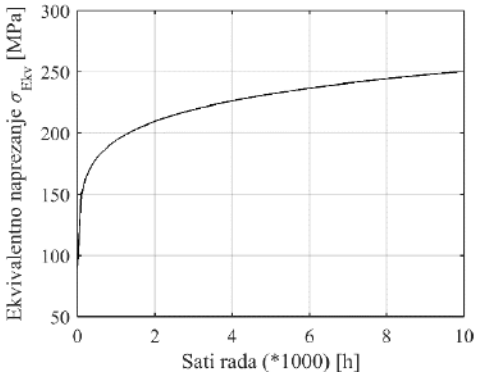
```
#include <stdio.h>

int main()
{
	float tempC, tempPretvorena;
	char odabir;

	printf("Unesi temperaturu [°C]: ");
	scanf_s("%f", &tempC);
	while (tempC<-273.15)
	{
		printf("Nefizikalna temperature, ponovno unesite: ");
		scanf_s("%f", &tempC);
	}

	printf("Odaberi pretvorbu: F ili K: ");
	scanf_s(" %c", &odabir);
	if (odabir == 'F')
		tempPretvorena = 9.0 / 5.0*tempC + 32.0;
	else if (odabir == 'K')
		tempPretvorena = 273.15 + tempC;
	else
		printf("Neispravan unos!\n");

	printf("Pretvorena temperatura iznosi: %f\n", tempPretvorena);

	return 0;
}
```

PRIMJER 7 Potrebno je za opterećenu gredu odrediti vremensku statičku čvrstoću, odnosno vrijeme kada ekvivalentno stacionarno dugotrajno naprezanje prelazi određenu granicu. Materijal grede je konstrukcijski čelik, za čvrstoću uzmite granicu tečenja od 225 MPa, a statičko naprezanje dobije se po proračunu iz prethodno riješenog programa (na primjer 88 MPa).

Rješenje Ekvivalentno stacionarno dugotrajno naprezanje računa se po izrazu [11]:

$$\sigma_{\text{Ekv}} = \left(t \cdot \sigma^{m_d} \right)^{\frac{1}{m_d}} \tag{3.1}$$

gdje je t vrijeme djelovanja stacionarnog naprezanja u satima, σ je statičko naprezanje, a m_d je eksponent krivulje dugotrajne statičke čvrstoće. Za ovaj primjer odabrana je vrijednost eksponenta m_d=9. Grafički prikaz izraza za ekvivalentno stacionarno naprezanje prikazan je na donjoj slici.



Slika 3.1. Ekvivalentno stacionarno naprezanje

Dakle, od nekog početnog naprezanja, kroz vrijeme raste iznos ekvivalentnog statičkog naprezanja. Nakon što ova ekvivalentna vrijednost prijeđe vrijednost statičke čvrstoće materijala, dolazi do pucanja. Prvi dio programa je napraviti unos statičnog naprezanja, statičke čvrstoće i eksponenta. Zatim je potrebno napraviti unos vremenskog koraka za analizu. Vrijeme *t* će se za prethodni izraz mijenjati unutar petlje sve dok je naprezanje manje od dopuštenog, nakon čega se naredbom *break* izlazi iz petlje.

```
int main()
{
	//Program za izračun dugotrajne čvrstoće konstrukcije
	double sigmaEkv,staticnoNaprezanje = 88;//Statičko naprezanje[MPa]
	double dopustenoNaprezanje = 225;// Statička čvrstoća materijala[MPa]
	double md = 9;// Eksponent krivulje dugotrajne statičke čvrstoće
	double t=0, deltaT = 100;// Vremenski korak u proracunu [h]

	// Proračun ekvivalentne čvrstoće kroz vrijeme
	printf("Vrijeme [h]\tEkvivalentno naprezanje [MPa]\n");
	while(1)
	{
		t = t + deltaT;
		sigmaEkv = pow(t * pow(staticnoNaprezanje,md), (1 / md));
		if (sigmaEkv > dopustenoNaprezanje)
		{
			printf("Nakon %.0lf h naprezanje prelazi dopusteno", t);
			break;
		}
		printf("%.0lf\t\t%lf\n", t, sigmaEkv);
	}
	return 0;
}
```

Rezultat programa prikazan je u nastavku. Program pokazuje „tablični“ prikaz rezultata:

```
Vrijeme [h]     Ekvivalentno naprezanje [MPa]
100             146.792847
200             158.545044
300             165.851104
...
4600            224.623797
Nakon 4700 h, naprezanje prelazi dopusteno
```

PRIMJER 8 Korisnik unosi glavnicu, broj obračunskih razdoblja i kamatnu stopu. Program računa iznos mjesečne kreditne rate korištenjem konformnog kamatnog faktora.

Rješenje Mjesečne kreditne rate s konformnim kamatnim faktorom r [12] računaju se pomoću:

$$r = (1 + p/100)^{1/m}$$

gdje je p kamatna stopa u postotku, a m broj godišnjih rata. Mjesečna kamata računa se po izrazu:

$$M = \frac{C \cdot (r-1)}{1 - r^{-n}}$$

gdje je C iznos glavnice (kredita) a n broj obračunskih razdoblja (n=*brojGodinaIsplate*·m). Program na osnovi unosa računa mjesečnu kamatu, a zatim ispisuje po obračunskim razdobljima koliko iznosi otplatni obrok i koliko je još ukupno preostalo novca za otplatu. Program je prikazan u nastavku.

```
int main()
{
    double C, p,M,r, preostaloKredita;
    int i,god,m,n,preostaloRazdoblja;
    printf("Unesi iznos kredita u kunama: ");    scanf_s("%lf", &C);
    printf("Unesi kamatnu stopu u postotku: ");    scanf_s("%lf", &p);
    printf("Unesi broj godina isplate: ");    scanf_s("%d", &god);
    m = 12;// Uplata se vrši 12 puta dogišnje
    n = god * m;// Broj obračunskih razdoblja
    r = pow(1 + p / 100,(double)1 / m);// Konformni kamatni faktor
    // neke banke računaju prema izrazu r = p / (m * 100) (direktna metoda)
    M = C * ( (r - 1) / (1 - pow(r,-n)) ); // Mjesečni otplatni obrok
    printf("Mjesečna rata iznosi %.2lf [kn]\n\n", M);

    preostaloRazdoblja = n;// Preostalo mjeseci za isplatu
    preostaloKredita = M * n;
    for (i = 0; i < n; i++)
    {
        preostaloRazdoblja = preostaloRazdoblja - 1;
        preostaloKredita = preostaloKredita - M;
        printf("Red. br. %d: Otplatni obrok iznosi %.2f kn,\n", i, M);
        printf("Preostalo je %d isplata i ukupno %.2f kn\n\n",
            preostaloRazdoblja, preostaloKredita);
    }
       return 0;
}
```

U testu programa uneseno je 10 tisuća kuna s kamatnom stopom od 7% na 5 godina isplate. Mjesečna rata tada iznosi 197.00 kn, a ukupno je potrebno isplatiti 11820 kn:

```
Unesi iznos kredita u kunama: 10000
Unesi kamatnu stopu u postotku: 7
Unesi broj godina: 5
Mjesečna rata iznosi 197.00 [kn]

Red. br. 1: Otplatni obrok iznosi 197.00 kn,
Preostalo je 59 isplata i ukupno 11622.94 kn
Red. br. 2: Otplatni obrok iznosi 197.00 kn,
Preostalo je 58 isplata i ukupno 11425.95 kn
…
```

3.7 Zadaci za vježbu

1⁺. Korisnik unosi broj redaka i stupaca, a program treba ispisati matricu te veličine, gdje su elementi na okviru matrice jednaki jedinici, a u unutrašnjosti jednaki nuli. Na primjer, ako korisnik unese 4 retka i 5 stupaca, program treba generirati ispis:

```
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
```

2⁺. Računalo generira slučajan broj između 1 i 128 koji korisnik treba pogoditi. U trenutku kad korisnik pogodi broj, program ispiše ukupan broj pogađanja. Napomena, za zadani interval [1, 128] metodom raspolavljanja broj sigurno pogađamo u 7 pokušaja (2^7 = 128).

3. Napišite program koji će ispisati uzorak zvjezdica (*) ili brojeva po zadanom pravilu u nastavku. Broj redaka određuje korisnik programa.

a) Trokut sa zvjezdicama:

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

Očekivani tijek programa za primjer a), a slično je i za ostale:

```
Unesi broj redaka: 3
*
**
***
```

b) Trokut s ponavljajućim rastućim brojevima po retku:

```
1
12
123
```

c) Trokut s ponavljajućim brojevima po retku:

```
1
22
333
```

d) Trokut s rastućim brojevima, s razmakom:

```
 1
 2 3
 4 5 6
```

e) Piramida sa zvjezdicama:

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

f) Romb sa zvjezdicama:

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

g∗) Paskalov trokut:

```
      1
    1   1
  1   2   1
1   3   3   1
```

4. Napravite program koji ispisuje tablicu množenja brojeva od 1 do 10.

5. Korisnik unosi donju i donju granicu (cijeli brojevi). U zadanom rasponu, potrebno je ispisati sve neparne brojeve te izračunati njihovu sumu.

6. Korisnik unosi donju i donju granicu (cijeli brojevi). U zadanom rasponu, potrebno je izračunati umnožak svih brojeva koji nisu djeljivi s 5.

7. Potrebno je ispisati sve brojeve od 1 do 100 koji su dijeljivi s 5 ili 7. Ispis treba biti izveden na način da se ispisuje po 5 brojeva u svakom retku. Očekivani tijek programa:

```
Brojevi koji su djeljivi s 5 ili 7 su:
5 7 10 14 15
20 21 25 28 30
...
```

8. Napišite program koji računa broj π pomoću beskonačne sume:

$$\pi = 4 \cdot \left(\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - ... \right)$$

Program izradi pomoću *while* petlje koja se ponavlja dok nije zadovoljena točnost koju unosi korisnik. Točnost podrazumijeva promjenu u aproksimaciji broja π u dva uzastopna koraka (*točnost = abs(piAproks(i)-piAproks(i-1))*).

9. Korisnik unosi koordinate središta elipse (*xS*,*yS*), te vrijednosti poluosi (*a* i *b*). Program zatim traži unoskoordinata točaka (*xT*,*yT*) te ispisuje nalazi li se točka unutar, izvan ili na elipsi. Unos točaka se ponavlja se sve dok korisnik ne pogodi točku koja se nalazi na elipsi.
Napomena: Jednadžba elipse je $((x-xS)/a)^2+((y-yS)/b)^2=1$

10. Korisnik unosi proizvoljan cijeli broj. Program treba ispisati koliko znamenaka broj sadržava.

11. Korisnik unosi proizvoljan cijeli broj. Program treba broj ispisati naopako (ako je broj 2367, program treba ispisati 7632).

12. Korisnik unosi proizvoljan cijeli broj. Program treba ispisati sve njegove djelitelje.

13. Napiši program koji provjerava je li uneseni broj prost (djeljiv samo s brojem 1 i samim sobom).

14. Napišite program koji simulira igru ruleta. Korisnik prvo unosi početno stanje računa. Zatim unosi iznos pojedinog *uloga* koji umanjuje stanje računa, te željeni broj od 0 do 36 ili boju (znak 'r' za crvenu, 'b' za crnu). Program zatim generira slučajni broj između 0 i 36, gdje su pozitivni parni brojevi crni, a pozitivni neparni brojevi crveni. Ako je korisnik pogodio broj, stanje računa povećava se za *ulog**35, a ako pogodi boju za *ulog**2.

15. Napišite program koji ispisuje sve proste brojeve između 1 i 1000.

16. Napišite program koji provjerava može li se uneseni broj izraziti kao zbroj dvaju prostih brojeva.
Očekivani tijek programa:

```
Unesi broj za provjeru: 16
Rezultati provjere:
16 = 3 + 13
16 = 5 + 11
```

3.8 Riješeni primjeri

Zadaci označeni znakom + riješeni su u nastavku.

ZADATAK 1 Ispis nul-matrice s jedinicama na okviru (Laboratorijska vježba 3)

Korisnik unosi broj redaka i stupaca. Program treba ispisati matricu te veličine gdje su elementi na okviru matrice jednaki jedinici, a u unutrašnjosti jednaki nuli. Na primjer, ako korisnik unese 4 retka i 5 stupaca, program treba generirati ispis:

```
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
```

Rješenje Ovo se rješava s ugniježđenom *for* petljom.

```
int main()
{
	int brojRedaka, brojStupaca;
	int i, j;
	printf("Unesite broj redaka: ");
	scanf_s("%d", &brojRedaka);
	printf("Unesite broj stupaca: ");
	scanf_s("%d", &brojStupaca);

	for (i = 1; i <= brojRedaka; i++)
	{
		for (j = 1; j <= brojStupaca; j++)
		{
			if (i == 1 || i == brojRedaka || j == 1 || j == brojStupaca)
				printf("1 ");
			else
				printf("0 ");

		}
		printf("\n");
	}

	return 0;
}
```

Test programa:

```
Unesite broj redaka: 5
Unesite broj stupaca: 7
1 1 1 1 1 1 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 1 1 1 1 1 1
```

ZADATAK 2 Pogađanje broja metodom polovljenja (Laboratorijska vježba 3)
Računalo generira slučajan broj između 1 i 128 koji korisnik treba pogoditi. U trenutku kad korisnik pogodi broj, program ispiše ukupan broj pogađanja. Napomena, za zadani interval [1, 128] metodom raspolavljanja broj sigurno pogađamo u 7 pokušaja (2^7 = 128).

Rješenje je prikazano ispod.

```
#include<stdio.h>
#include <stdlib.h>
#include<time.h>

int RandUGranicama(int donjaGranica, int gornjaGranica)
{
       return donjaGranica + rand() % (1 + gornjaGranica - donjaGranica);
}

int main()
{
    int slucajanBroj,broj,brojPogadjanja=1;
    srand(time(0));
    slucajanBroj = RandUGranicama(1,128);
    printf("Unesi broj izmedu 1 i 128: ");
    scanf_s("%d", &broj);

    while (1)
    {
        if (broj < slucajanBroj)
        {
            printf("Pokusaj s vecim brojem: ");
            scanf_s("%d", &broj);
        }
        else if (broj > slucajanBroj)
        {
            printf("Pokusaj s manjim brojem: ");
            scanf_s("%d", &broj);
        }
        else
        {
            printf("Pogoden u %d pokusaja!\n", brojPogadjanja);
            if (brojPogadjanja > 7)
                printf("\n Sramota!\n");
            break;
        }
        brojPogadjanja = brojPogadjanja + 1;
    }
       return 0;
}
```

4 POLJA

4.1 Jednodimenzionalna polja

Kada je u programu potrebno pohraniti veći skup podataka, potrebno je koristiti polja. Na primjer, ako mjerimo masu za 10 uzoraka te želimo pohraniti sva mjerenja u svrhu naknadne obrade (npr. izračun srednje mase, standardne devijacije,…), mogli bismo inicijalizirati 10 varijabli:

```
int masa1, masa2, masa3, masa4, masa5, masa6, masa7, masa8, masa9, masa10;
```

Umjesto ovoga, jednostavnije je prijaviti polje brojeva:

```
int masa[10];
```

gdje pojedine članove polja pozivamo s *masa[0]*, *masa[1]*, … , *masa[9]*. Polja se obično vizualiziraju kao redak tablice u kojem svaka kućica ima svoj naziv, te može sadržavati odgovarajuću (npr. *int*) vrijednost:

Naziv:	p[0]	p[1]	p[2]	p[3]	p[4]	p[5]
Vrijednost:	106	6	51	34	65	7

Članovi polja mogu se koristiti jednako kao i obične varijable, na primjer za unos članova polja:

```
printf("Unesite clan polje[0]: ");
scanf_s("%d", &polje[0]);
printf("Unesite clan polje[1]: ");
scanf_s("%d", &polje[1])
```

PRIMJER 1 Potrebno je izračunati srednju masu iz tri mjerenja.

Rješenje Kod korištenja običnih varijabli, svaka varijabla unosi se zasebno:

```
int main()
{
    double masa1, masa2, masa3;
    printf("Unesite prvu masu (kg): ");
    scanf_s("%lf", &masa1);
    printf("Unesite drugu masu (kg): ");
    scanf_s("%lf", &masa2);
    printf("Unesite treci masu (kg): ");
    scanf_s("%lf", &masa3);
    printf("Srednja masa: %lf kg", (masa1 + masa2 + masa3) / 3);
    return 0;
}
```

Za razliku od korištenja običnih varijabli, indeks polja može se programibilno mijenjati, a samim tim unos cijelog polja može se jednostavno izvesti jednom *for* petljom:

```
int main()
{
    int i;
    double mase[3];
    for (i = 0; i < 3; i++)
    {
        printf("Unesite %d. masu (kg): ", i + 1);
        scanf_s("%lf", &mase[i]);
    }
    printf("Srednja masa: %lf kg", (mase[0] + mase[1] + mase[2]) / 3);
    return 0;
}
```

INICIJALIZACIJA POLJA Polje se može inicijalizirati na nekoliko različitih načina. Prvi je način prikazan ranije, kada se samo rezervira polje željene veličine (npr. *int polje[10]*). Kod ovog načina potrebno je paziti da inicijalizirane varijable nemaju definiranu vrijednost, već će poprimiti vrijednost onoga što se nalazilo na djeliću memorije koji je sada rezerviran za polje (tzv. smeće). Ako se polje deklarira s *polje[10]={};*, sve će vrijednosti biti inicijalizirane na nulu. Polju se mogu i izravno pridijeliti željene vrijednosti, npr.:

```
int polje[5]={4, 3, 2, 1, 0};
```

Treći je način pridjeljivanje vrijednosti bez da se odredi veličina polja. U tom će se slučaju automatski inicijalizirati polje potrebne veličine. Na primjer, sljedeće polje biti će jednako prethodnom:

```
int polje[]={4, 3, 2, 1, 0};
```

PRIMJER 2 Aritmetička sredina i broj *velikih* (broj većih brojeva od a.s.) (Laboratorijska vježba 4). Korisnik unosi proizvoljan broj izmjerenih masa. Program treba pronaći aritmetičku sredinu te izbrojiti koliko je elemenata većih od aritmetičke sredine.

Rješenje Program je riješen na način da korisnik unosi proizvoljan broj izmjera. Kada se unese izmjera koja je jednaka nuli, program izlazi iz petlje (pomoću *break*). Kada program izađe iz *for* petlje, brojač *i* pamti indeks zadnjeg unesenog elementa polja. Međutim, broj elemenata je jednak *i+1*, jer indeksi kreću od nule. Kod izračuna aritmetičke sredine potrebno je prvo definirati *sumu* koja treba biti inicijalizira na nulu. Zatim se aritmetička sredina računa kao omjer sume i broja elemenata.

```
int main()
{
    int i=0,brIzmjera=0,brVelikih=0; //Vazno je inicijalizirati na nule
    double masa[100],suma=0.0,arSred;
    //Unos izmjera, sve dok se ne unese nula:
    for(i=0;i<100;i++)
    {
        printf("Unesi izmjeru  %d. (unesi 0 za kraj) [kg]:", i+1);
        scanf_s("%lf", &masa[i]);
        if (masa[i] <= 0)
            break;
    }
    brIzmjera = i + 1; //Broj izmjera je "i+1", jer brojac "i" ide od nule

    //Izracun aritmeticke sredine:
    for (i = 0; i < brIzmjera; i++)
        suma = suma + masa[i];
    arSred = suma / brIzmjera;

    //Izbroji koliko ima vecih od aritmeticke sredine:
    for (i = 0; i < brIzmjera; i++)
        if (masa[i] > arSred)
            brVelikih++;

    //Ispis rezultata
    printf("Srednja masa je %lf kg\n", arSred);
    printf("Velike mase: %d (od %d)", brVelikih, brIzmjera);

    return 0;
}
```

Test programa:

```
Unesi izmjeru  1. (unesi 0 za kraj) [kg]:12.3
Unesi izmjeru  2. (unesi 0 za kraj) [kg]:11.53
Unesi izmjeru  3. (unesi 0 za kraj) [kg]:9.09
Unesi izmjeru  4. (unesi 0 za kraj) [kg]:9.87
Unesi izmjeru  5. (unesi 0 za kraj) [kg]:0
Srednja masa je 8.558000 kg
Velike mase: 4 (od 5)
```

PRIMJER 3 Tablica (graf) funkcije. Potrebno je generirati tablicu vrijednosti *x* i *y* za funkciju *sin(x)* u rasponu od 0 do π s razmakom 0.1.

Rješenje Prvo je potrebno prijaviti polja *xPolje* i *yPolje*. U polje *x*-eva pridjeljuju se vrijednosti od 0 do π s razmakom 0.1, a u drugo polje vrijednosti *sin*(*x*). Česta pogreška u ovim slučajevima nastaje zbog „miješanja“ cjelobrojnih vrijednosti i realnih brojeva. Naime, indeksi polja moraju biti cijeli brojevi. Primjer neispravnog korištenja polja:

```
for (i = 0; i < 3.1416; i = i + 0.1)
    poljeX[i] = i;
```

Da bismo ispravno riješili zadani problem, potrebno je uvesti zasebnu cjelobrojnu varijablu *i* (koja se varira od 0 do broja elemenata polja), te zasebnu realnu varijablu *x* (koja se varira od 0 do π). Varijablu *i* potrebno je prije petlje inicijalizirati na nulu, a na samom kraju petlje uvećavati je za jedan.

```
int main()
{
    int i=0;
    double x,poljeX[100],poljeY[100];
    for (x = 0; x <= 3.1416; x = x + 0.1)
    {
        poljeX[i] = x;
        poljeY[i] = sin(x);
        i++;
    }

    return 0;
}
```

PRIMJER 4 Sortiranje polja

Rješenje U programiranju se često javlja potreba za sortiranjem polja. Ovdje je prikazana jednostavna verzija algoritma za sortiranje. Korisnik unosi veličinu vektora, a zatim i pojedine članove, dok se članovi vektora sortiraju od najmanjeg prema najvećem. Algoritam za svaki element (od prvog do predzadnjeg), vrši pretragu po svim elementima „ispred“ trenutnog. Ako pronađe manji element, vrši se zamjena.

```
int main()
{
    int v[100];
    int i,j,velicinaVektora, temp;

    printf("Unesi velicinu vektora: ");
    scanf_s("%d", &velicinaVektora);
    for (i = 0; i < velicinaVektora; i++)
    {
        printf("Unesite v[%d]: ", i);
        scanf_s("%d", &v[i]);
    }

    for(i=0;i<(velicinaVektora-1);i++)
        for(j=i+1;j<velicinaVektora;j++)
            if (v[j] < v[i])
            {
                temp = v[i];
                v[i] = v[j];
                v[j] = temp;
            }
    printf("Sortirani vektor: ");
    for (i = 0; i < velicinaVektora; i++)
        printf("%d ", v[i]);

      return 0;
}
```

Primjer korištenja:

```
Unesi velicinu vektora: 3
Unesite v[0]: 5
Unesite v[1]: 1
Unesite v[2]: 4
Sortirani vektor: 1 4 5
```

Pitanja za provjeru znanja 4.1

- Potrebno je izraditi vektor s prvih 100 parnih brojeva. U nastavku je pokušaj rješenja. Pronađite pogreške:

```
int i, vektor[100];
for (i = 2; i<200; i = i + 2)
        vektor[i] = i ;
```

- Što je potrebno izmijeniti u prethodnom programu za sortiranje, tako da polje bude sortirano od najvećeg prema najmanjem?

4.1.1 Zadaci za vježbu

1. Napišite program koji generira polje s 10 članova. Korisnik unosi prva dva člana, a svaki sljedeći član se računa kao zbroj prethodna dva. Ispiši dobiveno polje.

2⁺. Program generira vektor *A* s 10 nasumičnih cijelih brojeva u rasponu od 1 do 100. Potrebno je sve parne brojeve prepisati u zasebni vektor *B*, onim redoslijedom kojim se pojavljuju. U programu možete koristiti prethodno definiranu funkciju *RandUGranicama(d,g)* za generiranje slučajnih brojeva.

3. Korisnik unosi *n* realnih brojeva u polje *A*. Napraviti polje *dA* koje ima broj članova za jedan manje od polja *A*. Članovi u polju *dA* su jednaki razlici idućeg i trenutnog člana polja $dA(i)=A(i+1)-A(i)$.

4. Korisnik unosi polje s *n* realnih brojeva. Osigurajte da su uneseni elementi pozitivni. Generirajte novi vektor *B* kojemu je svaki član vektora jednak sumi trenutnog i svih prethodnih članova izvornog vektora.

5. Korisnik unosi polje s *n* realnih brojeva. Ispišite drugi po redu najveći broj i indeks na kojem se nalazi.

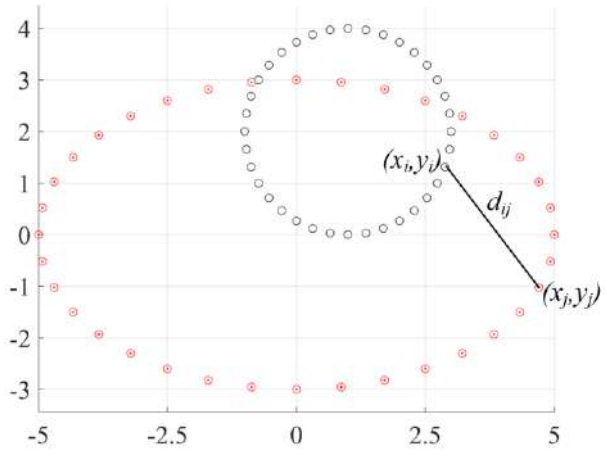
6. Korisnik unosi polje s *n* cjelobrojnih članova. Potrebno je ispisati koliko se često pojedini broj ponavlja. Primjer očekivanog tijeka programa:

```
Unesi veličinu polja: 7
Unesi elemente polja: 1 2 2 3 5 5 1
Rezultat:
Broj 1 se ponavlja 2 puta
Broj 2 se ponavlja 2 puta
Broj 3 se ponavlja 1 puta
...
```

7. Korisnik unosi polje brojeva. Nakon unosa, potrebno je unutar polja zamijeniti vrijednosti na način da polje ima obratni redoslijed.
Primjer:
Elementi u unesenom polju su: 10, 20, 30, 40, 50
Polje sa zamijenjenim vrijednostima: 50, 40, 30, 20, 10

8. Korisnik unosi realne brojeve (najviše 100) u polje, sve dok je aritmetička sredina polja manja ili jednaka prvom elementu polja. Pri završetku unosa, program ispisuje polje te njegovu aritmetičku sredinu.

9. Korisnik unosi poluosi elipse (*a* i *b*) sa središtem u ishodištu, radijus kružnice (*r*) te koordinate središta kružnice (*xS*,*yS*). Potrebno je u polja (*xe*,*ye*,*xk*,*yk*) pohraniti koordinate točaka elipse i kružnice, gdje se točke uzimaju promjenom kuta α od 0° do 360° s razmakom od 10°. Pronađite točku na elipsi i točku na kružnici s najmanjom udaljenosti (minimalni d_{ij} - slika ispod) te ispišite koordinate pronađenih točaka i udaljenost.



Napomena, za generiranje polja koristite parametarske jednadžbe:
kružnica: $xk = xS + r \cdot \sin(\alpha)$ i $yk = yS + r \cdot \cos(\alpha)$
elipsa sa središtem u ishodištu: $xe = a \cdot \sin(\alpha)$ i $ye = b \cdot \sin(\alpha)$

10*. Napišite program koji za uneseno polje traži dva broja čija je suma najbliža nuli. Primjer

```
...//Unos elemenata polja
Uneseno polje je: 37 -47 44 61 -52 15 82 -71 -32 2
Par brojeva sa sumom najbližom nuli je: -47, 44
```

11*. Napišite program koji za uneseno polje, traži točno četiri elementa polja kojima je suma jednaka nekom broju koji odabire korisnik. Primjer

```
...//Unos elemenata polja
Uneseno polje je: 7 3 6 12 1 15 21 9 41
Unesite neki broj: 37
Broj 37 se može dobiti kao suma brojeva: 3, 1, 12 i 21
```

4.1.2 Riješeni primjeri

Zadaci za vježbu označeni znakom + riješeni su u nastavku.

ZADATAK 2 Program generira vektor *A* s 10 nasumičnih cijelih brojeva u rasponu od 1 do 100. Potrebno je sve parne brojeve prepisati u zasebni vektor *B*, onim redoslijedom kojim se pojavljuju. U programu možete koristiti prethodno definiranu funkciju *RandUGranicama(d,g)* za generiranje slučajnih brojeva.

Rješenje je prikazano ispod.

```
int main()
{
	int v[10], vektorParnih[10];
	int i, brojParnih=0;

	printf("Izvorni vektor: ");
	for (i = 0; i < 10; i++)
	{
		v[i] = RandUGranicama(1, 100);
		printf("%d ", v[i]);
	}
	for(i=0;i<10;i++)
		if ( v[i]%2 == 0 )
		{
			vektorParnih[brojParnih] = v[i];
			brojParnih = brojParnih + 1;
		}
	printf("\nIzdvojeni vektor: ");
	for (i = 0; i < brojParnih; i++)
		 printf("%d ",vektorParnih[i]);

	return 0;
}
```

Ispis rezultata:

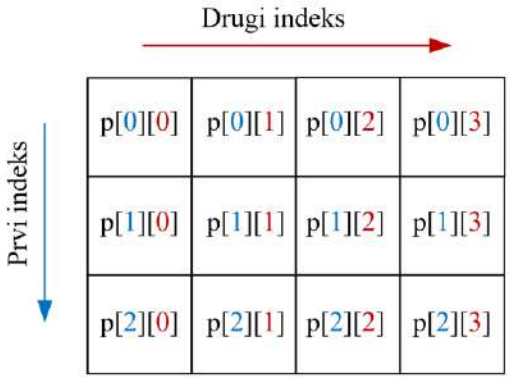
```
Izvorni vektor: 42 68 35 1 70 25 79 59 63 65
Izdvojeni vektor: 42 68 70
```

4.2 Višedimenzionalna polja

Za prijaviti višedimenzionalna polja, jednostavno se kod prijavljivanja dodaju još jedne uglate zagrade:

```
int polje[10][20];
```

Na ovaj način prijavljeno je dvodimenzionalno polje s 10 „redaka“ i 20 „stupaca“, što bi odgovaralo prijavljivanju ukupno 200 varijabli: *polje[0][0], polje[0][1], …, polje[0][19], polje[1][0],polje[1][1], …, …, polje [9][19]*. Ovo se može vizualno prikazati kao tablica, za primjer prijavljenog polja *int p[3][4]*, tablični prikaz je na donjoj slici.



Iako se može ići i na više od dvije dimenzije, u praksi se to rijetko koristi. Ako se elemente matrice želi inicijalizirati na određene vrijednosti, to se radi na sljedeći način:

```
double A[2][3] =    {
                    {2,1,-1},
                    {-3, -2, 2}
                    };
```

PRIMJER 5 Korisnik unosi željeni broj redaka i stupaca, a zatim sve elemente matrice cijelih brojeva.

Rješenje Za unos svih redaka i stupaca potrebno je koristiti ugniježđenu petlju. Prva petlja ide po svim redcima, a zatim unutarnja petlja, za svaki redak ide po svim stupcima.

```
int main()
{
    int mat[100][100],brRedaka,brStupaca,r,s;

    printf("Unesi broj redaka: ");
    scanf_s("%d", &brRedaka);
    printf("Unesi broj stupaca: ");
    scanf_s("%d", &brStupaca);
    for (r = 0; r < brRedaka; r++)
    {
        for (s = 0; s < brStupaca; s++)
        {
            printf("mat[%d][%d] = ", r, s);
            scanf_s("%d", &mat[r][s]);
        }
    }
    return 0;
}
```

U programu je inicijalizirana matrica sa 100 redaka i 100 stupaca. Time je određen maksimalan broj redaka i stupaca koji se u programu mogu koristiti, ali naravno može se koristiti i manje. U prethodnom

programu, korisnik prvo unosi broj redaka i stupaca, a zatim elemente matrice. Ako korisnik unese više od 100 redaka ili stupaca, program će se svejedno izvršavati. Međutim, prilikom unosa elementa matrice izvan maksimalnog raspona (npr. *mat[100][103]*) nastat će pogreška.

Ispis matrice Za ispisati matricu, kao nastavak na prethodni program potrebno je još jedan put dodati petlju sličnu prethodnoj. Dvije *for* petlje su iste kao i ranije. U unutarnjoj petlji nalazi se samo naredba za ispis elementa matrice iz *r*-tog retka i *s*-tog stupca *mat[r][s]*. Osim ispisa broja, dodana je oznaka „\t“, koja dodaje tabulator koji je važan zbog urednog ispisa. Nakon unutarnje petlje, sljedeća *printf* naredba ispisuje samo jedan novi redak. Ovo je važno za odvojiti pojedine retke u ispisu.

```
for (r = 0; r < brRedaka; r++)
{
    for (s = 0; s < brStupaca; s++)
    {
        printf("%d\t",mat[r][s]);
    }
    printf("\n");
}
```

Test programa, korisnik prvo unosi broj redaka i stupaca, a zatim jedan po jedan element matrice:

```
Unesi broj redaka: 2
Unesi broj stupaca: 3
mat[0][0] = 5
mat[0][1] = 3
mat[0][2] = 1
mat[1][0] = 9
mat[1][1] = 7
mat[1][2] = 5
5       3       1
9       7       5
```

4.2.1 Zadaci za vježbu

Ovdje su zadaci za vježbu koji se rješavaju korištenjem dvodimenzionalnih polja (matrica).

12. Napišite program za zbrajanje dviju matrica jednake veličine.

13. Napišite program koji učitava dvije matrice i provjerava jesu li jednake.

14⁺. Potrebno je napisati sljedeći program. Korisnik sa standardnog ulaza unosi matricu, ograničeno do 10 redaka i stupaca. Osigurati ispravan unos u smislu veličine matrice; ako unos nije ispravan, osigurati ponavljanje sve dok se ne unese ispravno. Potrebno je pronaći najmanji pozitivan broj te najveći negativan broj.

15⁺. Napravite program za obradu podataka o radu strojeva. U program se unosi tablica (matrica) s podacima o broju radnih sati pojedinog stroja, po kvartalima, za jednu godinu. Tablica ima 4 stupca u kojima se nalaze podaci o broju radnih sati po kvartalima:

	Broj radnih sati pojedinog stroja			
Kodni broj stroja	Kvartal 1	Kvartal 2	Kvartal 3	Kvartal 4
1	…	…	…	…
2				
3				
…				

Potrebno je: a) Pronaći stroj s najmanjim brojem radnih sati. b) Koliko strojeva ima u barem jednom kvartalu nula radnih sati.

16. Napišite program za unos matrice cijelih brojeva i obradu podataka. Matrica ima *r* redaka i *s* stupaca, a korisnik programa odabire broj redaka i stupaca. Program treba napraviti sljedeće stavke te ispisati rezultat za svaku od njih:

a) Izračunati aritmetičku sredinu pozitivnih elemenata matrice
b) Pretvoriti elemente koji su veći od aritmetičke sredine u aritmetičku sredinu
c) Ako je matrica kvadratna. Izračunati sumu glavne dijagonale (*i== j*) i sporedne dijagonale (*j == s-i+1)*
d) Ako je matrica kvadratna. Po redoslijedu, zamijeniti elemente glavne dijagonale s elementima u prvom retku
e) Napravite novu matricu kojoj je svaki element suma svih elemenata iznad njega
f) Izračunati sumu elemenata donje trokutaste matrice (elementi na glavnoj dijagonali i ispod glavne dijagonale).

17. Napišite program za transponiranje matrice.

18*. Napišite program koji učitava dvije kvadratne matrice jednakih dimenzija i računa matrični umnožak.

19*. Napišite program koji za unesenu matricu ispisuje sve retke koji se nalaze u matrici ali, bez dupliranja ispisa identičnih redaka. Primjer:

```
...//Unos elemenata matrice
Unesena matrica je:
0 1 0 0 1
1 0 0 0 0
0 1 0 0 1
1 0 1 0 1
Retci koji se nalaze u matrici:
0 1 0 0 1
1 0 0 0 0
1 0 1 0 1
```

4.3 Riješeni zadaci

Zadaci za vježbu označeni znakom + riješeni su u nastavku.

ZADATAK 14 Potrebno je napisati sljedeći program. Korisnik sa standardnog ulaza unosi matricu, ograničeno do 10 redaka i stupaca. Osigurati ispravan unos u smislu veličine matrice. Ako unos nije ispravan, osigurati ponavljanje sve dok se ne unese ispravno. Potrebno je pronaći najmanji pozitivan broj te najveći negativan broj.

Rješenje je prikazano ispod.

```
int main()
{
	int m[10][10], brRedaka, brStupaca;
	int i, j, najveciNegativni, najmanjiPozivivni;
	do
	{
		printf("Unesi broj redaka: ");
		scanf_s("%d", &brRedaka);
	} while (brRedaka > 10);
	do
	{
		printf("Unesi broj stupaca: ");
		scanf_s("%d", &brStupaca);
	} while (brStupaca > 10);

	for (i = 0; i < brRedaka; i++)
		for (j = 0; j < brStupaca; j++)
		{
			printf("Element matrice [%d][%d]: ", i, j);
			scanf_s("%d", &m[i][j]);
		}
	//Pretpostavlja se najmanji mogući broj i traži se veći:
	najveciNegativni = -1e9;
	//Pretpostavlja se najveći mogući broj i traži se što manji:
	najmanjiPozivivni = 1e9;
	for (i = 0; i < brRedaka; i++)
		for (j = 0; j < brStupaca; j++)
		{
			if (m[i][j] > 0 && m[i][j] < najmanjiPozivivni)
				najmanjiPozivivni = m[i][j];
			if (m[i][j] < 0 && m[i][j] > najveciNegativni)
				najveciNegativni = m[i][j];
		}
	printf("Najveci negativni broj u matrici je: %d\n", najveciNegativni);
	printf("Najmanji pozitivni  broj u matrici je: %d\n", najmanjiPozivivni);

	return 0;
}
```

ZADATAK 15 Obrada podataka o godišnjem radu strojeva po kvartalima (Laboratorijska vježba 5)
U program se unosi tablica (matrica) s podacima o broju radnih sati pojedinog stroja, po kvartalima, za jednu godinu. Tablica ima 4 stupca u kojima se nalaze podaci o broju radnih sati po kvartalima:

	Broj radnih sati pojedinog stroja			
Kodni broj stroja	Kvartal 1	Kvartal 2	Kvartal 3	Kvartal 4
1	…	…	…	…
2				
3				
…				

Zadatak je: a) Pronaći stroj s najmanjim brojem radnih sati. b) Koliko strojeva ima u barem jednom kvartalu nula radnih sati?

Rješenje je prikazano ispod.

```
int main()
{	int i, j, polje[100][4], minSati, sumaSati = 0, idStroja, brojStrojeva;
	printf("Unesi broj strojeva: ");  scanf_s("%d", &brojStrojeva);
	for (i = 0; i<brojStrojeva; i++)
	{
		for (j = 0; j<4; j++)
		{
			printf("Unesi broj radnih sati za stroj %d. Q%d: ", i + 1, j + 1);
			scanf_s("%d", &polje[i][j]);
		}
	}
	minSati = polje[0][0] + polje[0][1] + polje[0][2] + polje[0][3];
	idStroja = 0;
	for (i = 1; i<brojStrojeva; i++)
	{
		sumaSati = 0;
		for (j = 0; j<4; j++)
		{
			sumaSati = sumaSati + polje[i][j];
		}
		if (sumaSati<minSati)
		{
			minSati = sumaSati;
			idStroja = i;
		}
	}
	printf("Stroj %d radi najmanje sati (%d sati)\n", idStroja + 1, minSati);

	int brojacNula = 0, strojImaNulu;
	for (i = 0; i<brojStrojeva; i++)
	{
		strojImaNulu = 0;
		for (j = 0; j<4; j++)
		{
			if (polje[i][j] == 0)
				strojImaNulu = 1;
		}
		brojacNula = brojacNula + strojImaNulu;
	}
	printf("Strojeva koji u nekim kvartalima rade 0 sati ima: %d\n", brojacNula);
	return 0;
```

4.4 Znakovi i znakovni nizovi

Znakovi su varijable tipa *char*, a često se koriste u interakciji s korisnikom. Na primjer, od korisnika se može zatražiti upit o tome želi li ispis nekih rezultata:

```
#include<stdio.h>

int main()
{
	char provjera;
	printf("Zelite li ispis (y-yes, n-no)? ");
	scanf_s("%c", &provjera);
	if (provjera == 'y')
		printf("Ispis!\n");
	else if(provjera == 'n')
		printf("Nema ispisa!\n");

	return 0;
}
```

Korisnik zatim unosi *y*, ako želi ili *n* ako ne želi. Tijek programa tako može biti:

```
Da li zelite ispis (y-yes, n-no)? y
Ispis!
```

Navodnici Pazite na to da kada se radi sa znakovima, koriste se jednostruki navodnici, a ne dvostruki:

```
char znak;
	znak = 'a'; //Ispravno
	znak = "a"; //Nije ispravno
```

ASCII tablica Pod znakove u C-u spada 128 znakova koji su definirani prema ASCII tablici. Znakovi se mogu koristiti i kao obične cjelobrojne varijable uz ograničenje da broj može iznositi između 0 i 127. Primjer primjene aritmetičkih operacija sa znakovnim varijablama

```
	char c;
	c = 10;
	printf("c=%d\n", c) ;
	c = 8*c;
	printf("c=%d\n", c);
	c = c-33;
	printf("c=%d\n", c);
```

Ispis:

```
c=10
c=80
c=47
```

U nastavku je program koji ispisuje dio ASCII tablice. Ovdje se koristi činjenica da se sa znakovnim varijablama mogu koristiti i aritmetički operatori, te da se taj znak može ispisati i kao broj i kao znak.

```
int main()
{
	char c;
	printf("ASCII broj \tZnak\n");
	for (c = 32; c < 126; c++)
		printf("%d\t\t%c\n", c,c);

	return 0;
}
```

U nastavku je izdvojen samo dio ispisa:

```
ASCII broj      Znak
32
33              !
43              +
48              0
49              1
50              2
65              A
66              B
97              a
98              b
122             z
```

Prvi znak u ispisu (redni broj 32) izgleda kao da nije ispisan, zapravo se radi o *razmaku*. Slično tako, u ASCII tablici nalaze se i znakovi za tabulator „\t“ te novu liniju „\n“, kao razni drugi znakovi, među koje spadaju i brojevi, znakovi za aritmetičke operacije itd. Znakovi specifični za hrvatski jezik (č,ć…) nisu sadržani u ASCII tablici.

Pitanja za provjeru znanja 4.2

- Napravite jednostavni program koji provjerava je li uneseni znak malo ili veliko slovo (ili nije slovo). Iskoristite poznavanje ASCII redoslijeda znakova.

ZNAKOVNI NIZOVI Kao posebna vrsta polja, mogu se izdvojiti znakovni nizovi ***char[]***, koji se u programiranju nazivaju *string*. Ovo su nizovi tipa *char*, a specifičnost znakovnih nizova je što završavaju sa specijalnim nul-znakom '\0'. Kod pisanja kôda niz se definira korištenjem navodnika.
Primjerice, niz *char s[]=“niz 1“*, rezultira nizom:

Naziv:	s[0]	s[1]	s[2]	s[3]	s[4]	s[5]
Vrijednost:	n	i	z		1	\0

PRIJAVLJIVANJE Kod prijavljivanja niza znakova, isti se rezultat dobije i ako se niz prijavljuje dodavanjem pojedinačnih znakova. Niz se ispisuje korištenjem oznake formata *%s*:

```
char s1[] = "niz 1";
char s2[] = {'n', 'i', 'z', ' ', '1', '\0'};
printf("Znakovni niz 1: %s\n", s1);
printf("Znakovni niz 2: %s\n", s2);
```

Ispis:

```
Znakovni niz 1: niz 1
Znakovni niz 2: niz 1
```

UČITAVANJE Kod učitavanja znakovnog niza od unosa korisnika, ponovno se koristi funkcija *scanf_s*. Kod znakovnih nizova, prilikom učitavanja niza potrebno je i definirati maksimalni broj znakova. Ako je prijavljen niz *char niz[10]*, za učitavanje niza koristi se *scanf_s("%s",&niz,9)*. Broj 9 znači da se može

učitati maksimalno 9 znakova u polje, a zadnji znak treba biti rezerviran za nul-znak '\0'. Kod učitavanja niza na ovaj način, u niz će se učitavati svi znakovi do prvog razmaka:

```
char punoIme[50];
	printf("Unesite ime i prezime: ");
	scanf_s("%s", &punoIme,49);
	printf("Puno ime: %s\n", punoIme);
```

Korisnik unosi ime i prezime (Ante Bilic). U ispisu je vidljivo samo ime, jer je funkcija učitavala sve znakove do razmaka:

```
Unesite ime i prezime: Ante Bilic
Puno ime: Ante
```

PRIMJER 6 Korisnik unosi ime i prezime u zasebne znakovne nizove. Ispišite uneseno ime i prezime. **Rješenje** programa je u nastavku.

```
#include<stdio.h>

int main()
{
	char ime[20], prezime[30];
	printf("Unesite ime: ");
	scanf_s("%s", &ime,19);
	printf("Unesite prezime: ");
	scanf_s("%s", &prezime, 29);
	printf("Puno ime: %s %s\n", ime,prezime);

	return 0;
}
```

Test:

```
Unesite ime: Ante
Unesite prezime: Bilic
Puno ime: Ante Bilic
```

4.5 Inženjerski primjeri

PRIMJER 7 Sortiranje radnika po broju radnih sati. U matričnom obliku zadana je tablica podataka o radnicima. Svaki redak sadržava podatke o jednom radniku. Prvi stupac sadržava kodni broj radnika, drugi stupac ukupan broj redovnih radnih sati, a treći stupac ukupan broj prekovremenih radnih sati za tekuću godinu kao što je prikazano u donjoj tablici.

Kodni broj radnika	Redovni radni sati	Prekovremeni radni sati
1	1040	0
10	980	480
20	720	240
21	840	80

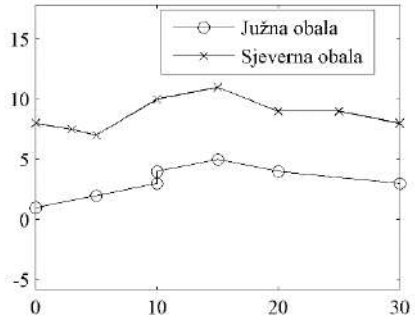
Rješenje Nakon što su podaci o radnicima učitani u obliku matrice, potrebno je prvo napraviti zaseban vektor koji sadržava ukupan broj radnih sati (zbrajanjem druga dva stupca). Ovaj je vektor zatim potrebno sortirati od najvećeg do najmanjeg člana. Uz ovaj vektor, potrebno je pohraniti i kodne brojeve radnika koji će se sortirati jednakim redoslijedom kao i broj radnih sati tako da se na kraju može lakše identificirati radnik.

```
int main()
{
	int M[100][3] = {	{1, 1040,0},
				{10, 980, 480},
				{20, 720, 240},
				{21, 840, 80}
			};
	int ukRadSat[100], indeksiRadnika[100];
	int i, j, brRadnika = 4,temp;
	//Zbrajanje ukupnih radnih sati u zasebno polje
	printf("Ukupan broj radnih sati po radnicima: \n");
	for (i = 0; i < brRadnika; i++)
	{
		ukRadSat[i] = M[i][1] + M[i][2];
		indeksiRadnika[i] = M[i][0];
		printf("%d\n",ukRadSat[i]);
	}
	//Sortiranje poja "ukRadSat" i "indeksiRadnika"
	for(i=0;i<(brRadnika-1);i++)
	for(j=i+1;j<brRadnika;j++)
	if (ukRadSat[j] > ukRadSat[i])
	{
		//Zamjena broja radnih sati
		temp = ukRadSat[i];
		ukRadSat[i] = ukRadSat[j];
		ukRadSat[j] = temp;
		//Zamjena indeksa
		temp = indeksiRadnika[i];
		indeksiRadnika[i] = indeksiRadnika[j];
		indeksiRadnika[j] = temp;
	}
	//Ispis sortiranih rezultata
	printf("Radnici (indeksi) sortirani po broju radnih sati: \n");
	for (i = 0; i < brRadnika; i++)
		printf("%d\t%d\n",indeksiRadnika[i],ukRadSat[i]);
	return 0;
```

Za sortiranje je korišten algoritam koji kreće od prvog člana vektora (*i=0*), a zatim ga uspoređuje sa svim sljedećim članovima vektora (*j=1* do *brRadnika*). Svaki put kada se pronađe član s većim iznosom, vrši se zamjena. Rezultat ovoga je da će se na prvom mjestu u vektoru nalaziti najveći iznos iz cijelog vektora. Sada se ponavlja isti postupak s idućim članom vektora (*i=1*), koji se uspoređuje sa svim sljedećim članovima (*j=2* do *brRadnika*). Rezultat ovoga je da će se na drugom mjestu nalaziti sljedeći najveći član vektora (na prvom mjestu ostaje najveći iz prethodnog koraka). Ovo se ponavlja sve do predzadnjeg člana (*i=brRadnika-2*), kojega je potrebno usporediti samo s preostalim zadnjim članom vektora (*j=brRadnika-1*). Ovaj algoritam za sortiranje je općenit i primjenjuje se i u drugim zadacima sortiranja.
Rezultat programa je sortirani vektor s ukupnim brojem radnih sati i vektor s indeksima pripadajućih radnika.

```
Ukupan broj radnih sati po radnicima:
        1040
        1460
         960
         920
Radnici (indeksi) sortirani po broju radnih sati:
          10        1460
           1        1040
          20         960
          21         920
```

PRIMJER 8 Zadana su dva skupa točaka koja predstavljaju dvije linije obale. Potrebno je pronaći dvije najbliže točke između ta dva skupa točaka. Svaki skup točaka zadan je pomoću dva vektora s *x* i *y* koordinatama zasebno. Zadani skup točaka prikazan je na donjoj slici.



Rješenje zadatka prikazano je u nastavku. Za pronaći najbliže točke, prvi je korak pretpostaviti rješenje. Pretpostavljeno je da su najbliže točke prve točke iz oba skupa (*x1[0],y1[0]*) i (*x2[0],y2[0]*) i da je najmanja udaljenost jednaka udaljenosti između tih dviju točaka ($udaljenost = \sqrt{\Delta x^2 + \Delta y^2}$). Nadalje je potrebno usporediti sve točke iz prvog skupa sa svim točkama iz drugog skupa. Svaki put kada se nađe manja udaljenost od prethodno pretpostavljene, zamjenjuje se *minUdaljenost* i pohranjuju se pripadajući indeksi točaka *mInd1* i *mInd2*.

```
int main()
{
    double x1[]={0, 5, 10, 10, 15, 20, 30}, y1[]={1, 2, 3, 4, 5, 4, 3};
    double x2[]={0, 3, 5, 10, 15, 20, 25, 30,}, y2[]={8, 7.5, 7, 10, 11, 9, 9, 8,};
    int s1, s2, brToc1 = 7, brToc2 = 8;
    int mInd1 = 1, mInd2 = 1; // Početno rješenje
    double minUdaljenost;

    minUdaljenost = sqrt( pow(x1[mInd1]-x2[mInd2],2) + pow(y1[mInd1]-y2[mInd2],2) );
    //Uspoređivanje svih točaka iz prvog skupa sa svim točkama drugog skupa
    for(s1 = 0;s1<brToc1;s1++)
    for(s2 = 0; s2 < brToc2; s2++)
    if(sqrt(pow(x1[s1] - x2[s2], 2) + pow(y1[s1] - y2[s2], 2)) < minUdaljenost)
    {
        mInd1 = s1;
        mInd2 = s2;
        minUdaljenost=sqrt(pow(x1[s1] - x2[s2], 2) + pow(y1[s1] - y2[s2], 2));
    }
    printf("Najbliže točke su:\n");
    printf("Točka %d. (Jug: x=%.2f,y=%.2f) \n", mInd1, x1[mInd1], y1[mInd1]);
    printf("Točka %d. (Sjever: x=%.2f,y=%.2f)\n", mInd2, x2[mInd2], y2[mInd2]);
    printf("Udaljenost iznosi %.2f km\n", minUdaljenost);

       return 0;
}
```

Pokretanjem programa ispisuju se indeksi najbližih točaka i njihove koordinate u prostoru:

```
Najbliže točke su:
Točka 2. (Jug: x=5.00,y=2.00)
Točka 3. (Sjever: x=5.00,y=7.00)
Udaljenost iznosi 5.00 km
```

5 FUNKCIJE

Funkcija je zasebni dio programa koja izvršava željene radnje. Iako se svaki program može riješiti bez korištenja funkcija, funkcije mogu pojednostavniti složene programe. Ovo pogotovo dolazi do izražaja u slučajevima kada se neki dio programa ponavlja više puta.
Funkcije poboljšavaju razumljivost/čitljivost programskog kôda. Na primjer, umjesto:

```
if (broj < 0)
        broj = -1 * broj;
```

Razumljivije je (i jednostavnije) umjesto toga koristiti funkciju: *broj=abs(broj)*.

Pored glavne i jedinstvene *main* funkcije koju svaki program mora imati, u *C*-u postoji određeni skup ugrađenih (predefiniranih) funkcija kao što su već korištene funkcije *printf i* scanf za ispis, odnosno unos podataka. Njihove se definicije nalaze u biblioteci *'stdio.h'*. Tu su također i brojne elementarne matematičke funkcije: *sin, cos, sqrt…*, čije se definicije nalaze u biblioteci 'math.h'. Međutim, pored predefiniranih funkcija moguće je kreirati i vlastite korisnički definirane funkcije.
Funkcije se općenito razlikuju po ulaznim argumentima i rezultatima koje vraćaju. U općem obliku, funkcija ima tip rezultata (npr *double* ili *int*) koji se navodi na početku funkcije te ponovno nakon *return*, svaka funkcija ima svoj jedinstveni naziv, proizvoljan broj ulaznih argumenata bilo kojeg tipa (npr. *int, double, double[]*…), te programski kôd koji definira što funkcija radi.

Opći oblik korisnički definirane funkcije Funkciju je potrebno definirati na početku programa (prije *main* funkcije). Opći oblik za definiranje funkcije je:

```
tipRezultata NazivFunkcije(tip1 arg1, tip2 arg2,…)
{
        radnjeFunkcije;
        return rezultat;
}
```

PROTOTIP FUNKCIJE U C-u, funkcije mogu pozivati samo funkcije definirane u kôdu iznad njih. Kada program sadržava velik broj funkcija, teško je paziti na ovo pravilo, a problem se može premostiti korištenjem ***prototipa funkcije***. Prototip funkcije služi da se na početku programskog kôda odmah nakon dodavanja biblioteka (nakon *#include<stdio.h>*) navedu sve funkcije koje će se koristiti u programu. Prototip može biti potpuno isti kao i prva linija koja definira naziv i tip funkcije s dodanim znakom točka-zarez. Za prethodni primjer prototip funkcije bi bio:

```
tipRezultata NazivFunkcije(tip1 arg1, tip2 arg2,…);
```

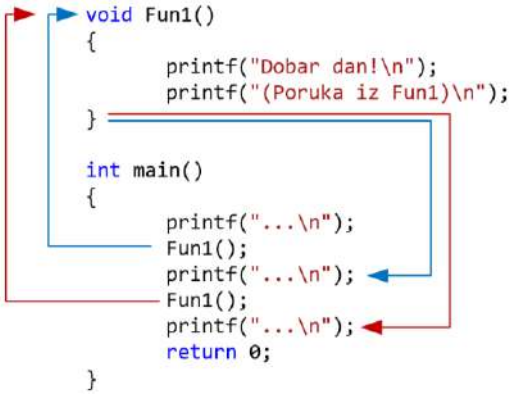
Dakle, kao prvi korak mogu se definirati prototipovi svih funkcija koje će se koristiti u programu, a zatim se definira i sadržaj pojedinih funkcija. U ovim slučajevima treba paziti na kružne definicije funkcija (u prvoj funkciji pozivate drugu, a u drugoj prvu).

5.1 Funkcije bez argumenata

Najjednostavnija funkcija je funkcija koja nema ulaznih argumenata niti povratnu vrijednost. Opći oblik kod definiranja ovakvih funkcija je:

```
void NazivFunkcije()
{
	radnjeFunkcije;
}
```

Ovakva funkcija, kada se pozove, izvršava radnje koje su specificirane unutar funkcije. Tok programa koji koristi ovakvu funkciju shematski je prikazan na sljedećoj skici.



S obzirom na to da program dva puta poziva funkciju, dva puta se ispisuje poruka iz funkcije:

```
...
Dobar dan!
(Poruka iz Fun1)
...
Dobar dan!
(Poruka iz Fun1)
...
```

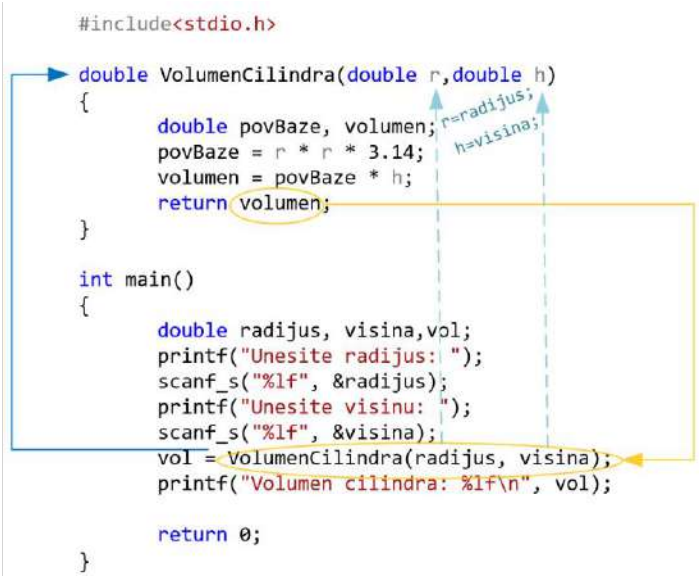
PRIMJER 1 Program za izračun naprezanja s kontrolom uz ponavljanje (Laboratorijska vježba 6). Provjeru naprezanja u grednom nosaču napraviti tako da se program ponavlja sve dok ga korisnik ne prekine.

Rješenje je prikazano u nastavku.

```
#include <stdio.h>
void FunkcijaProgramGreda()//Definicija funkcije
{
	float L, q, b, h, I, E, sigma; b = 10; h = 20;
	I = b * h*h*h / 12; E = 210000;
	printf("Unesi duljinu grede [mm]: ");
	scanf_s("%f", &L); printf("Unesi iznos opterecenja [N/mm]: ");
	scanf_s("%f", &q); sigma = h / 2 * q*L*L / (8 * I);
	printf("Naprezanje iznosi sigma=%f MPa\n", sigma);
	if (sigma<100.0)
		printf("Naprezanje je dopusteno\n");
	else if (sigma<120.0)
		printf("Naprezanje je granicno\n");
	else if (sigma<130.0)
		printf("Naprezanje je na gornjoj granici\n");
	else
		printf("Naprezanje nije dopusteno\n");
}
int main()
{
	int zelimPonavljat = 1;

	while (zelimPonavljat == 1)
	{
		FunkcijaProgramGreda();
		printf("Zelite li ponovit proracun (1-da ili 0-ne)? ");
		scanf_s("%d", &zelimPonavljat);
	}
	return 0;
}
```

5.2 Funkcija s argumentima i povratnim rezultatom

Primjer funkcije koja ima ulazne argumente i povratne rezultate prikazan je u nastavku. Prilikom poziva funkcije, potrebno joj je poslati dva realna broja tipa *double*. Dva poslana broja zapravo se „kopiraju“ u lokalne varijable *r* i *h*. Nakon što funkcija izračuna volumen cilindra, rezultat se vraća nazad u glavnu funkciju. U ovom slučaju, taj se rezultat pohranjuje u lokalnu varijablu *vol* u glavnoj funkciji te se ispisuje u sljedećoj liniji.




```
#include<stdio.h>

double VolumenCilindra(double r,double h)
{
        double povBaze, volumen;
        povBaze = r * r * 3.14;
        volumen = povBaze * h;
        return volumen;
}

int main()
{
        double radijus, visina,vol;
        printf("Unesite radijus: ");
        scanf_s("%lf", &radijus);
        printf("Unesite visinu: ");
        scanf_s("%lf", &visina);
        vol = VolumenCilindra(radijus, visina);
        printf("Volumen cilindra: %lf\n", vol);

        return 0;
}
```

Ispis programa:

```
Unesite radijus: 18
Unesite visinu: 2
Volumen cilindra: 2034.720000
```

Pitanja za provjeru znanja 5.1

- Sljedeći program trebao bi računati volumen cilindra pomoću korisnički definirane funkcije. Program nije ispravan, zašto?

```
#include<studio.h>
float IzracunVolumenaCilindra()
{
        float r, h, volumen;
        volumen = r * r*h;
        return volumen;
}
int main()
{
        float r = 4, h = 12, vol;
        vol = IzracunVolumenaCilindra();
        printf("Volumen cilindra r=%fi h=%f iznosi %f", r, h, vol);
        return 0;
}
```

- Bi li program bio ispravan da se u glavnoj funkciji za izračun volumena koristila naredba `vol=IzracunVolumenaCilindra(r,h);`? Kopirajte prethodni programski kôd, i popravite program

5.3 Lokalne varijable

Varijable definirane u bilo kojoj funkciji su lokalne varijable, što znači da nisu vidljive u drugim funkcijama. Na primjer, sljedeći program ne može se koristiti za izračun površine kruga.

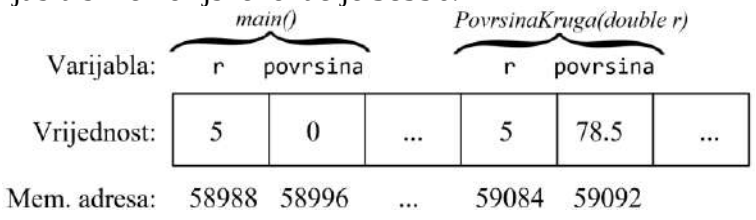
```
#include<stdio.h>
void PovrsinaKruga(double r)
{
	double povrsina;
	povrsina = r * r * 3.1415;
}
int main()
{
	double r=5,povrsina=0;
	PovrsinaKruga(r);
	printf("Povrsina kruga: %lf\n", povrsina);
	return 0;
}
```

Ispis programa:

```
Povrsina kruga: 0.000000
```

Iako funkcija *PovrsinaKruga* ispravno računa površinu, lokalna varijabla *povrsina* unutar funkcije nije ista varijabla kao i varijabla *povrsina* definirana u *main* funkciji. Program ispisuje vrijednost 0.000000 jer lokalna varijabla *povrsina* u funkciji *main* sadržava tu vrijednost (definirano u liniji `double r=5,povrsina=0;`).

RADNA MEMORIJA Za vizualizaciju lokalnih varijabli možemo promatrati lokaciju pojedine varijable u računalnoj radnoj memoriji kao što je prikazano na sljedećoj slici. Svaka varijabla ima pripadajuću fizičku memorijsku lokaciju na kojoj se nalazi. Varijabla *r*, koja je definirana u funkciji *main* (memorijska adresa 58988) i funkciji *PovrsinaKruga* (memorijska adresa 59084), sadržava zapravo dvije fizički odvojene varijable. U ovom slučaju, one mogu imati istu vrijednost jer kada se poziva funkcija *PovrsinaKruga(r)*, funkciji se zapravo šalje realni broj 5.0000, koji se zatim pohranjuje u lokalnu varijablu *r*. Kod izračuna površine, izračunata površina unutar funkcije *PovrsinaKruga()* pohranit će se u lokalnu varijablu *povrsina* na memorijskoj adresi 59092, a prilikom ispisa u glavnoj funkciji, ispisuje se varijabla s memorijske lokacije 58996.



Pitanja za provjeru znanja 5.2

- Sljedeći program računa volumen cilindra pomoću dvije korisnički definirane funkcije. Je li program ispravan?

```
double PovrsinaBazeCilindra(double r, double h);
double VolumenCilindra(double r, double h);
double VolumenCilindra(double r, double h)
{
	return PovrsinaBazeCilindra(r, h)* h;
}
double PovrsinaBazeCilindra(double r, double h)
{
	double povrsina;
	povrsina = VolumenCilindra(r, h) / h;
	return povrsina;
}
int main()
{
	double r=5,h=10;
	printf("Volumen: %lf\n",VolumenCilindra(r,h));
	return 0;
}
```

5.4 Funkcija s poljima kao argumentima

Funkcije se često koriste za obradu podataka pohranjenim u poljima. Jedan od jednostavnih primjera je funkcija za statističku obradu skupa podataka.

PRIMJER 2 Zadatak je napraviti funkciju za izračun standardne devijacije skupa podataka (vektora) koje unosi korisnik. Program treba koristiti korisnički definiranu funkciju za izračun standardne devijacije.

Rješenje Srednja vrijednost i standardna devijacija vektora ***x*** s *n* članova računaju se pomoću izraza:

$$srednjaVrijednost = \frac{1}{n}\sum_{i=1}^{n} x(i)$$

$$standardnaDevijacija = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x(i) - srednjaVrijednost\right)^2}$$

Funkcija za izračun vrijednosti aritmetičke sredine i standardne devijacije prikazana je u nastavku.

```
double StDevVektora (double vektor[100], int n)
{
	int i;
	double std,ave,sum = 0;
	for(i=0;i<n;i++)
		sum = sum + vektor[i];
	ave = sum / n;

	sum = 0;
	for(i = 0;i<n;i++)
		sum = sum + pow(vektor [i] - ave, 2);

	return sqrt(sum / n);
}
```

Glavna funkcija sadržava dio za unos podataka, a zatim poziva funkciju za ispis standardne devijacije:

```
int main()
{
	double std, polje[100];
	int i, brClanova;
	printf("Unesite broj clanova polja: ");
	scanf_s("%d", &brClanova);
	for (i = 0; i < brClanova; i++)
	{
		printf("Unesi polje[%d]: ", i);
		scanf_s("%lf", &polje[i]);
	}
	std= StDevVektora(polje, brClanova);
	printf("Standardna devijacija podataka iznosi %lf",std);

	return 0;
}
```

Primjer unosa polja s 5 članova:

```
Unesite broj clanova polja: 5
```

```
Unesi polje[0]: 2.5
Unesi polje[1]: 3
Unesi polje[2]: 3.2
Unesi polje[3]: 3
Unesi polje[4]: 1
Standardna devijacija podataka iznosi 0.803990
```

POLJA KAO ARGUMENTI U FUNKCIJAMA Ponašanje funkcije može biti drugačije kada se koriste polja kao argumenti u odnosu na običnu varijablu. Kada se polja šalju funkcijama kao ulazni argumenti, izmjene napravljene u polju, ostaju zapamćene i izvan te funkcije pa tako ostaju zapamćene i u glavnoj funkciji. Sljedeći program pokazuje primjer u kojemu funkcija prima polje brojeva. Unutar funkcije, brojevi su izmijenjeni te su u nastavku programa ispisani u glavnoj funkciji.

```c
#include<stdio.h>
void UvecajBrojeve(int brojevi[3])
{
	brojevi[0] = brojevi[0] + 1;
	brojevi[1] = brojevi[1] + 2;
	brojevi[2] = brojevi[2] + 3;
}
int main()
{
	int brojevi[3] = { 1, 2, 3 };
	printf("Brojevi (prije fun.): %d,%d,%d\n", brojevi[0], brojevi[1], brojevi[2]);
	UvecajBrojeve(brojevi);
	printf("Brojevi (nakon fun.): %d,%d,%d\n", brojevi[0], brojevi[1], brojevi[2]);
	return 0;
}
```

Izmjene napravljene u funkciji zadržane su i u glavnoj funkciji:

```
Brojevi (prije fun.): 1,2,3
Brojevi (nakon fun.): 2,4,6
```

Isti program napravljen korištenjem običnih varijabli:

```c
#include<stdio.h>
void UvecajBrojeve(int broj0, int broj1, int broj2)
{
	broj0 = broj0 + 1;
	broj1 = broj1 + 2;
	broj2 = broj2 + 3;
}
int main()
{
	int broj0 = 1, broj1 = 2, broj2 = 3;
	printf("Brojevi (prije fun.): %d,%d,%d\n", broj0, broj1, broj2);
	UvecajBrojeve(broj0,broj1,broj2);
	printf("Brojevi (nakon fun.): %d,%d,%d\n", broj0, broj1, broj2);
	return 0;
}
```

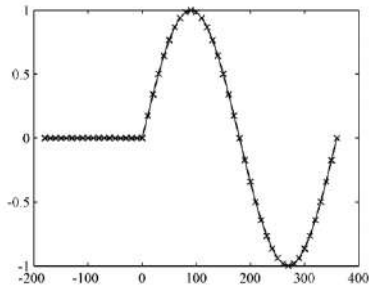
Ispis sada pokazuje da funkcija nije izmijenila brojeve:

```
Brojevi (prije fun.): 1,2,3
Brojevi (nakon fun.): 1,2,3
```

5.5 Zadaci za vježbu

1. Izradi funkciju za evaluaciju polinoma drugog reda $P(x)=ax^2+bx+c$. Funkcija prima koeficijente *a*, *b* i *c*, argument *x*, te vraća rezultat $P(x)$. Zatim izradi program u kojem korisnik unosi prvo koeficijente polinoma (*a*, *b* i *c*). Nakon toga program za proizvoljan *x* ispisuje vrijednost polinoma, što se ponavlja sve dok korisnik ne unese broj 0.

2. Izradi korisnički definiranu funkciju naziva *sinPozitivni*. Funkcija prima kut u stupnjevima i u slučaju da je kut pozitivan vraća sinus tog kuta, a u slučaju da je kut negativan funkcija vraća nulu. (Izgled funkcije prikazan je na donjoj slici).



Nadalje, napravite program u koji korisnik unosi kut u stupnjevima, a program ispisuje rezultat prema funkciji *sinPozitivni*.

3⁺. Potrebno je napraviti program za izračun prosječne vrijednosti elemenata glavne i sporedne dijagonale za matrice kvadratne matrice. Korisnik unosi matricu koju zatim program treba ispisati. Za unos, ispis i izračun prosječne vrijednosti potrebno je koristiti funkcije.

4⁺. Potrebno je napraviti program za unos dviju matrica, zbrajanje i ispis zbrojene matrice. Možete koristiti gotove funkcije za unos i ispis matrica iz prethodnog zadatka. Za zbrajanje napravite zasebnu funkciju.

5. Izradi funkciju koja prima jednodimenzionalno polje, te veličinu polja. Funkcija pronalazi najmanji i najveći član polja te ih zamjenjuje. Ispiši novo polje unutar i izvan funkcije.
Primjer, *polje[]={1 3 4 6 3 0}*, rezultat -> *polje[]={1 3 4 0 3 6}*;

6. Napravite program koji ispisuje vrijednost funkcije *f(x)* za raspon koji unosi korisnik. Funkcija je definirana kao $f(x)=\cdot g(x)+g(x)\cdot h(x)$, gdje je $g(x)=sin(0.1\cdot x)$, a $h(x)=0.01x^2+2x+2.71$.
Primjer ispisa:

```
f(0.000000)=0.000000
f(0.200000)=0.082203
f(0.400000)=0.180416
f(0.600000)=0.294639
f(0.800000)=0.424858
```

7. Korisnik unosi pozitivni cijeli broj *n*. Napravite program koji računa *f(n)*, definiran kao *f(n)=1* za *n<2*, te *f(n)= f(n-1)+ f(n-2)* za *n*≥2.

Primjeri:

```
f(1)=1, f(2)=2, f(3)=3, f(4)=5, f(5)=8, f(6)=13, f(7)=21, f(8)=34, f(9)=55, f(10)=89,
f(11)=144, f(12)=233, f(13)=377, f(14)=610, f(15)=987, f(16)=1597, f(17)=2584, f(18)=4181,
f(19)=6765.
```

5.6 Riješeni zadaci

Zadaci označeni znakom + sada su riješeni.

ZADATAK 3 Program za unos i ispis matrice pomoću funkcija (Laboratorijska vježba 6).
Potrebno je napraviti program za izračun prosječne vrijednosti elemenata glavne i sporedne dijagonale matrice za kvadratne matrice. Korisnik unosi matricu koju zatim program treba ispisati. Za unos, ispis i izračun prosječne vrijednosti potrebno je koristiti funkcije.
Rješenje je prikazano ispod.
Glavna funkcija:

```
int main()
{
	int A[100][100];
	int brRedaka, brStupaca;
	float prosjek;

	printf("Unesi broj redaka: ");
	scanf_s("%d", &brRedaka);

	printf("Unesi broj stupaca: ");
	scanf_s("%d", &brStupaca);

	UnosMatrice(A, brRedaka, brStupaca);
	IspisMatrice(A, brRedaka, brStupaca);

	if (brRedaka == brStupaca)
	{
		prosjek = ProsjekDijagonala(A, brRedaka, brStupaca);
		printf("Prosjek %f\n", prosjek);
	}
	return 0;
}
```

Funkcija za unos matrice (u programskom kôdu, funkcije se trebaju nalaziti iznad *main* funkcije):

```
void UnosMatrice(int mat[100][100], int brRedaka, int brStupaca)
{
	int r, s;

	for (r = 0; r < brRedaka; r += 1)
	{
		for (s = 0; s < brStupaca; s += 1)
		{
			printf("mat[%d][%d] = ", r, s);
			scanf_s("%d", &mat[r][s]);
		}
	}
}
```

Funkcija za ispis matrice:

```
void IspisMatrice(int mat[100][100], int brRedaka, int brStupaca)
{
	int r, s;

	for (r = 0; r < brRedaka; r += 1)
	{
		for (s = 0; s < brStupaca; s += 1)
		{
			printf("%d ", mat[r][s]);
		}
		printf("\n");
	}
}
```

Funkcija za izračun prosjeka obiju dijagonala matrice:

```
float ProsjekDijagonala(int mat[100][100], int brRedaka, int brStupaca)
{
	float suma = 0;
	int r = 0;
	while (r < brRedaka)
	{
		suma += mat[r][r];
		suma += mat[r][brRedaka - 1 - r];
		r++;
	}
	return suma / (2 * brRedaka);
}
```

Za izračun prosjeka dijagonala matrice, funkcija se može izvesti na različite načine. Primjer u nastavku koristi ugniježđenu petlju.

```
float ProsjekDijagonala(int mat[100][100], int brRedaka, int brStupaca)
{
	float suma = 0;
	int r, s;
	for (r = 0; r < brRedaka; r += 1)
	{
		for (s = 0; s < brStupaca; s += 1)
		{
			if (r == s)
				suma += mat[r][s];

			if (r + s == brRedaka - 1)
				suma += mat[r][s];
		}
	}
	return suma / (2 * brRedaka);
}
```

ZADATAK 4 Potrebno je napraviti program za unos dviju matrica, zbrajanje i ispis zbrojene matrice. Možete koristiti gotove funkcije za unos i ispis matrica iz prethodnog zadatka. Za zbrajanje napravite zasebnu funkciju.

Rješenje je prikazano ispod.

Funkcija za zbrajanje:

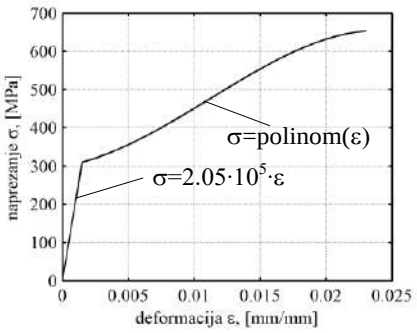
```
void ZbrajanjeMatrica(int A[100][100], int B[100][100], int n, int m, int C[100][100])
{
	int r, s;
	for (r = 0; r < n; r += 1)
		for (s = 0; s < m; s += 1)
			C[r][s] = A[r][s] + B[r][s];
}
```

Jednom gotove funkcije izrazito olakšavaju izradu novih programa. Glavna funkcija sada se svodi na pozivanje već gotovih funkcija:

```
int main()
{
	int A[100][100], B[100][100], C[100][100];
	int brRedaka, brStupaca;

	printf("Unesi broj redaka: ");
	scanf_s("%d", &brRedaka);
	printf("Unesi broj stupaca: ");
	scanf_s("%d", &brStupaca);

	UnosMatrice(A, brRedaka, brStupaca);
	UnosMatrice(B, brRedaka, brStupaca);
	ZbrajanjeMatrica(A, B, brRedaka, brStupaca, C);
	printf("Zbrojena matrica: \n");
	IspisMatrice(C, brRedaka, brStupaca);
	return 0;
}
```

5.7 Inženjerski primjeri

PRIMJER 3 Nelinearni dijagram naprezanja. Potrebno je napraviti korisnički definiranu funkciju za izračun naprezanja u ovisnosti o deformaciji. Funkcija je zadana u dva djela, prvi je dio linearan, drugi je dio zadan polinomom trećeg stupnja kao što je prikazano na donjoj slici. Linearni dio vrijedi do ε=0.0015, a nakon toga vrijedi polinom do ε=0.023 nakon čega dolazi do popuštanja [13], [14].



polinom(ε)=–4.35·10^7·ε^3+1.46·10^6·ε^2+4763·ε+300

Rješenje Dijagram se sastoji od tri djela. Prvi je linearna funkcija, drugi je polinom, a nakon granice popuštanja naprezanje se može uzeti jednako nuli. Za izradu korisnički definirane funkcije potrebno je koristiti kontrolu toka koja će ovisno o području vratiti naprezanje po drugačijoj zakonitosti. Prva verzija funkcije prikazana je u nastavku.

```
double NaprezanjeCelik(double x)
{
	//Granica proporcionalnosti(elasticno podrucje)
	double granicaProp = 0.0015;
	//Granica popuštanja
	double granicaPop = 0.023;
	double naprezanje;

	if(x < granicaProp)
			naprezanje = x * 2.07e5;
	else if(x < granicaPop)
		naprezanje = -4.35e7 * pow(x,3) + 1.46e6 * pow(x, 2) + 4763 * x + 300;
	else
		naprezanje = 0;
	return naprezanje;
}
```

PRIMJER 4 Potrebno je izračunati brzinu slobodnog pada padobranca za prvih 10 sekunda.

Rješenje U slučaju kada se zanemari otpor zraka, na padobranca djeluje samo sila težine $G=m\cdot g$. Brzina slobodnog pada može se izračunati iz drugog Newtonovog zakona [15]:

$$\sum Fz = m\cdot a \Rightarrow a = \sum Fz / m \Rightarrow a = (m\cdot g)/m = g = 9.81 \text{ m/s}$$

Integriranjem ovog izraza, dobije se brzina v:

$$v(t) = \int_0^t a(t)dt = \int_0^t -9.81dt + v_0$$

Prethodno rješenje ne uzima u obzir otpor zraka, koji se inače može aproksimirati izrazom $F_D=0.5\cdot\rho\cdot c_D\cdot A\cdot v^2$, gdje je c_D koeficijent otpora, ρ je gustoća zraka (1.225 kg/m^3), A je površina tijela projicirana u smjeru gibanja, a v je brzina. Za padobranca, može se uzeti približno A=1m^2 i c_D=1.0. Suma sila te ubrzanje sada su funkcija brzine:

$$a = \sum Fz / m = (F_D(v) + G) / m$$

U ovom slučaju može se pronaći rješenje običnom integracijom, stoga se ovdje koristi iterativna Eulerova metoda. S obzirom na to da je poznata promjena brzine a(t) u svakom trenutku, brzina u sljedećem trenutku v(t+Δt) se računa kao prethodna brzina v(t) plus promjena brzine u tom vremenu:

$$v(t+\Delta t) = v(t) + \Delta t \cdot a(t)$$

Zadatak se rješava iterativnim korištenjem prethodnog izraza:

```
double SumaSilaZaPadobranca(double v)
{
	double ro = 1.225, cD = 1, A = 1, masa = 80, g = -9.81;
	double Fd, G;
	G = masa * g;
	Fd = 0.5 * ro * cD * A * pow(v, 2);
	return Fd + g;
}
int main()
{
	double masa = 80;
	double t, v = 0, a = -9.81, deltaT = 0.1;

	printf("t[s]\tv[m/s]\ta[m/s^2]\n");
	for (t = 0; t <= 10; t = t + deltaT)
	{
		printf("%.2lf\t%.2lf\t%.2lf\n", t, v, a);
		a = SumaSilaZaPadobranca(v) / masa;
		v = v + deltaT * a;
	}
	return 0;
}
```

Ovaj se program sada može i proširiti i koristiti u slučajevima složenijih funkcija za izračun otpora zraka i drugih sila. Ispis rezultata za ovaj slučaj:

```
t[s]    v[m/s]  a[m/s^2]
0.00    0.00    -9.81
0.10    -0.98   -9.81
0.20    -1.96   -9.80
0.30    -2.94   -9.78
…
9.70    -35.48  -0.18
9.80    -35.50  -0.17
9.90    -35.52  -0.16
10.00   -35.53  -0.15
```

6 RAD S DATOTEKAMA

Potreba za povezivanjem rada programa s datotekama izrazito je važna u inženjerskoj praksi. Kod obrađivanja velikog skupa podataka iz mjerenja (temperature, tlakovi,…), rezultati mjerenja uglavnom se prvo pohranjuju u datoteke. Obrada se vrši naknadno, a prvi korak u programiranju je učitati podatke iz datoteke u program. Za ovo se koristi funkcija ***fscanf***. Funkcija je slična već korištenoj funkciji *scanf_s*. Još jedan česti inženjerski primjer rada s datotekama je izrada izvještaja o rezultatima proračuna. Za ispis izvještaja u obliku tekstualne datoteke koristi se funkcija ***fprintf***; koja je praktički jednaka funkciji *printf*, osim što se ispis odvija u specificiranoj datoteci.
Rad s datotekama može se podijeliti na učitavanje iz datoteka, ispis u novu datoteku i dodavanje podataka u postojeću datoteku. Datoteke mogu biti bilo kojeg tipa, a ovdje će se koristiti (.txt) tekstualne datoteke.

6.1 Ispis u datoteku

Jednostavni primjer rada s datotekama je ispis poruke u tekstualnu datoteku. Kod rada s datotekama, prvo je potrebno prijaviti varijablu (`FILE *nazivVarijable`) te otvoriti datoteku funkcijom ***fopen_s***. Funkcija prima tri argumenta: varijablu koja će biti pokazivač na datoteku, naziv datoteke te način na koji će se datoteka koristiti (*"w"* – za zapisivanje). Za ispis u datoteku, koristi se funkcija *fprintf* na potpuno isti način kao i funkcija *printf*. S tim da funkcija *fprintf* kao prvi argument prima pokazivač na željenu datoteku (varijabla *dat*):

```
#include <stdio.h>
int main()
{
    FILE* dat;

    fopen_s(&dat, "datoteka.txt", "w");
    fprintf(dat, "Ovo je moj %d. ispis :)",1);
    return 0;
}
```

Pokretanjem programa, u komandni prozor ispisuje se samo poruka o završetku programa:

```
Press any key to close this window . . .
```

Međutim, u radnoj mapi se generira datoteka *datoteka.txt*, u kojoj se nalazi navedeni ispis.

Način otvaranja Kod otvaranja datoteke s *fopen*_s, način rada može biti čitanje (*"r"*), pisanje (*"w"*), i dodavanje sadržaja u postojeću datoteku (*"a"*). Kada se u datoteku zapisuje, ako datoteka s navedenim nazivom ne postoji u radnoj mapi, tada će program kreirati novu datoteku. Ako datoteka postoji, program će prebrisati sadržaj datoteke.

Potencijalne pogreške kod učitavanja datoteke mogu se ispitati odmah prilikom otvaranja datoteke. Funkcija *fopen_s* vrati nulu kod uspješno otvorene datoteke, inače je nastala pogreška:

```
int provjera;
provjera=fopen_s(&dat, "datoteka.txt", "r");
if (provjera == 0)
    printf("Datoteka je uspjesno otvorena!\n");
else
    printf("Greska prilikom otvaranja!\n");
```

6.2 Učitavanje iz datoteke

Funkcija koja učitava jedan po jedan znak datoteke je ***fgetc***. Prilikom ovoga važno je znati da svaka datoteka završava s posebnom oznakom ***EOF*** (kratica za *end of file*).

PRIMJER 1 Potrebno je napraviti program koji ispisuje cjelokupni sadržaj datoteke „*datoteka.txt*“.

Rješenje je u nastavku.

```
#include <stdio.h>
int main()
{
    FILE* dat;
    int provjera;
    char c;

    provjera=fopen_s(&dat, "datoteka.txt", "r");
    if (provjera == 0)
        printf("Datoteka je uspjesno otvorena!\n");
    else
    {
        printf("Greska prilikom otvaranja!\n");
        return 1;
    }

    while (1)
    {
        c = fgetc(dat);
        if (c == EOF)
            break;
        else
            printf("%c", c);
    }

    return 0;
}
```

Ovim programom može se ispisati sadržaj bilo koje datoteke. Na primjer, ispisana je datoteka koja je generirana u prethodnom poglavlju:

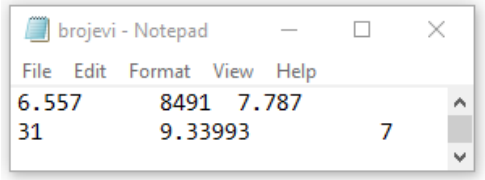
```
Datoteka je uspjesno otvorena!
Ovo je moj 1. ispis :)
Press any key to close this window . . .
```

U ovom programu, učitava se jedan po jedan znak datoteke pomoću funkcije *fgetc*. Ova funkcija prima pokazivač na datoteku, koji na početku „pokazuje“ na prvi znak datoteke. Kao rezultat funkcije dobiva se znak na koji pokazivač pokazuje, te se pokazivač pomiče na sljedeći znak. Ovo se može ponavljati sve dok nije učitan zadnji znak u datoteci, *EOF*. Preporučljivo je zatvoriti datoteku nakon korištenja pomoću funkcije ***fclose(dat).***

UČITAVANJE BROJEVA Kod učitavanja cijelih ili realnih brojeva koji su pohranjeni u datotekama, koristi se funkcija ***fscanf_s***. Ova se funkcija koristi slično kao i funkcija *scanf_s*, s tim da se prvo navodi datoteka iz koje se učitavaju vrijednosti. U najjednostavnijem slučaju, *fscanf_s* se može, slično kao i *fgetc*, koristiti za učitavanje jednog znaka: `fscanf_s(dat, "%c", &c);`. Ako se u datoteci nalaze brojevi, tada se za učitavanje cijelih brojeva koristi `fscanf_s(dat, "%d", &broj)`, a za realne `fscanf_s(dat, "%lf",`

`&broj)`. Kod učitavanja brojeva, program učitava sve brojeve do razmaka. Prije učitavanja broja funkcija preskače razmake (bez obzira na broj razmaka, čak i ako se radi o novoj liniji *\n* ili tabulatoru *\t*).

Za učitavanje prva dva broja iz sljedeće datoteke koriste se dva poziva funkcije *fscanf_s*:




```
FILE* dat;
int cijeliBroj;
double broj;
fopen_s(&dat, "brojevi.txt", "r");
fscanf_s(dat, "%lf", &broj);
fscanf_s(dat, "%d", &cijeliBroj);
printf("Realni: %lf Cijeli: %d", broj, cijeliBroj);
```

Ispis:

```
Realni: 6.557000 Cijeli: 8491
```

UČITAVANJE MATRIČNIH PODATAKA Za primjer učitavanja matrice, učitat će se datoteka s dva stupca i pet redaka:

```
0.0    2.3
0.5    3.32
1.15   4.54
2.35   5.65
3.11   7.51
```

Iako se brojevi u redcima mogu učitavati jedan po jedan, može se koristiti i `fscanf_s(dat, "%lf %lf",…)`, za učitavanje dva broja istovremeno. Nakon učitavanja svakog retka koristi se funkcija *fgetc(dat)* za provjeru trenutnog znaka. Ako je on jednak znaku za kraj datoteke *EOF*, program izlazi iz petlje.

```
FILE* dat;
double stupacA[10],stupacB[10];
int i,brojac;
fopen_s(&dat, "brojevi.txt", "r");
for (brojac = 0; brojac <10; brojac++)
{
    fscanf_s(dat, "%lf %lf", &stupacA[brojac], &stupacB[brojac]);
    if (fgetc(dat) == EOF)
        break;
}
for(i=0;i<brojac;i++)
    printf("%lf %lf\n", stupacA[i], stupacB[i]);
```

UČITAVANJE ZAGLAVLJA Često datoteke imaju nekoliko linija zaglavlja, koje nije potrebno učitavati. Za preskočiti cijelu liniju datoteke (do oznake za novi red '*\n*') može se koristiti petlja koja učitava znak po znak (*fgetc*) dok ne dođe do oznake za novu liniju:

```
while (1)
{
	if (fgetc(dat) == '\n')
		break;
}
```

REWIND Ako se datoteka unutar istog programa treba više puta učitavati od početka, može se koristiti funkcija ***rewind(dat)***. Ova funkcija „premotava“ datoteku na početak, tj. postavlja pokazivač na početak datoteke kao da je iznova otvorena. Funkciji se šalje pokazivač na datoteku, a nakon toga se učitavanje iz datoteke može raditi na isti način kao i ranije.

6.3 Zadaci za vježbu

1. Potrebno je napraviti program koji učitava matricu iz datoteke oblika:

```
n m
a11    a12    …      a1m
a21    a22    …      a2m
…      …      …      …
an1    an2    …      anm
```

Dakle, datoteka sadržava matricu veličine *nxm*, a broj redaka i stupaca definirani je u prvom retku. Broj redaka i stupaca u općem slučaju nije unaprijed poznat. Za konkretni primjer možete koristiti datoteku:
`datotekaMatrica34.txt`

```
3 4
0.81 0.91 0.27 0.96
0.90 0.63 0.54 0.15
0.11 0.09 0.95 0.97
```

2. Potrebno je napraviti program koji učitava donju trokutastu matricu iz datoteke oblika:

```
n m
a11
a21    a22
…      …      …
an1    an2    …      anm
```

Elemente iznad glavne dijagonale potrebno je postaviti na nulu. Primjer datoteke:
`donjaTrokutasta.txt`

```
4 4
0.9572
0.4854    0.9157
0.8003    0.7922    0.8491
0.1419    0.9595    0.9340    0.3922
```

3. Zadana je datoteka koja sadržava tri stupca, općenito oblika:

```
i       j       M(i,j)
…       …       …
```

Prva dva stupca predstavljaju indekse retka i stupca, a treći stupac predstavlja vrijednost pripadnog elementa matrice. Potrebno je napraviti program koji učitava datoteku u matricu, a elemente koji nisu definirani postaviti na nulu. Koristite za primjer sljedeću datoteku:

`sparseMatrice.txt`

```
1       2       3.23
2       1       1.15
2       2       0.54
3       3       0.99
```

4⁺. Potrebno je učitati dva stupca iz datoteke koja sadržava zaglavlje prije podataka. Prvih nekoliko redaka datoteke prikazano je ispod, a punu datoteku možete pronaći među online primjerima. Datoteka je dobivena iz automobilskog računala za vrijeme vožnje. Potrebno je učitati podatke te ih ispisati.

`brojOkretajaIPotrosnja.txt`

```
Zaglavlje sadržava 5 redaka
Nakon zaglavlja, datoteka u dva stupca sadržava podatke:
Broj okretaja, [RPM]      Potrošnja goriva, [l/100km]
Oznaka decimalnog broja: točka '.'
Odvajanje stupaca: tabulator '\n'
1228   256
1568.34      256
...
```

5⁺. Potrebno je učitati podatke o izmjerenim temperaturama iz datoteke koja sadržava jedan stupac s brojevima.

`mjerenjaTemperature.txt`

```
88,71
88,82
89,09
89,37
88,85
...
```

6.4 Riješeni primjeri

Zadaci označeni znakom + su riješeni.

ZADATAK 4 Potrebno je učitati dva stupca iz datoteke koja sadržava zaglavlje prije podataka. Prvih nekoliko redaka datoteke prikazano je ispod, a punu datoteku možete pronaći među online primjerima. Datoteka je dobivena iz automobilskog računala za vrijeme vožnje. Potrebno je učitati podatke te ih ispisati.

`brojOkretajaIPotrosnja.txt`

```
Zaglavlje sadržava 5 redaka
Nakon zaglavlja, datoteka u dva stupca sadržava podatke:
Broj okretaja, [RPM]      Potrošnja goriva, [l/100km]
Oznaka decimalnog broja: točka '.'
Odvajanje stupaca: tabulator '\n'
1228   256
1568.34      256
...
```

Rješenje U nastavku je prikazan program koji učitava prethodnu datoteku.

```
int main()
{
	FILE* dat;
	int i, brMjerenja,provjera;
	double rpm[10000], potrosnja[10000];
	fopen_s(&dat,"brojOkretajaIPotrosnja.txt", "r");
	// Učitavanje(preskakanje) 5 redaka zaglavlja
	for (i = 0; i < 5; i++)
		while (1)
		if (fgetc(dat) == '\n')
			break;

	i = 0;
	do
	{
		//Učitavanje dva stupca u dva vektora:
		fscanf_s(dat, "%lf %lf",&rpm[i], &potrosnja[i]);
		i++;
	} while (fgetc(dat) != EOF);
	brMjerenja = i;
	printf("Ucitano je %d mjerenja!", brMjerenja);
	printf("Broj okretaja [RPM]\tPotrosnja [l/100km]\n");
	for (i = 0; i < brMjerenja; i++)
		printf("%lf\t%lf\n", rpm[i], potrosnja[i]);

	return 0;
}
```

Ispis rezultata programa:

```
Ucitano je 507 mjerenja!
Broj okretaja [RPM]     Potrosnja [l/100km]
1228.000000             256.000000
1215.240000             81.380000
1307.920000             39.090000
…
```

ZADATAK 5 Učitavanje podataka s drugačijom decimalnom oznakom. Potrebno je učitati podatke o izmjerenim temperaturama iz datoteke koja sadržava jedan stupac s brojevima.

`mjerenjaTemperature.txt`

```
88,71
88,82
89,09
89,37
...
```

Rješenje Ovo bi inače bilo jednostavno, ali problem je što je za oznaku decimalnog broja korišten zarez umjesto točke. Za riješiti problem, datoteka će se učitati kao niz znakova, pri čemu se svi zarezi zamjenjuju točkom te se rezultat zapisuje u novu datoteku. Nakon što je datoteka ispravljena, potrebno ju je zatvoriti korištenjem funkcije *fclose*. Zatim se datoteka može ponovno otvoriti te s njom raditi na uobičajen način.

```
int main()
{
	FILE *dat, *datIspravak;
	double temperature[1000];
	int i;
	char znak;

	fopen_s(&dat,"mjerenjaTemperature.txt","r");
	fopen_s(&datIspravak, "mjerenjaTempIspravak.txt", "w");
	do
	{
		znak = fgetc(dat);
		if (znak == ',')
			znak = '.';
		fprintf(datIspravak, "%c", znak);
	} while (znak != EOF);

	fclose(datIspravak);
	fopen_s(&datIspravak, "mjerenjaTempIspravak.txt", "r");
	printf("Ucitana mjerenja: \n");
	i = 0;
	do
	{
		fscanf_s(datIspravak, "%lf", &temperature[i]);
		printf("%lf\n", temperature[i]);
		i++;
	} while (feof(datIspravak)==0);

	return 0;
}
```

Rezultat programa su uspješno učitane temperature u polje realnih brojeva:

```
Ucitana mjerenja:
88.710000
88.820000
89.090000
...
```

6.5 Inženjerski primjeri

PRIMJER 2 Učitavanje podataka o vožnji iz ASCII datoteke s definiranim odvajanjem.

U datoteci *podaciOVoznji.csv* koju možete pronaći među online primjerima, nalaze se rezultati raznih mjerenja za vrijeme vožnje automobila. Ovakav oblik datoteke (*.csv*) često se javlja kod rada s eksperimentalnim mjerenjima ili bazama podataka. S obzirom na to da nije u potpunosti standardiziran, može biti potrebno raditi (pred-)programe za učitavanje. Ispod su prikazana prva 4 retka datoteke. Redni brojevi retka naznačeni su u prikazu ispod, kako bi se naglasilo što sve pripada prvom retku. Kod ovakvih datoteka prvi redak sadržava nazive stupaca, a stupci su odvojeni s posebnom vrstom oznake, u ovom slučaju točka-zarez (;).

`podaciOVoznji.csv`

```
1  Vrijeme;Opterećenje, [%];Temperatura hladila, [°C];Tlak u usisu,
   [kPa];Broj okretaja, [RPM];Brzina, [km/h];Otvorenost klapne,
   [%];Potrošnja goriva, [l/100km]
2  14.3.2017 10:19;46.44;88.71;60.19;1228;0;20.48;256
3  14.3.2017 10:19;53.6;88.82;55.93;1568.34;0;23.04;256
4  14.3.2017 10:19;53.86;89.09;58.86;1579.2;2;20.85;256
…  …
```

Zadatak je učitati sve podatke, te ispisati učitane rezultate.

Rješenje Riješeni program prikazan je na sljedećoj stranici. Svi su podaci učitani u zasebni stupac matrice *var*, a nazivi su učitani kao znakovni nizovi u varijablu *nazivi*. Pokretanjem programa, ispisuju se nazivi stupaca, a zatim i sve učitane vrijednosti:

```
Naziv varijable var[0]: Vrijeme
Naziv varijable var[1]: Opterecenje, [%]
Naziv varijable var[2]: Temperatura hladila, [°C]
Naziv varijable var[3]: Tlak u usisu, [kPa]
Naziv varijable var[4]: Broj okretaja, [RPM]
Naziv varijable var[5]: Brzina, [km/h]
Naziv varijable var[6]: Otvorenost klapne, [%]
Naziv varijable var[7]: Potrosnja goriva, [l/100km]
Ispis ucitanih rezultata:
46.4    88.7    60.2    1228.0  0.0     20.5    256.0
53.6    88.8    55.9    1568.3  0.0     23.0    256.0
...
28.2    85.0    36.0    2266.7  76.9    20.0    4.3
28.0    84.7    36.0    2278.3  77.8    19.6    4.4
27.9    85.1    35.1    2294.7  78.0    19.3    4.2
...
```

```c
int main()
{
	FILE *dat;
	char nazivi[8][100]; //8 znakovnih nizova nazivi[0],nazivi[1],...
	double var[8][3000]; //8 polja realnih brojeva var[0],var[1],...
	char znak;
	int i, j,brojVarijabli,brojMjerenja;
	//Učitavanje naziva varijabli definiranih u zaglavlju
	fopen_s(&dat,"podaciOVoznji.csv","r");
	i = 0;
	do
	{
		for (j = 0; j < 100; j++)
		{
			znak = fgetc(dat);
			if (znak != ';' && znak != '\n')
				nazivi[i][j] = znak;
			else
			{	//Na kraju znakovnog niza, treba postaviti nul-znak
				nazivi[i][j] = '\0';
				break;
			}
		}
		i++;
	} while (znak != '\n');
	brojVarijabli = i;
	//Ispis učitanih naziva varijabli
	for (i = 0; i < brojVarijabli; i++)
		printf("Naziv varijable var[%d]: %s\n", i, nazivi[i]);
	i = 0;
	while(1)//Petlja za učitavanje mjerenja u polja
	{
		do //Vrijeme se ne ucitava, petlja za preskakanje
		{
			znak = fgetc(dat);
		} while (znak != ';' && znak != EOF);
		//"&& znak != EOF" zatvara gornju petlju na kraju datoteke
		if (znak == EOF)//Zatvara vanjsku petlju na kraju datoteke
			break;
		//Učitavanje ostalih varijabli:
		for (j = 1; j < brojVarijabli; j++)
		{
			fscanf_s(dat, "%lf ", &var[j][i]);
			znak = fgetc(dat);//Učitavanje ";"
		}
		i++;
	}
	brojMjerenja = i;
	printf("Ispis ucitanih rezultata: \n");
	for (j = 0; j < brojMjerenja; j++)
	{
		for (i = 1; i < brojVarijabli; i++)
			printf("%.1lf\t", var[i][j]);
		printf("\n");
	}
	return 0;
}
```

7 NUMERIČKE METODE

Ovo će poglavlje pokazati nekoliko numeričkih postupaka koji su bitni za rješavanje inženjerskih problema. Numerički postupci u ovom poglavlju odnose se na funkcije s jednom varijablom. Pokazat će se metode za numeričko rješavanja problema: integracije, numeričke derivacije, interpolacije, rješavanja nelinearnih jednadžbi i traženja minimuma ili maksimuma funkcije. Osim ovoga, ukratko je objašnjen problem rješavanja linearnih sustava jednadžbi.

Specifičnost numeričkih postupaka je primjenjivost na širokom spektru problema. Numerički postupci rješavaju inače teške matematičke probleme uzastopnim ponavljanjem jednostavnih aritmetičkih operacija. Često se numerički postupci svode na iterativno ponavljanje nekoliko jednostavnih koraka sve dok je točnost aproksimacije veća od neke zadane:

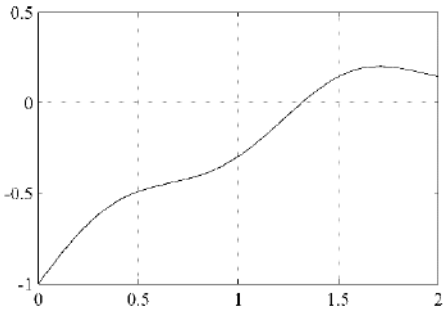
```
tol=1e-5;
do
{
	KoraciNumerickeMetode;
	tocnost=IzracunTocnostiAproksimacije;
}while(fabs(tocnost)>tol);
IspisRezultata;
```

Tolerancija je najčešće neki vrlo mali broj, u ovom primjeru 10^{-5}. Treba paziti da tolerancija ne bude previše mala, jer računalo realne brojeve zapisuje do određenog broja decimala (*float* 6, *double* 15). Da bi se mogla kontrolirati konvergencija metode, potrebno je imati neki izračun točnosti aproksimacije. Nakon što se izvrše koraci numeričke metode, računa se točnost. Petlja se ponavlja dok je apsolutna vrijednost točnosti veća od tolerancije. Ovakav algoritam koriste metode za rješavanje nelinearnih jednadžbi i traženje minimuma ili maksimuma funkcije. Na sličan način provode se i postupci numeričke integracije i derivacije.

Test funkcija Za ispitivanje metoda numeričke derivacije, integracije, rješavanje nelinearne jednadžbe i traženje ekstrema bit će potrebne neke testne funkcije. Ovdje će se koristiti funkcija $y=x-1-0.2*x^2+0.1*\sin(5*x)$:

```
double Fun(double x)
{
	return -0.2*x*x+x-1+0.1*sin(5*x);
}
```

Sljedeća slika prikazuje graf funkcije u rasponu od 0 do 2.

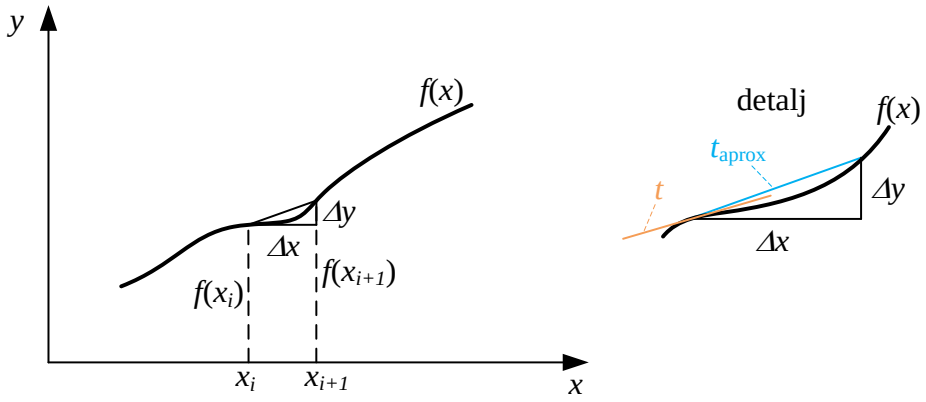


7.1 Numerička derivacija

Za izračun numeričke derivacije najjednostavnija je metoda diferencije unaprijed. Ova je metoda aproksimacija matematičke definicije derivacije kao što je prikazano u nastavku. Kod matematičke definicije, Δx teži u nulu, dok se kod numeričke derivacije koristi konačna veličina Δx:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x} = \frac{\Delta y}{\Delta x} \tag{7.1}$$

Dakle, za aproksimaciju vrijednosti derivacije u točci x, potrebno je izračunati vrijednost funkcije u toj točci kao i u točci odmaknutoj za Δx. Iz toga se može izračunati vrijednost Δy (razlika visine). Derivacija se zatim računa kao omjer promjene visine funkcije s pomakom Δx. Ovo predstavlja nagib pravca tangente na funkciju u točci x kao što je prikazano na donjoj slici.



Slika 7.1 Izračun numeričke derivacije i detalj s razlikom nagiba stvarne tangente i tangente dobivene s aproksimacijom nagiba prema vrijednosti numeričke derivacije.

Numerička derivacija za funkciju *Fun* svodi se na jednu liniju programa:

```
derivacija=(Fun(x+deltaX)- Fun(x))/deltaX;
```

Za ranije definiranu funkciju *Fun*, numerička derivacija može se računati korištenjem funkcije:

```
double DerivacijaFun(double x)
{
	double derivacija, deltaX;
	deltaX = 0.001;
	derivacija = (Fun(x + deltaX) - Fun(x)) / deltaX;
	return derivacija;
}
```

Pitanja za provjeru znanja 7.1

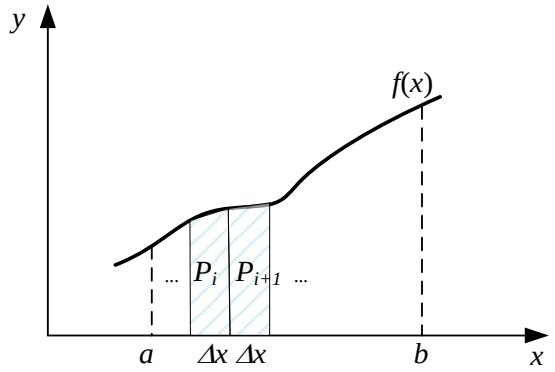
- Analitički odredi derivaciju funkcije $f(x)=\sin(x)$. Usporedi rezultat analitičke derivacije i numeričke za $f'(0.5)$. Nije potrebno raditi C-program, dovoljan je kalkulator. Je li rezultat jednak u oba slučaja?
- Što će biti rezultat za slučaj funkcije $f(x)=7x+2$?
- Kako povećati točnost numeričke derivacije?
- Što ako se uzme *deltaX*=10^{-20} ? Objasnite rezultat.

7.2 Numerička integracija

Numerička integracija koristi se za izračun određenog integrala odnosno površinu ispod krivulje u zadanom intervalu. Opći izraz za numeričku integraciju funkcije s jednom varijablom je:

$$\int_a^b f(x)dx \approx \sum_i P_i \tag{7.2}$$

gdje su a i b donja i gornja granica intervala, f je funkcija koja se integrira a P_i su površine pod-intervala koje se računaju nekom od metoda numeričke integracije. Shematski prikaz kako se odvija numerički postupak integracije prikazan je na donjoj slici. Interval između donje i gornje granice (a i b), se podijeli na proizvoljan broj pod-intervala širine Δx. Najčešće se uzima da je širina svakog pod-intervala jednaka. Svakom se podintervalu zatim računa površina P_i, nekom aproksimacijskom metodom. Najjednostavnije bi bilo uzeti vrijednost funkcije na sredini intervala, te računati površinu pod-intervala umnoškom širine Δx i visine (vrijednosti funkcije na sredini pod-intervala).

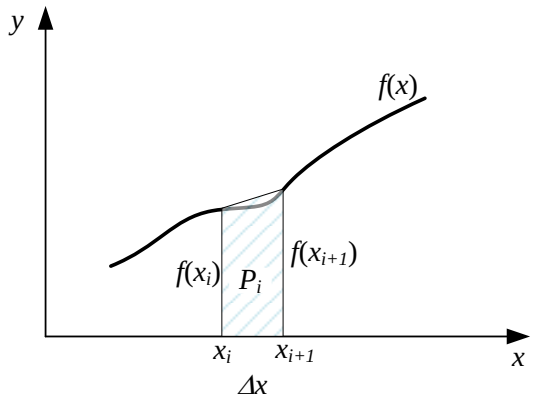


Slika 7.2 Numerička integracija

7.2.1 Integracija trapeznim pravilom

Metoda koja daje točnije rješenje od prethodno navedene naziva se numerička integracija trapeznim pravilom. Kod ove metode za svaki pod-interval se računa vrijednost funkcije na oba kraja pod-intervala. Površina pod-intervala zatim se aproksimira trapezom (vidi sliku ispod) širine Δx i visinama stranica jednakima vrijednostima funkcije na krajevima:

$$P_i = \Delta x \cdot \frac{f(x_i) + f(x_{i+1})}{2} \tag{7.3}$$



Slika 7.3 Izračun površine trapeznim pravilom

Funkcija za izračun određenog integrala funkcije *Fun* s podjelom funkcije na 1000 pod-intervala je prikazana u nastavku. Ulazni parametri su donja i gornja granica integracije, a rezultat je aproksimacija određenog integrala u zadanim granicama. Točnost funkcije može se povećati smanjivanjem širine pod-intervala, što se jednostavno postiže povećavanjem nazivnika 1000 (na npr. 10000).

```
double OdredeniIntegralFun(double donja, double gornja)
{
	double integral, sirinaIntervala, x;
	sirinaIntervala = (gornja - donja) / 1000;
	integral = 0.0;
	for (x = donja; x<gornja; x = x + sirinaIntervala)
	{
		integral = integral + (Fun(x) + Fun(x + sirinaIntervala)) / 2 * sirinaIntervala;
	}
	return integral;
}
```

Funkcija za integriranje može se jednostavno koristiti kao što je prikazano u nastavku:

```
int main()
{
	printf("Integral funkcije Fun (2 do 3): %lf\n", OdredeniIntegralFun(2,3));
	return 0;
}
```

Rezultat funkcije je ispis vrijednosti integrala:

```
Integral funkcije Fun (2 do 3): 0.244139
```

Opća funkcija za numerički integral Općenitije, integracija se može provesti nad poljem koordinata funkcije. Ovim se postiže to da funkcija nije ograničena na integriranje funkcije *Fun*.

```
double OdredeniIntegralPolja(double x[], double y[],int n)
{
	double integral=0.0;
	int i;
	for (i = 0; i < (n - 1); i++)
		integral = integral + (y[i] + y[i + 1]) / 2 * (x[i + 1] - x[i]);
	return integral;
}
```

U glavnoj funkciji, program za izračun integrala sada izgleda nešto složenije, jer je prvo potrebno koordinate točaka funkcije pohraniti u polje, a nakon toga slijedi poziv funkcije za integriranje.

```
int main()
{
	double donjaGranica = 2, gornjaGranica = 4, korak = 0.25;
	double x, poljeX[1000], poljeY[1000];
	int i=0;
	for (x = donjaGranica; x <= gornjaGranica; x = x + korak)
	{
		poljeX[i] = x;
		poljeY[i] = Fun(x);
		i++;//Brojac tocaka
	}
	printf("Integral iznosi: %lf\n",OdredeniIntegralPolja(poljeX,poljeY,i));
	return 0;
}
```

Primjer tabličnog rješavanja Potrebno je izračunati određeni integral funkcije $f(x)=x-1-0.2*x^2+0.1*\sin(5*x)$, u granicama [2,4]. Prikazom postupka u obliku tablice dobiva se uvid u korake numeričkog postupka. U prvom stupcu navedene su vrijednosti x u zadanim granicama. Odabrana širina intervala je $\Delta x=0.2$. U drugom su stupcu pripadne vrijednosti za y. U trećem je stupcu koeficijent koji množi funkciju, koji za trapezno pravilo iznosi $c=0.5$ prema izrazu (7.3). Koeficijent $c=0.5$ koristi se samo za prvu i zadnju točku u tablici. Za ostale točke koristi se koeficijent $c=1$, iz razloga što se ove točke ponavljaju u susjednim intervalima (0.5+0.5=1). Četvrti se stupac računa kao umnožak $\Delta x*y*c$, a suma četvrtog stupca predstavlja numerički izračunat integral koji iznosi **0.231**. Za usporedbu, kod prethodnog c-programa s 1000 podintervala ($\Delta x=0.001$) rezultat je bio **0.244**.

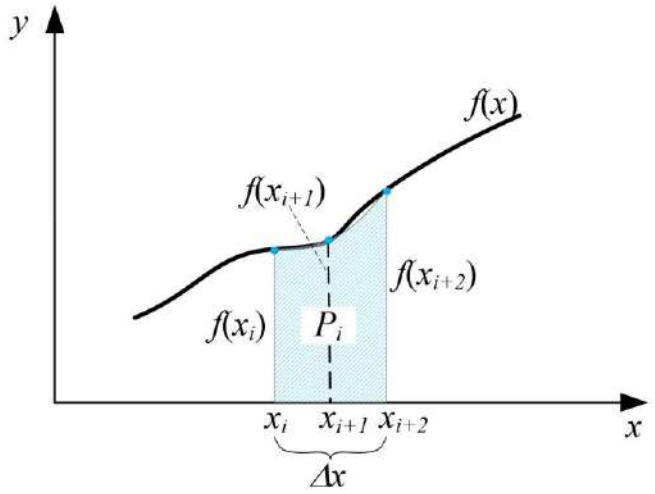
Δx=0.1			
x	***y***	***c***	***Δx*y*c***
2	0.15	0.5	0.007
2.1	0.13	1	0.013
2.2	0.13	1	0.013
2.3	0.15	1	0.015
2.4	0.19	1	0.019
2.5	0.24	1	0.024
2.6	0.29	1	0.029
2.7	0.32	1	0.032
2.8	0.33	1	0.033
2.9	0.31	1	0.031
3	0.27	0.5	0.013
		Sum:	**0.231**

7.2.2 Integracija Simpsonovim pravilom

Još točnija aproksimacija površine (kod glatkih funkcija) dobije se korištenjem Simpsonovog pravila. Integracija Simpsonovim pravilom svaki pod-interval aproksimira s površinom ispod parabole interpolirane kroz oba kraja intervala (x_i i $x_{i+2}=x_i+\Delta x$) i točku na sredini intervala ($x_{i+1}=x_i+\Delta x/2$). Izraz za izračun površine intervala širine Δx Simpsonovim pravilom je:

$$P_i = \Delta x \cdot \frac{f(x_i)+4\cdot f(x_{i+1})+f(x_{i+2})}{6} \qquad (7.4)$$

Izračun jednog dijela površine Simpsonovim pravilom shematski je prikazan na sljedećoj slici



Slika 7.4 Izračun površine Simpsonovim pravilom

Pitanja za provjeru znanja 7.2

- Analitički odredi integral funkcije $f(x)=|x|$ od -1 do 1. Koja će numerička metoda dati točniji rezultat (trapezno pravilo ili Simpsonovo)?
- Isto pitanje za integral funkcije $f(x)=x^2$ od -1 do 1.
- Može li trapezno pravilo dati pogrešan rezultat za kod integracije funkcije $f(x)=|x|$. Ako da, u kojem slučaju?
- Može li Simpsonovo pravilo dati pogrešan rezultat za kod integracije funkcije $f(x)= x^2$. Ako da, u kojem slučaju?
- Koja će metoda biti točnija za funkciju ispod?

$$f(x)=\begin{cases} -x, \text{za } x<0 \\ x^2, \text{za } x \geq 0 \end{cases}$$

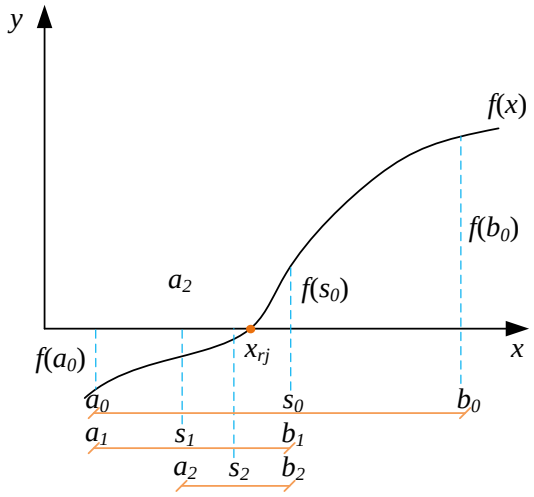
7.3 Rješavanje nelinearnih jednadžbi

Nelinearne jednadžbe s jednom varijablom mogu se svesti na oblik $f(x)=0$. U nastavku će se pokazati dvije metode za rješavanje ovog problema: metoda polovljenja intervala i Newtonova metoda.

7.3.1 Metoda polovljenja intervala

Metoda polovljenja intervala (još se naziva metoda bisekcije) traži nultočku unutar zadanog intervala omeđenog točkama a i b. Prvi korak metode je provjeriti nalazi li se uopće nultočka unutar intervala. Ako su predznaci vrijednosti $f(a)$ i $f(b)$ različiti, nultočka se sigurno nalazi unutar intervala (ako je funkcije neprekidna). Predznaci su različiti ako je zadovoljen logički uvjet $f(a)\cdot f(a)<0$, odnosno umnožak vrijednosti funkcije u točkama a i b je manji od nule. Korištenjem ove provjere veličina intervala se može smanjivati i provjeravati je li nultočka unutar nekog intervala manje veličine. Kod metode polovljenja intervala, između točke a i b napravi se središnja točka $s=(a+b)/2$. Sada se može zasebno provjeravati nalazi li se nultočka unutar intervala $[a,s]$ i $[s,a]$. Rezultat ove provjere može biti da se nultočka nalazi u

lijevom intervalu, u desnom intervalu ili da je nultočka baš u točci *s*. Zatim je moguće suziti interval (izmijeniti vrijednost *a* ili *b*). Ovaj se postupak ponavlja nekoliko puta kao što je prikazano na donjoj slici. Za provjeru točnosti i uvjet ponavljanja može se uzeti širina trenutnog *b*-*a*, ili visina funkcije u srednjoj točci *f*(*s*).



Slika 7.5 Shematski prikaz metode polovljenja intervala.

Za izračun nultočke metodom polovljenja koristi se funkcija koja prima gornju i donju granicu, a vraća kao rezultat nultočku (*x*-koordinatu):

```
double MetodaPolovljenjaFun(double a, double b)
{
	double s=(a+b)/2,fa,fb,fs,tol=0.00001;
	fa = Fun(a);
	fb = Fun(b);
	if (fa*fb>0)
	{
		printf("Nema nultocke !\n");
		return s;
	}
	do
	{
		fa = Fun(a); fb = Fun(b);
		s = (a + b) / 2; fs = Fun(s);
		if (fa*fs<= 0.0)
			b = s;
		else
			a = s;
	} while (fabs(fs)>tol);
	return s;
}
```

Funkcija se poziva tako da joj se pošalje donja i gornja granica (0 i 2) unutar kojih se traži nultočka:

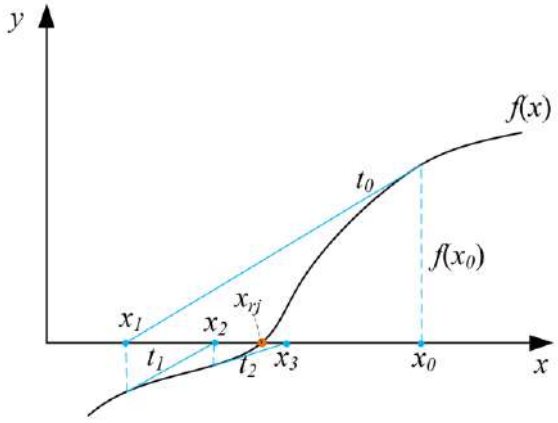
```
int main()
{
	double x, f;
	x= MetodaPolovljenjaFun(0, 2);
	f=Fun(x);
	printf("Nultocka se nalazi u x=%.3lf, f(%.3lf)=%lf',x,x,f);
	return 0;
}
```

Pokrenite program, a zatim se ispisuje rezultat:

```
Nultocka funkcije je: x=1.317, f(1.317)=0.000003
```

7.3.2 Newtonova metoda

Newtonova metoda načelno je brža od prethodne, odnosno potreban je manji broj koraka dok dođe do rješenja iste točnosti. Za razliku od prethodne metode, Newtonova metoda kreće iz jedne točke (x_0), koja se proizvoljno unosi. Program će konvergirati brže ako je početna točka blizu nultočke. Ako je funkcija nepravilnog oblika i početna nultočka nije dobro odabrana, ova metoda može divergirati. Prednost ove metode u odnosu na metode polovljenja je što rješava problem u manji broj iteracija, ali nedostatak je moguća divergencija koja se ne može dogoditi kod metode polovljenja. Newtonova metoda u svakom koraku funkciju aproksimira s pravcem koji je tangenta na funkciju kroz točku (x,$f(x)$), kao što je prikazano na slici ispod. Jednadžba pravca može se odrediti iz točke i nagiba dobivenog numeričkom derivacijom. Znači u prvom koraku funkcija se aproksimira pravcem t_0 kao što je označeno na donjoj slici. Zatim se pronalazi nultočka pravca t_0, koja je na slici označena s x_1. Ova se točka zatim koristi za izraditi novi tangentni pravac i tako dalje.



Prethodni postupak svodi se na izraz:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{7.5}$$

Za izračun nultočke Newtonovom metodom može se koristiti funkcija za izračun derivacije iz prethodnog poglavlja. Funkcija koja računa nultočku (x), a prima početno rješenje Newtonove metode:

```
double NewtonovaNultocka(double x)
{
	double tol = 0.00001;
	do
	{
		x = x - Fun(x) / DerivacijaFun(x);
	} while (fabs(Fun(x))>tol);
	return x;
}
```

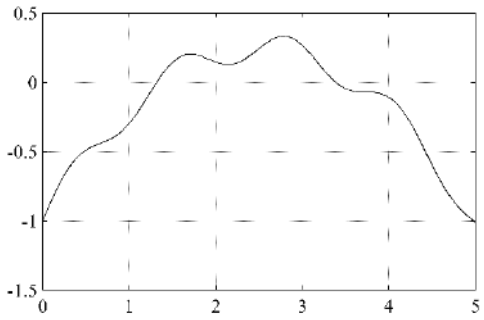
Funkcija se poziva tako da joj se pošalje početna aproksimacija:

```
int main()
{
	double x0,x,f;
	printf("Unesite početno rješenje: ");
	scanf_s("%lf",&x0);
	x= NewtonovaNultocka(x0);
	f=Fun(x);
	printf("Nultocka se nalazi u x=%.3lf, f(%.3lf)=%lf',x,x,f);
	return 0;
}
```

Pokrenimo program a za početnu točku uzmimo nulu. Pokretanjem programa dobije se isto rješenje kao i kod metode polovljenja:

```
Unesi početnu točku: 0
Nultocka se nalazi u x=1.317, f(1.317)=-0.000004
```

Treba paziti na to da funkcije mogu sadržavati više nul-točki. Ako nacrtamo graf funkcije u malo širem intervalu, očito je da postoji još jedna nultočka koja se nalazi između 3.0 i 4.0.



Slika 7.6 Graf nelinearne test funkcije

Pokretanjem programa iz početne točke u blizini druge nultočke, dobit ćemo:

```
Unesi početno rješenje: 5
Nultocka se nalazi u x=3.384, f(3.384)=0.000001
```

Isti bi se rezultat dobio metodom polovljenja intervala da se odabrao interval npr. između 2 i 5.

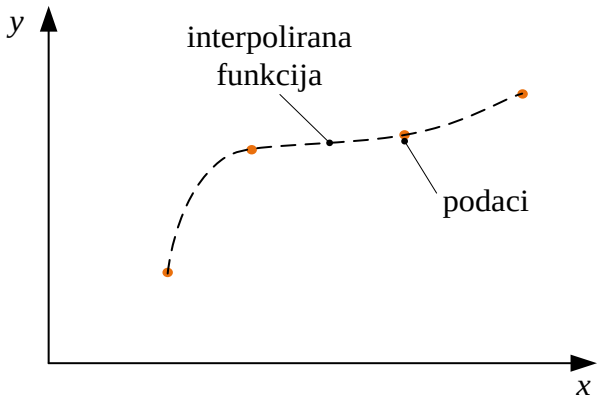
Pitanja za provjeru znanja 7.3

- Koju biste metodu i inicijalizaciju odabrali za rješavanje nelinearne jednadžbe $\ln(x) = -1/2$ (pazite na domenu funkcije)?
- Može li se dogoditi da Newtonova metoda u prethodnom slučaju „izađe“ iz domene funkcije? Ako da, pod kojim uvjetima?

- Koju biste metodu i inicijalizaciju odabrali za rješavanje nelinearne jednadžbe $\ln(-x+1)\sqrt{x}=-1/2$ (pazite na domenu funkcije)?
- Za funkciju sin(1/x), potrebno je pronaći sve nultočke između x=0.01 i x=0.1. Koju metodu odabrati?

7.4 Interpolacija

Kada je zadan skup točaka iz kojeg je potrebno dobiti kontinuiranu funkciju, koriste se metode interpolacije ili aproksimacije. Metodama interpolacije dobije se funkcija koja prolazi kroz sve točke, a metodama aproksimacije dobije se funkcija željenog oblika (npr. polinom trećeg stupnja) koja približno prolazi kroz skup točaka. Interpolacija se više koristi kod malog broja podataka. Ako postoji velik skup podataka interpolacija može stvarati probleme, stoga se češće koriste metode aproksimacije. Primjera primjene ovih metoda ima mnogo za skupove 2D, 3D i nD podataka. Na primjer, pri ispitivanju materijala na vlačno opterećenje rezultat mjerenja je skup deformacija i pripadnih naprezanja. Za dobiti funkciju ovisnosti naprezanja o deformaciji, potrebno je koristiti metode aproksimacije i/ili interpolacije. Općenito rečeno, postoji skup točaka s x i y koordinatama, a traži se funkcija $y=f(x)$.
Interpolacija je postupak konstruiranja funkcije koja točno prolazi kroz skup točaka. Na primjer, na donjoj slici funkcija prolazi kroz četiri točke.



Slika 7.7 Interpolacija

7.4.1 Lagrangeov interpolacijski polinom

Ako postoje samo dva podatka odnosno samo dvije točke u prostoru, kroz njih se može provući pravac. Ovaj se postupak može nazvati interpolacijom polinoma prvog stupnja. Ako su zadane tri točke kroz njih se može provući parabola odnosno polinom drugog stupnja. Općenito, ako je zadano n točaka, kroz njih se može interpolirati polinom stupnja n-1. Za ovo se koristi Lagrangeov interpolacijski polinom. Ako je zadano n točaka i vektori koordinata $\boldsymbol{X}$ i $\boldsymbol{Y}$ s članovima x_i i y_i, kroz njih se može provući polinom:

$$P(x)=\sum_{j=1}^{n}\left[y_j\cdot\prod_{\substack{k=1\\k\neq j}}^{n}\frac{x-x_k}{x_j-x_k}\right] \tag{7.6}$$

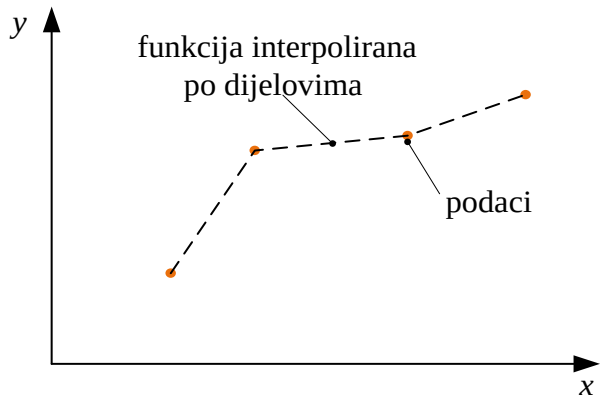
Ovim izrazom ukoliko se razloži dobije se polinom tipa $P(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+...+a_1x+a_0$. U slučaju da je *n*=2, finalni polinom bit će oblika $P(x)=ax+b$ iako to nije direktno vidljivo iz prethodnog izraza. S obzirom na to da se ovdje koriste metode programiranja, nije potrebno dobiti izraz za finalni polinom, već samo evaluirati izraz (7.6).

Funkcija koja vrši Lagrangeovu interpolaciju prima polje točaka *x* i *y* koordinata, broj točaka, te željeni *xUlaz* za koji se računa vrijednost dobivena interpolacijom:

```
double InterpLag(double x[],double y[], int n, double xUlaz)
{
	int i,j,k;
	double suma=0,produkt=1;

	for(j=0;j<n;j++)
	{
		produkt=1;
		for(k=0;k<n;k++)
			if(k!=j)
			{
				produkt=produkt*(xUlaz-x[k])/(x[j]-x[k]);
			}
		suma=suma+y[j]*produkt;
	}
	return suma;
}
```

7.4.2 Po dijelovima linearna interpolacija

Kada se Lagrangeov polinom koristi za interpolaciju velikog broja točaka, rezultat može sadržavati nagle oscilacije koje vrlo vjerojatno nisu sadržane u osnovnoj funkciji koju podaci predstavljaju. Stoga se kod velikog broja točaka umjesto jednog polinoma koristi interpolacija po dijelovima. Najjednostavniji način interpolacije po dijelovima je koristiti pravce interpolirane kroz dvije točke redoslijednom po rastućoj *x* vrijednosti. Primjer linearne interpolacije po dijelovima prikazan je na sljedećoj slici.



Slika 7.8 Interpolacija po dijelovima

Kod ovog načina interpolacije, problem nije pronaći funkciju pravca koji prolazi kroz dvije točke, već koje dvije točke odabrati ovisno o *x* vrijednosti na kojoj se traži rezultat interpolacije. Za svaku ulaznu *x* točku potrebno je dakle prvo pronaći između koje dvije točke se nalazi, a zatim izvršiti evaluaciju pomoću jednadžbe pravca kroz dvije točke *:*

$$y = ax + b = y_d + \frac{y_g - y_d}{x_g - x_d} \cdot \left(x - x_d \right) \tag{7.7}$$

gdje su x_d,y_d koordinate prve (donje po *x* vrijednosti) točke, a x_g,y_g koordinate druge (gornje) točke.

Linearna interpolacija:

```
double InterpLin(double x[], double y[], int n, double xUlaz)
{
	int i, j, k, iD, iG=n-1;
	for (i = 1; i < n; i++)
	{
		if (x[i] >= xUlaz)
		{
			iG = i;
			break;
		}
	}
	iD = iG - 1;
	return y[iD] + (y[iG] - y[iD]) / (x[iG] - x[iD]) * (xUlaz - x[iD]);
}
```

Linearna i Lagraneova interpolacija koriste se na isti način. Prvo se definiraju *x* i *y* koordinate interpolacijskih točaka u zasebnim poljima. Kod svakog poziva funkcije potrebno je poslati koordinate točaka, broj točaka, te željeni *x* za koji se evaluira vrijednost korištenjem odabrane metode interpolacije:

```
int main()
{
	double x,xTocke[] = { 0, 10, 20, 40 }, yTocke[] = { 30, 25, 10, 7 };
	for (x = 0; x <= 40; x = x + 2)
	{
		printf("Lagrange(%lf)=%lf\n", x, InterpLag(xTocke, yTocke, 4, x));
		printf("Linear(%lf)=%lf\n", x, InterpLin(xTocke, yTocke, 4, x));
	}
	return 0;
}
```

Pitanja za provjeru znanja 7.4

- Ako je zadana tablica sunčevog zračenja za svaki sat kroz cijelu godinu; koju metodu interpolacije koristiti za interpolaciju na razini minute?
- Što ako je isto potrebno napraviti za jedan dan?

7.5 Rješavanje linearnih sustava

Za rješavanje linearnih sustava jednadžbi postoji velik broj algoritama koji se koriste ovisno o veličini i vrsti sustava jednadžbi. Linearni sustav s *n* jednadžbi može se u općem obliku zapisati ovako:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ &\ldots \\ &a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{aligned} \tag{7.8}$$

Ovo se skraćeno u matričnom obliku zapisuje kao **Ax**=**b**, gdje je **A** kvadratna matrica veličine *n*, a **x** i **b** su vektor-stupci s *n* elemenata. Ovaj se sustav zapisuje pomoću proširene matrice:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right] \tag{7.9}$$

Općenito kod rada s ovakvim sustavom jednadžbi dopuštene su tri operacije:

1. Zamjena položaja dvaju redaka,
2. Množenje retka s ne-nula skalarnom vrijednosti i
3. Dodavanje jednom retku umnožak skalara i drugog retka.

Bilo koja od ovih operacija ne utječe na rezultat, stoga se ove operacije mogu koristiti za dovođenje sustava u oblik koji je lakše rješavati.

7.5.1 Gaussova eliminacija

Jedan od načina za rješavanje sustava jednadžbi manipulacijom korištenjem prethodnih operacija naziva se metoda Gaussove eliminacije. Kod metode Gaussove eliminacije cilj je prvo prethodni sustav svesti na gornju trokutastu matricu:

$$\left[\begin{array}{cccc|c} a^{\nabla}_{11} & a^{\nabla}_{12} & \cdots & a^{\nabla}_{1n} & b^{\nabla}_{1} \\ 0 & a^{\nabla}_{22} & \cdots & a^{\nabla}_{2n} & b^{\nabla}_{2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a^{\nabla}_{nn} & b^{\nabla}_{n} \end{array}\right] \tag{7.10}$$

Ovo je moguće dobiti korištenjem operatora dodavanja jednom retku umnožak skalara i drugog retka. Trokut $^{\nabla}$ označava da se radi o sustavu svedenom na gornju trokutastu matricu a brojke načelno neće biti jednake onima iz izvorne matrice. Primjer:

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -2 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}\right] \begin{array}{l} R_2+\frac{3}{2}R_1\to R_2 \\ R_3+R_1\to R_3 \end{array} \rightarrow \left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & -0.5 & 0.5 & 1 \\ 0 & 2 & 1 & 5 \end{array}\right] \begin{array}{l} \\ R_3+4R_2\to R_3 \end{array} \rightarrow \left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & -0.5 & 0.5 & 1 \\ 0 & 0 & 3 & 9 \end{array}\right]$$

Bez obzira kojim se od navedenih dopuštenih operacija došlo do gornje trokutaste matrice, rješenje sustava biti će jednako. Nakon dobivanja trokutaste matrice, može se izračunati vrijednost nepoznanice $x_n = b^{\nabla}_{n} / a^{\nabla}_{nn}$. Sada se ova vrijednost može uvrstiti u prethodnu jednadžbu. Jednadžba *n-1,* koja je ranije imala dvije nepoznanice, sada ima samo jednu i može se izračunati vrijednost nepoznanice x_{n-1}.

Sada se obje prethodno izračunate nepoznanice mogu uvrstiti u jednadžbu *n-2* i tako dalje. Ovaj postupak može se zapisati pomoću jednadžbe:

$$x_i = \frac{b^{\nabla}_i - \sum_{j=i+1}^{j<=n} a^{\nabla}_{ij} \cdot x_j}{a^{\nabla}_{ii}} \tag{7.11}$$

S tim da se prvo računa x_n, i ide se unazad prema x_1. Suma u brojniku predstavlja sumu svih nedijagonalnih članova pomnoženih s ranije izračunatim vrijednostima nepoznanica. Primjenom na prethodno dobivenoj trokutastoj matrici dobije se rezultat:

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & -0.5 & 0.5 & 1 \\ 0 & 0 & 3 & 9 \end{array}\right] \xrightarrow{x_3=\frac{9-(0)}{3}=3} \left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & -0.5 & 0.5\cdot 3 & 1 \\ 0 & 0 & 3 & 9 \end{array}\right] \xrightarrow{x_2=\frac{1-(0.5\cdot 3)}{-0.5}=1} \left[\begin{array}{ccc|c} 2 & 1\cdot 1 & -1\cdot 3 & 8 \\ 0 & -0.5 & 0.5\cdot 3 & 1 \\ 0 & 0 & 3 & 9 \end{array}\right]$$

$$\rightarrow x_1 = \frac{8-(-1\cdot 1+1\cdot 3)}{2} = 5$$

Funkcija za rješavanje linearnog sustava pomoću ovog algoritma je prikazana u nastavku. Funkcija prvo izrađuje proširenu matricu *Ab* iz postojećih matrica *A* i *b* koje su ulazni argumenti funkciji. Zatim slijedi eliminacija unaprijed. Kod eliminacije unaprijed kreće se od prvog retka, a zatim se računa potreban *omjer* takav da se može eliminirati prvi ne-nula element u redcima ispod. Rezultat ovoga je gornje trokutasta matrica, a zatim se može krenuti sa supstitucijom unatrag prema izrazu (7.11).

```
#include<stdio.h>
#define SIZE 100
void RjesiLinSustav(int n, double A[SIZE][SIZE], double b[SIZE], double x[SIZE])
{
	int i, j, k;
	double omjer, sumNeDijagClan;
	//Napravi matricu Ab od matrice A i nuliraj vektor x
	for (i = 0; i < n; i++)
	{	A[i][n] = b[i]; x[i] = 0;   }
	for (i = 0; i < (n - 1); i++) //Eliminacija unaprijed
	{
		for (j = (i + 1); j < n; j++)
		{
			omjer = A[j][i] / A[i][i];
			// Eliminacija prvog ne - nula elementa retka dodavanjem
			// j - tom retku skalarni umnožak i - tog retka
			for (k = 0; k <= n; k++)
				A[j][k] = A[j][k] - omjer * A[i][k];
		}
	}
	for(i = n-1; i>=0; i--) // Substitucija unatrag
	{
		sumNeDijagClan = 0;
		for(j=i+1;j<n;j++)
			sumNeDijagClan += A[i][j]*x[j];
		x[i] = (A[i][n] - sumNeDijagClan) / A[i][i];
	}
}
```

Da bi se poopćio rad funkcije, dodana je definicija *SIZE*, te joj je pridijeljen broj 100. Ovo znači da će se svugdje u programu riječ *SIZE* zamijeniti brojem 100 prije prevođenja programa u .exe datoteku. Ovo omogućava laku izmjenu maksimalne veličine linearnog sustava. Kod prethodne funkcije, također je potrebno paziti da funkcija izmijeni matricu *A*, tako da je potrebno napraviti kopiju matrice prije slanja u slučaju ako vam je matrica potrebna u nastavku programa. Za ovaj slučaj to nije potrebno. U glavnoj funkciji sada se definira matrica *A* te vektor *b*. Također je potrebno deklarirati vektor *x* u koji će biti pohranjeni rezultati, te broj jednadžbi *n*:

```
int main()
{
	int n = 3;
	double A[SIZE][SIZE] ={
				{2,1,-1},
				{-3, -2, 2},
				{-2, 1, 2}
				};

	double x[SIZE], b[SIZE] = { 8,-11,-3 };
	RjesiLinSustav(n, A, b, x);
	printf("Rjesenje x: \n%lf\n%lf\n%lf\n",x[0],x[1],x[2]);
	return 0;
}
```

Ispis rezultata:

```
Rjesenje x:
5.000000
1.000000
3.000000
```

7.5.2 Gaussova metoda s pivotiranjem

Prethodni osnovni algoritam ima jednu pogrešku (koju je relativno lako ispraviti). Ako je element na dijagonali jednak nuli, funkcija neće dati rješenje čak ni ako postoji. Ako u prethodnom primjeru matricu zamijenimo s:

```
double A[SIZE][SIZE] ={
			{0,1,-1},
			{-3, -2, 2},
			{-2, 1, 2}
			};
```

Ispis rezultata biti će:

```
Rjesenje x:
-nan(ind)
-nan(ind)
-nan(ind)
```

Ovo se događa zbog dijeljenja s nulom.

Ovaj se problem može riješiti ugrađivanjem pravila o zamjeni položaja dvaju redaka. Ovo se zove *pivotiranje*, a ugrađuje se unutar petlje eliminacije unaprijed. Jedna varijanta *pivotiranja* je pronaći redak za zamjenu tako da dijagonalni element ima najveći mogući iznos (apsolutna vrijednost). Za ovo je potrebno proći po svim elementima koji se nalaze ispod trenutno dijagonalnog elementa i pronaći najveći član. Drugim riječima, kod *i*-tog koraka eliminacije unaprijed, potrebno je u rasponu od *j*=*i* do *n* pronaći

najveći član $|a_{ji}|$. Zatim se vrši zamjena *i*-tog i *j*-tog retka. Ako je najveći član u *i*-tom retku, nema potrebe za zamjenom. Ovo se može programski izvesti tako da se sljedeće doda u petlju eliminacije unaprijed, prije ugniježđene petlje:

```
for (redakMax =0, j=i; j < n; j++)
	if (fabs(A[j][i]) > redakMax)
	{
		redakMax = fabs(A[j][i]);
		redakMaxInd = j;
	}
for (j = 0; j <= n; j++)
{
	temp = A[redakMaxInd][j];
	A[redakMaxInd][j] = A[i][j];
	A[i][j] = temp;
}
```

Ostatak funkcije potpuno je isti kao i kod prethodne. Na istom primjeru gdje je ranije funkcija dala grešku, sada će raditi ispravno:

```
Rjesenje x:
-1.666667
3.222222
-4.777778
```

Postupak *pivotiranja* važan je ne samo zbog izbjegavanja ovih kritičnih grešaka, već i zbog povećanja točnosti rada funkcije. Naime, u slučaju da je dijagonalni element puno manji u odnosu na elemente u stupcu ispod, dijeljenje velikih brojeva s malima uzrokuje veću numeričku popa grešku u odnosu na obrnutu situaciju.

7.6 Zadaci za vježbu

1⁺. Zadana je funkcija brzine automobila:

$$f(t)=\begin{cases}100(0.5t)^2, & \text{ako je } t\le 2\\ 50t, & \text{ako je } 2<t\le 4\\ 200, & \text{ako je } 4<t\le 10\\ 0, & \text{inače}\end{cases}$$

potrebno je:

a) U zadanom intervalu ispiši vrijeme i brzinu.

b) Izračunati pomak i ubrzanje.

c) Izračunati trzaj (engl. jerk), tj. derivaciju ubrzanja.

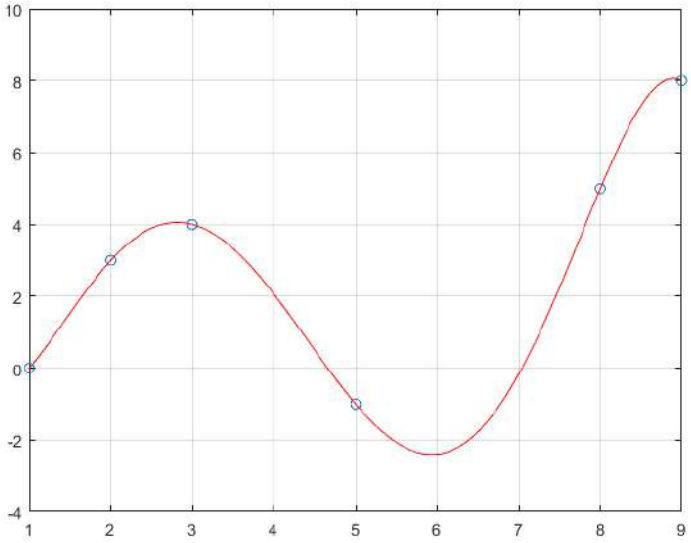
2. Zadana je funkcija pomaka čestice definirana s:

$$f(x)=\begin{cases}0.25x^2, \text{ za } x<4\\ 2+0.5x, \text{ za } 4\le x<8\\ 6, \text{ za } x\ge 8\end{cases}$$

Potrebno je u rasponu od 0 do 10, s razmakom od 1, izračunati brzinu i ubrzanje. Program treba ispisati u četiri stupca, vrijeme, pomak, brzinu i ubrzanje.

3⁺. Potrebno je napraviti program koji će pronaći ekstrem za opće funkcije oblika f(x). Za primjer uzmite f(x)=sin(x). Ekstrem se može pronaći korištenjem Newtonove metode gdje se traži nultočka derivacije funkcije.

4⁺. Za čvorove prema slici, prikazujte površinu ispod Langrangeovog polinoma za ispravno unesene granice integracije.



5. Funkcija *Fun(t)* iz zadatka 6.1 predstavlja brzinu gibanja (km/h) automobila u trajanju od 10 sekundi. Potrebno je napraviti sljedeći program:
Izračunaj ukupno potrošenu energiju [J], odnosno potrebno je riješiti integral:

$$E = \int_0^{80} 0.35 \cdot 1.5 \cdot \frac{1}{2} \cdot 1.225 \cdot Fun(t)^3 dt$$

Ispišite potrošnju goriva: E*0.35/34200000 [litara].

Prethodni izraz poznat je iz mehanike fluida. Koeficijent otpora automobila dobiva se tunelskim ispitivanja ili numeričkom (CFD) analizom. U ovom primjeru odabrana vrijednost je c_D=0.35. Sljedeća brojka u izrazu predstavlja napadnu površinu automobila, odabrana je A=1.5 m^2. Zatim slijedi gustoća zraka ρ=1.225 kg/m^3. Općenito, jednadžba za izračun sile otpora glasi:

$$F_D = 0.5\, \rho\, c_D\, A\, v^2$$

Iz dinamike je poznato da se snaga računa kao umnožak sile i brzine $P=F_D v$, a integracijom snage po vremenu dobije se utrošena energija. Za izračun potrošnje u litrima, ova se snaga množi s pretpostavljenom učinkovitosti motora te dijeli sa specifičnom energijom goriva.

6. Napravite program koji za funkciju y=sin(x*x) traži sve nultočke u rasponu od 1 do 5π.

7. Zadana je funkcija f(x)=sin(x). Napravi program koji daje usporednu tablicu postupka integracije s Trapeznim i sa Simpsonovim pravilom. Korisnik unosi donju i gornju granicu integracije te broj integracijskih intervala, a svaki je širine Δx. Usporedna tablica sadržava ukupni iznos integrala do trenutne x vrijednosti, za obje metode. Rezultate je potrebno ispisati za svaki interval Δx. Primjer kako treba izgledati ispis rezultata:

```
Trapez integral do x        Simpson integral do x
1.5                         1.43
2.6                         2.56
...
```

8. Zadana je funkcija $f(x)=A\cdot x^2+B\cdot x+1$, gdje A i B unosi korisnik. Potrebno je izraditi program koji će ispisati vrijednost funkcije $f(x)$, derivacije $f'(x)$ te integrala $fInt(x) = \int_0^x f(x)dx$. Ispis izradite u tri stupca gdje se x varira u rasponu od 0 do 10, s razmakom Δx=0.1. Primjer ispisa:

```
f(x)        df(x)/dx    fInt(x)
0.0         -1.39       0.0
0.1         -1.35       0.543
…
```

9. Funkcija *f*(*x*) je definirana Lagrangeovim interpolacijskim polinomom kroz točke:

```
x=[0 1 2 3]
y=[y0 5 7 8]
```

Funkcija *g*(*x*) je definirana kao:

$$g(x) = \int_0^x f(x)dx$$

Korisnik unosi proizvoljnu vrijednost *y0*. Napravite program koji rješava jednadžbu *f*(*x*)**g*(*x*)=1

10. Mjerena je ekspanzija klipa u cilindru gdje su podaci dani u obliku matrice. Prvi stupac prikazuje podatke o volumenu u $[cm^3]$, a drugi stupac promjenu tlaka prilikom ekspanzije u [MPa]. Podaci iznose:

$$\begin{bmatrix} 600 & 2.2 \\ 622 & 2.6 \\ 634 & 2.88 \\ 638 & 3.02 \\ 645 & 3.14 \\ 652 & 3.44 \\ 663 & 3.5 \\ 670 & 3.58 \\ 688 & 3.62 \\ 692 & 3.7 \\ 705 & 3.78 \end{bmatrix}$$

Potrebno je odrediti dobiveni rad prilikom ekspanzije klipa. Za dobiti funkciju tlaka o volumenu $p = f(V)$, koristite po dijelovima linearnu interpolaciju. Rad se dobije rješavanjem integrala:

$$W = \int_{V1}^{V2} p\, dV$$

11. Na tijelo mase $m = 1\ [kg]$ djeluje sila u trajanju od 1 $[s]$, čiji su podaci izmjereni u vremenu i prikazani u 2:

$$F = [2.2, 3.6, 8.12, 5.02, 1.2], [N]$$
$$t = [0.05, 0.22, 0.6, 0.72, 0.96], [s]$$

Potrebno je Lagrangeovom interpolacijom uspostaviti vezu između sile F i vremena t te odrediti brzinu tijela nakon djelovanja sile. Pomoć: Zakon očuvanja količine gibanja

$$\int_{t_0}^{t} F(t)dt = mv$$

11. Zadana je funkcija s dvije varijable:

$$f(x, y) = 2x^2y - y^2 + 3x^3$$

Potrebno je numerički odrediti gradijent funkcije u točki $T(4,2)$. Gradijent funkcije:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

12. Za dani skup točaka potrebno je uraditi Lagrangeov i po dijelovima linearni interpolacijski polinom.

```
x=[ 1,3,4,5,7,10]
y=[1,4,10,6,2,7]
```

Potrebno je odrediti apsolutnu razliku Δ između površina Lagrangeovog inerpolacijskog polinoma i po dijelovima linearne interpolacije.

7.7 Riješeni primjeri

Primjeri označeni znakom $^+$ sada su riješeni.

ZADATAK 1 Izračun prijeđenog puta i ubrzanja iz funkcije brzine (Laboratorijska vježba 8)

Zadana je funkcija brzine automobila:

$$f(t)=\begin{cases}100(0.5t)^2, & \text{ako je } t\le 2\\ 50t, & \text{ako je } 2<t\le 4\\ 200, & \text{ako je } 4<t\le 10\\ 0, & \text{inače}\end{cases}$$

potrebno je:

a) U zadanom intervalu ispiši vrijeme i brzinu.

b) Pomak i ubrzanje.

c) Trzaj (engl. jerk), tj derivacija ubrzanja.

Rješenje je prikazano ispod.

Glavna funkcija:

```
int main()
{
	double vrijeme[100], brzina[100], ubrzanje[100], pomak[100], jerk[100], t;
	int i, brojTocaka;
	brojTocaka = 0;
	for (t = 0.0; t <= 11.0; t = t + 0.25)
	{
		vrijeme[brojTocaka] = t;
		brojTocaka++;
	}
	for (i = 0; i<brojTocaka; i++)
	{
		brzina[i] = Fun(vrijeme[i]);
		ubrzanje[i] = DerivacijaFun(vrijeme[i]); //Pazite jedinice (m,km,s,h,...)
		pomak[i] = OdredeniIntegralFun(0.0, vrijeme[i]);
		jerk[i] = DrugaDerivacijaFun(vrijeme[i]);
	}
	//Ispis
	printf("t[s]\t\ts[m]\t\tv[km/h]\t\t[m/s^2]\t\t[m/s^3]\n");
	for (i = 0; i<brojTocaka; i++)
	{
		printf("%lf\t%lf\t%lf\t%lf\t%lf\n", vrijeme[i], pomak[i], brzina[i],
ubrzanje[i], jerk[i]);
	}
	return 0;
}
```

Funkcija:

```
double Fun(double t)
{
	double brzina;

	if (t<2)
		brzina = 100 * (0.5*t)*(0.5*t);
	else if (t <= 4)
		brzina = 50 * t;
	else if (t <= 10)
		brzina = 200;
	else
		brzina = 0;

	return brzina;
}
```

Funkcija za izračun derivacije i integrala koristi od ranije gotove funkcije *DerivacijaFun* i *OdredeniIntegralFun*. Funkcija za izračun druge derivacije:

```
double DrugaDerivacijaFun(double t)
{
	double derivacija, deltaX;
	deltaX = 0.001;
	derivacija = (DerivacijaFun(t + deltaX) - DerivacijaFun(t)) / deltaX;
	return derivacija;
}
```

ZADATAK 3 Program za pronalaženje ekstrema funkcija (Laboratorijska vježba 9).
Potrebno je napraviti program koji će pronaći ekstrem za opće funkcije oblika *f(x)*. Za primjer uzmite *f(x)=sin(x)*.

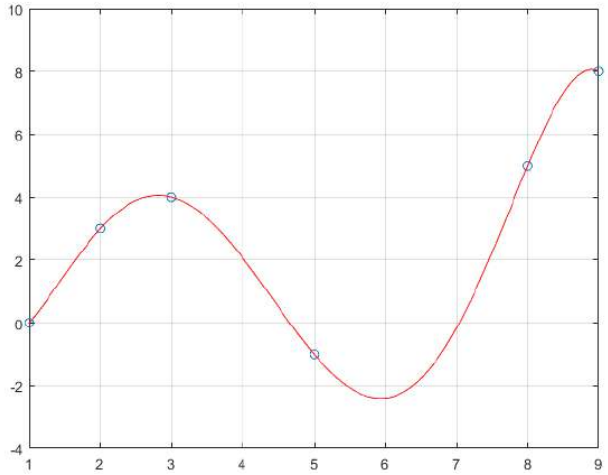
Rješenje Ekstrem se može pronaći korištenjem Newtonove metode gdje se traži nultočka derivacije funkcije. Uz to je potrebno koristiti i drugu derivaciju (derivacija derivacije). Ranije su prikazane funkcije za deriviranje, a funkcija za traženje ekstrema može se napraviti na sljedeći način:

```
double EkstremFun(double x)
{
	double tol = 0.00001;
	do
	{
		x = x - DerivacijaFun(x) / DrugaDerivacijaFun(x);
	} while (fabs(DerivacijaFun(x))>tol);
	return x;
}
```

Glavna funkcija sada za traženje ekstrema poziva već gotovu funkcije i ispisuje rezultat:

```
int main()
{
	printf("Ekstrem funkcije je : %lf\n",EkstremFun(1));
	return 0;
}
```

ZADATAK 4 Za čvorove prema slici, prikažite površinu ispod Langrange-ovog polinoma za ispravne parametre (granice).



Rješenje Prvo je izrađena funkcija *Fun* koja prima realni broj *x*, te vraća *y* koji se računa Lagrangeovom interpolacijom kroz točke *xI* i *yI* prema zadanoj slici.

```
double Fun(double x)
{
	int nI = 6;
	double xI[] = { 1, 2, 3,  5, 8, 9 };
	double yI[] = { 0, 3, 4, -1, 5, 8 };
	return InterpLag(xI, yI, nI, x);
}
```

Za evaluaciju Lagrangeovog polinoma (*InterpLag*), i za izračun integrala (*OdredeniIntegralFun*) koriste se gotove funkcije. U glavnom programu sad je preostalo

```
int main()
{
	double a, b, integral;
	while (1)
	{
		printf("Unesi donju granicu integracije (izmedju cvorova): ");
		scanf_s("%lf", &a);
		printf("Unesi gornju granicu integracije (izmedju cvorova): ");
		scanf_s("%lf", &b);
		if (a > b || a < 1 || b > 9)
		{
			printf("\nKRIVI unos!!!\n");
			break;
		}
		integral = OdredeniIntegralFun(a, b);
		printf("Rezultat integracije lang. polinoma: %lf\n\n", integral);
	}
}
```

Primjer rada programa:

```
Unesi donju granicu integracije (izmedju cvorova): 2
Unesi gornju granicu integracije (izmedju cvorova): 4
Rezultat integracije lang. polinoma: 6.993265

Unesi donju granicu integracije (izmedju cvorova): 2
Unesi gornju granicu integracije (izmedju cvorova): 3
Rezultat integracije lang. polinoma: 3.765780

Unesi donju granicu integracije (izmedju cvorova): 0
Unesi gornju granicu integracije (izmedju cvorova): 0

KRIVI unos!!!
```

7.8 Inženjerski primjeri

PRIMJER 1 Napravi program za izračun koeficijenta otpora fluida u ravnim cijevima, rješavanjem Darcy–Weisbach jednadžbe [16]. Za primjer odaberite Re=10^5 te relativnu hrapavost 0.001.

Rješenje Ovdje je potrebno riješiti nelinearnu Darcy–Weisbach jednadžbu za izračun koeficijenta otpora trenja fluida za ravnu cijev (f) za određeni Reynolds broj (Re) i hrapavost ($relHrap$):

$$\frac{1}{\sqrt{f}} = -2\log\left(\frac{relHrap}{3.51} + \frac{2.51}{\mathrm{Re}\sqrt{f}}\right)$$

Za uneseni Reynolds broj Re i relativnu hrapavost, program treba pronaći koeficijent f. Za primjer je odabran Re=10^5 te $relHrap=0.001$. Time se dobije nelinearna jednadžba (f je nepoznanica x):

$$\frac{1}{\sqrt{x}} = -2\log\left(\frac{0.0001}{3.51} + \frac{2.51}{10^5\sqrt{x}}\right)$$

Pretvaranjem u problem traženja nultočke, potrebno je riješiti:

$$fun = \frac{1}{\sqrt{x}} + 2\log\left(\frac{0.0001}{3.51} + \frac{2.51}{10^5\sqrt{x}}\right) = 0$$

(Razmislite kako biste pokušali analitički „izvući“ x iz jednadžbe?) Za numeričko rješavanje, prethodna funkcija se definira u zasebnoj korisnički definiranoj funkciji:

```
double Fun(double x),
{
	double relHrap = 0.001, Re = 100;
	return 1.0 / sqrt(x) + 2 * log10(relHrap / 3.51 + 2.51 / (Re*sqrt(x)));
}
```

Za izračun nultočke metodom polovljenja te Newtonovom metodom, koriste se ranije izrađene funkcije. Glavna funkcija sada za rješavanje nelinearne jednadžbe može samo pozvati već gotove funkcije i ispisati rezultat:

```
int main()
{
	printf("Nultocka funkcije je (Polovlje): %lf\n", MetodaPolovljenjaFun(0.00001, 1));
	printf("Nultocka prema Newtonu: %lf\n", NewtonovaNultocka(1));
	return 0;
}
```

PRIMJER 2 Rješavanje složenih jednadžba stanja plinova.

U pojedinim termodinamičkim proračunima može se koristiti jednadžba za idealni plin [9] $P\bar{v} = R_u T$, gdje je P tlak, $\bar{v}$ je molarni volumen (m^3/mol), R_u plinska konstanta (R_u=8.3145 J/molK) i T temperatura. Ako su poznate dvije veličine, treću je jednostavno izračunati (npr. ako su poznata temperatura i tlak, specifični volumen jednak je RT/P). U kompliciranijim slučajevima, jednadžba idealnog plina više ne daje dobre rezultate, a tada se mogu koristiti složenije jednadžbe, kao što je Beattie-Bridgeman [17] jednadžba:

$$P = \frac{R_u T}{\bar{v}^2}\left(1 - \frac{c}{\bar{v}T^3}\right)(\bar{v} + B) - \frac{A}{\bar{v}^2}$$

$$gdje\ su: A = A_0\left(1 - \frac{a}{\bar{v}}\right)\ i\ B = B_0\left(1 - \frac{b}{\bar{v}}\right)$$

Konstante A_0, a, B_0, b i c uzimaju se ovisno o plinu, a za zrak iznose: A_0= 131.8441 a=0.01931, B_0= 0.04611, b=-0.001101 i c=0.000434. Potrebno je napraviti program koji za dvije unesene veličine daje kao rezultat treću (npr. korisnik unosi tlak i temperaturu, a program računa molarni volumen $\bar{v}$).

Rješenje Problem će se rješavati metodama za rješavanje nelinearnih jednadžbi. Prvo će se napraviti funkcije u obliku $f(x)$=0:

$$\frac{R_u T}{\bar{v}^2}\left(1 - \frac{c}{\bar{v}T^3}\right)(\bar{v} + B) - \frac{A}{\bar{v}^2} - P = 0$$

Što će biti x, ovisi o slučaju, ali u svakom slučaju može se prvo napraviti korisnički definirana funkcija:

```
double BBJednadzba(double P,double T,double vm)
{
    double A,B,A0, a, B0, b, c, Ru=8.3145;
    A0 = 131.8441; a = 0.01931; B0 = 0.04611; b = -0.001101; c = 0.000434;
    A = A0 * (1 - a / vm); B = B0 * (1 - b / vm);

    return Ru * T / (vm * vm) * (1 - c / (vm * T * T * T)) * (vm + B) - A / (vm * vm) - P;
}
```

Cilj je za korisnički uneseni tlak i temperaturu, specifični volumen (vn) pronaći Newtonovom metodom. Tada dva od tri ulazna argumenta trebaju biti konstantna, a za Newtonovu metodu se može izraditi zasebna funkcija. Ova se funkcija ulazne vrijednosti tlaka i temperature držati konstantnima pri traženju nultočke:

```
double PronadiSpecVol(double P, double T, double x)
{
	double der, y, dx = 0.000001, tol = 0.0001;
	do
	{
		der = (BBJednadzba(P, T, x + dx) - BBJednadzba(P, T, x)) / dx;
		y = BBJednadzba(P, T, x);
		x = x - y / der;
	} while (fabs(y) > tol);
	return x;
}
```

Glavna funkcija sada se svodi samo na pozivanje prethodne funkcije, gdje je pretpostavljeno početno rješenje za specifični volumen $\bar{v}$=0.025 m^3/mol.

```
int main()
{
	double P, T, vm;
	printf("Unesite tlak (Pa): ");
	scanf_s("%lf", &P);
	printf("Unesite temperaturu (K): ");
	scanf_s("%lf", &T);;;
	vm = PronadiSpecVol(P, T, 0.025);
	printf("Molarni volumen iznosi %lf (m3/mol)\n", vm);

	return 0;
}
```

Korisnik unosi podatke, a program ispisuje :

```
Unesite tlak (Pa): 101000
Unesite temperaturu (K): 400
Molarni volumen iznosi 0.048658 (m3/mol)
```

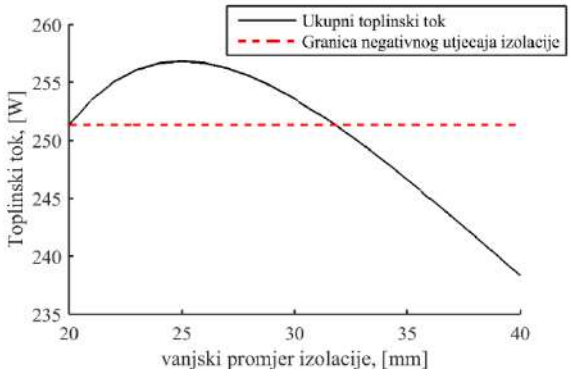
PRIMJER 3 Ovisnost toplinskog otpora o debljini izolacije cijevi i kritična debljina izolacije.
Zadatak je pronaći kritičnu debljinu izolacije cijevi. Promjer cijevi jednak je D_c=40 mm, temperatura cijevi je 400 K, a temperatura okoliša iznosi 300 K. Duljina cijevi iznosi L=10m. Pretpostavite da je izolacija od mineralne vune s toplinskom vodljivošću λ=0.05 W/mK. Za koeficijent konvekcije uzmite h=2 W/m^{2}K.

Rješenje Toplinski otpor sloja izolacije te otpor konvekcije prema okolišu mogu se računati prema izrazima [9]:

$$R_{izolacija} = \frac{\ln(r_v / r_u)}{2\pi L\lambda}$$

$$R_{konvekcija} = \frac{1}{2\pi r_v Lh}$$

gdje je r_v vanjski polumjer sloja izolacije, r_u unutarnji polumjer sloja izolacije, a λ je toplinska vodljivost izolacije. Unutarnji polumjer izolacije jednak je vanjskom polumjeru cijevi, r_u=20 mm. Očekivana ovisnost debljine izolacije o toplinskom toku prikazana je na donjoj slici. Za pronaći kritičnu debljinu, potrebno je pronaći točku kada toplinski postaje ponovno jednak početnom. Tek nakon te debljine izolacija ima pozitivan utjecaj.



Prvo je napravljena funkcija koja prima vanjski i unutarnji polumjer cijevi, a kao rezultat vraća toplinski tok izračunat po prethodnim izrazima.

```
//ru, rv - vanjski i unutarnji polumjer cijevi
double ToplinskiTok(double ru, double rv)
{
	double lamda = 0.05;// Toplinska vodljivost izolacije[W / K]
	double h = 2;// Koeficjent konvekcije[W / m2K]
	double L = 10;// Izložena duljina cijevi[m]
	double Tc = 400, Tokol = 300;// Temperatura cijevi i okoliša[K]
	double Q, Rizolacija, Rkonvekcija;

	// Toplinski i otpori i toplinski gubici
	Rizolacija = log(rv / ru) / (2 * 3.14 * L * lamda);
	Rkonvekcija = 1 / (2 * 3.14 * rv * L * h);
	Q = (Tc - Tokol) / (Rizolacija + Rkonvekcija);

	return Q;
}
```

Zatim je napravljena funkcija *Fun*, kojoj je rezultat razlika između toplinskog toka bez i s izolacijom. Ulaz u funkciju je debljina izolacije *x*.

```
double Fun(double x) // x - debljina izolacije
{
	double ru = 0.02;// Unutarnji promjer izolacije[m]
	return ToplinskiTok(ru,ru+x) - ToplinskiTok(ru,ru);
}
```

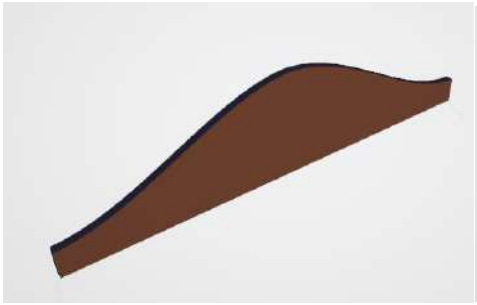
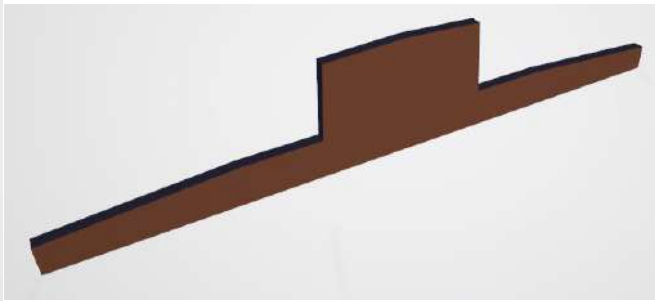
Glavna funkcija poziva već od ranije poznatu funkciju za traženje nultočke *MetodaPolovljenjaFun*.

```
int main()
{
	double debljinaIzol;
	debljinaIzol = MetodaPolovljenjaFun(0.0001, 0.1);
	printf("Kriticna debljina izolacije je %lf mm\n", debljinaIzol * 1000);
	return 0;
}
```

Rezultat programa:

```
Kriticna debljina izolacije je 11.815225 mm
```

PRIMJER 4 Za simetrične limove „Lim2“ i „Lim2“, izmjerene su koordinate točaka s jedne strane (desne strane) osi simetrije. Lim1 ima glatku konturu, dok se kontura za Lim2 sastoji od ravnih segmenata.

Potrebno je napraviti funkciju "Lim1" koja Lagrangeovom interpolacijom opisuje točke:

x[]={0.0, 20.0, 40, 80};

y[]={40, 35, 25, 10};

Napravite funkciju "Lim2" koja linearnom interpolacijom po segmentima opisuje točke:

x[]={0, 20, 21, 40, 80};

y[]={30, 29, 10, 10, 7};

Potrebno je:

a) Tablično ispišite vrijednosti funkcija "Lim1" i "Lim2" u rasponu od x=0 do x=80 s razmakom deltaX=5.

b) Izračunajte površine limova 1 i 2 te ispišite rezultate. Površina se računa pomoću integrala:

$$P1 = 2 \cdot \int_0^{80} Lim1(x)dx \text{ i } P2 = 2 \cdot \int_0^{80} Lim2(x)dx$$

c) Izračunajte statičke momente tromosti:

$$WL1 = 2 \cdot \int_0^{80} \frac{1}{2} Lim1(x)^2 dx \text{ i } WL2 = 2 \cdot \int_0^{80} \frac{1}{2} Lim2(x)^2 dx$$

d) Izračunajte težište te ispišite rezultate. (Težište limova $T1x$=0 i $T2x$=0 jer su simetrični):

$$T1y=WL1/P1 \text{ i } T2y=WL2/P2$$

Rješenje Prvo je potrebno izraditi funkcije koje opisuju oblik. S obzirom na simetriju, funkcije se izrađuju samo za x>0. Za funkciju Lim1 potrebno je koristiti Lagrangeovu interpolaciju (jer je glatka geometrija).

```
double Lim1(double xIn) //Funkciju računamo Lagrangeovim polinomom
{
	double x[] = { 0.0, 20.0, 40, 80 };
	double y[] = { 40, 35, 25, 10 };
	int n = 4;
	return InterpLag(x,y,n, xIn);
}
```

Za funkciju Lim2 potrebno je koristiti linearnu interpolaciju.

```
double Lim2(double xIn) //Koristimo linearnu interpolaciju po segmentima
{
	double x[] = { 0, 20, 21, 40, 80 };
	double y[] = { 30, 29, 10, 10, 7 };
	int n = 4;
	return InterpLin(x,y,n, xIn);
}
```

S obzirom na to da se integracija radi odvojeno nekoliko puta, praktično je napraviti jedinstvenu funkciju za integriranje. Takva funkcija prima polje točaka te broj točaka, a kao rezultat vraća evaluirani integral. Ova funkcija prikazana je u poglavlju 7.2.1, *OdredeniIntegralPolja(poljeX,poljeY,brojTocaka)*. Tu funkciju potrebno je kopirati u programski kôd. Konačno, u nastavku je prikazana glavna funkcija koja koristi sve prethodno navedene funkcije:

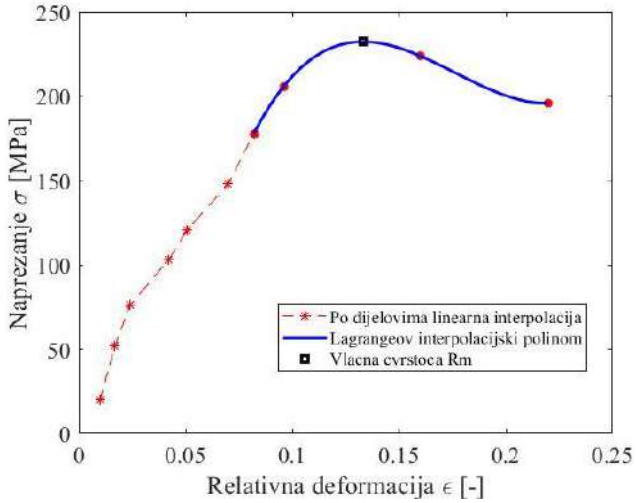
```
int main()
{
	double xIn, povL1, povL2, wL1, wL2, poljeX[1000];
	double poljeYL1[1000], poljeYL2[1000], poljeW1[1000], poljeW2[1000];
	int i, brojTocaka;
	printf("x[mm]\t\tyLim1[mm]\tyLim2[mm]\n");
	for (xIn = 0.0; xIn <= 80.0; xIn = xIn + 5.0)
		printf("%lf\t%lf\t%lf\n", xIn, Lim1(xIn), Lim2(xIn));
	i = 0;
	for (xIn = 0.0; xIn <= 80.0; xIn = xIn + 1)
	{
		poljeX[i] = xIn;
		poljeYL1[i] = Lim1(xIn);
		poljeYL2[i] = Lim2(xIn);
		poljeW1[i] = 0.5*Lim1(xIn)*Lim1(xIn);
		poljeW2[i] = 0.5*Lim2(xIn)*Lim2(xIn);
		i = i + 1;
	}
	brojTocaka = i;
	povL1 = 2 * OdredeniIntegralPolja(poljeX, poljeYL1, brojTocaka);
	povL2 = 2 * OdredeniIntegralPolja(poljeX, poljeYL2, brojTocaka);
	wL1 = 2 * OdredeniIntegralPolja(poljeX, poljeW1, brojTocaka);
	wL2 = 2 * OdredeniIntegralPolja(poljeX, poljeW2, brojTocaka);
	printf("Povrsina lima 1 iznosi: %lf [mm2]\n", povL1);
	printf("Povrsina lima 2 iznosi: %lf [mm2]\n", povL2);
	printf("Teziste lima 1 iznosi: %lf [mm]\n", wL1 / povL1);
	printf("Teziste lima 2 iznosi: %lf [mm]\n", wL2 / povL2);
	return 0;
}
```

PRIMJER 5 Interpolacija mjernih podataka s kidalice i određivanje mehaničkih svojstva materijala. Na kidalici su urađena mjerenja nekog materijala te su dobiveni rezultati vlačnog naprezanja σ i relativne deformacije ϵ. Rezultati mjerenja prikazani su u formatu matrice gdje je prvi redak iznos naprezanja, a drugi redak iznos relativnih deformacija kao:

$$\begin{bmatrix} 0.01 & 0.017 & 0.024 & 0.042 & 0.051 & 0.07 & 0.082 & 0.096 & 0.16 & 0.22 \\ & 20 & 52 & 76 & 103 & 121 & 148 & 178 & 206 & 224 & 196 \end{bmatrix}$$

Može se usvojiti linearna veza od prve do sedme mjerne točke, te polinomska veza od sedme do zadnje točke. Potrebno je odrediti:

a) srednju vrijednost modula elastičnosti E
b) vlačnu čvrstoću R_m materijala (R_m predstavlja najveći iznos naprezanja koje materijal pretrpi tokom ispitivanja)



Rješenje

a) Modul elastičnosti predstavlja nagib linearnog dijela dijagrama vlačne čvrstoće. S obzirom na to da se radi o linearnoj interpolaciji po dijelovima umjesto jednog kontinuiranog pravca, potrebno je za sve točke u linearnom dijelu odrediti nagibe između dviju susjednih točaka te ih usrednjiti.

```
double modulElasticnost(double epsilon[], double sigma[], int n) {

	int i, brojusrednjavanja = n - 1;
	double sumaModula = 0, modulSrednji;

	for (i = 0; i < brojusrednjavanja; i++)
	{

		sumaModula = sumaModula + (sigma[i + 1] - sigma[i]) / (epsilon[i + 1] -
epsilon[i]);

	}
	modulSrednji = sumaModula / brojusrednjavanja;

	return modulSrednji;
}
```

U funkciju koja određuje srednji modul elastičnosti E_{sr} šalju se vektori mjernih podataka s prvih 7 točaka koje definiraju to područje.

b) Da bi se odredila vlačna čvrstoća R_m, potrebno je odrediti ekstrem u interpolaciji Lagrangeova polinoma u zadnje četiri točke. Postupak traženja ekstrema je već prethodno objašnjen i koristi već definirane funkcije *DerivacijaFun, DrugaDerivacijaFun* i *EkstremFun* uz nadopunu argumenata funkcija jer je ovdje Lagrangeov interpolacijski polinom funkcija čiji se ekstrem traži.

```
double DerivacijaFun(double x, double X[], double Y[], int n)
{
	double derivacija, deltaX;
	deltaX = 0.001;
	derivacija = (InterpLag(X, Y, n, x + deltaX) - InterpLag(X, Y, n, x)) / deltaX;
	return derivacija;
}
double DrugaDerivacijaFun(double x, double X[], double Y[], int n)
{
	double derivacija, deltaX;
	deltaX = 0.001;
	derivacija = (DerivacijaFun(x + deltaX, X, Y, n ) - DerivacijaFun(x, X, Y, n)) / deltaX;
	return derivacija;
}
double EkstremFun(double x, double X[], double Y[], int n)
{
	double tol = 0.00001;
	do
	{
		x = x - DerivacijaFun(x, X, Y, n) / DrugaDerivacijaFun(x, X, Y, n);
	} while (fabs(DerivacijaFun(x, X, Y, n)) > tol);
	return x;
}
```

U glavnoj funkciji prikazani su unos mjernih podataka i rastav mjernih podataka na linearni i polinomski dio.

```
int main() {
	// Unos mjernih podataka
	double epsilon[] = { 0.01, 0.017, 0.024, 0.042, 0.051, 0.07, 0.082, 0.096, 0.16, 0.22 };
	double sigma[] = { 20, 52, 76, 103, 121, 148, 178, 206, 224, 196 };
	int nTocaka = sizeof(epsilon) / sizeof(double); //Odredivanje broja mjernih tocaka
	int nLin = 7, nPol=4; //Broj tocaka u linearnom i polinomskom dijelu
	double epsLin[10], sigmaLin[10], epsLag[10], sigmaLag[10];
	int i, j=0;
	//Tocke u linearnom dijelu
	for (i = 0; i < nLin; i++)
	{
		epsLin[i] = epsilon[i];
		sigmaLin[i] = sigma[i];
	}
	//Tocke u polinomskom dijelu
	for (i = nLin-1; i < nTocaka; i++)
	{
		epsLag[j] = epsilon[i];
		sigmaLag[j] = sigma[i];
		j++;
	}
	double x0 = 0.1; //Pocetno rjesenje za odrediti ekstrem
	double Rm = InterpLag(epsLag, sigmaLag, nPol, EkstremFun(x0, epsLag, sigmaLag, nPol));
	printf("Srednji modul jest %.5lf GPa\n", modulElasticnost(epsLin, sigmaLin, nLin)/1e3);
	printf("Vlacna cvrstoca Rm iznosi %.5lf MPa\n", Rm);
	return 0;
}
```

PRIMJER 6 Radna točka hidrauličke pumpe

Dani su mjerni podaci o visini dobave pumpe $H(\dot{V})$ određenog proizvođača. Podaci su iskazani u dva vektora gdje prvi vektor sadrži iznose protoka $\dot{V}$ $[lit/s]$ pri kojima je mjerenje izvršeno a drugi vektor sadrži odgovarajuće iznose visine dobave H $[m]$.

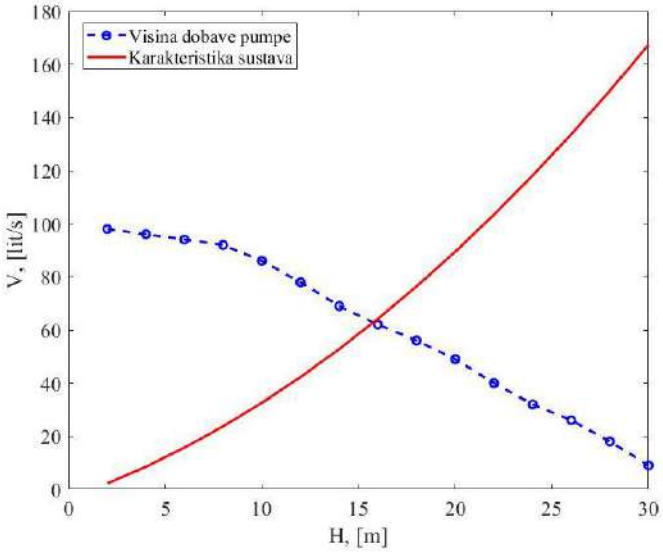
$$H = [98 \quad 96 \quad 94 \quad 92 \quad 86 \quad 78 \quad 69 \quad 62 \quad 56 \quad 49 \quad 40 \quad 32 \quad 26 \quad 18 \quad 9]$$

$$\dot{V} = [2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \quad 14 \quad 16 \quad 18 \quad 20 \quad 22 \quad 24 \quad 26 \quad 28 \quad 30]$$

Navedenu pumpu potrebno je postaviti u neki sustav čije su radne karakteristike definirane jednadžbom

$$H_{sustav}(\dot{V}) = c_1\dot{V}^2 + c_2\dot{V} + c_3$$

Eksperimentalnim mjerenjem utvrđeno je da sljedeći koeficijenti imaju vrijednosti $c_1 = 0.105, c_2 = 2.54$ i $c_3 = -3.32$. Po dijelovima linearna interpolacija može se usvojiti za vezu između visine dobave pumpe i protoka. Potrebno je pronaći radnu točku pumpe, tj. pri kojem protoku treba raditi



Rješenje Radna se točka odredi tako da se nađe zajednički protok iz visine dobave pumpe $H(\dot{V})$ i jednadžbe koja opisuje karakteristike sustava tj. iz jednakosti $H(V) = \dot{H}_{sustav}$. Sustav je definiran zasebnom funkcijom koja prima protok $\dot{V}$ a vraća karakterističnu vrijednost sustava.

```
double Sustav(double V) {
	double Hsustav;
	double c1 = 0.105, c2 = 2.54, c3 = -3.32;
	Hsustav = c1 * V * V + c2 * V + c3;
	return Hsustav;
}
```

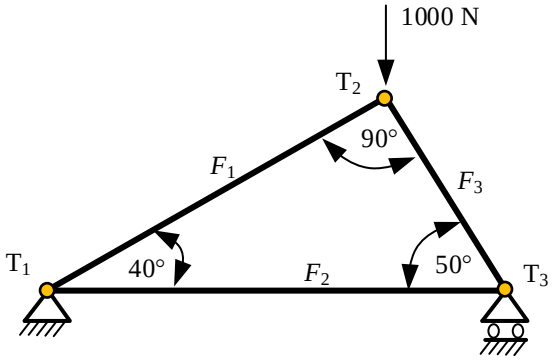
Visina dobave pumpe dobije se po dijelovima linearnom interpolacijom gdje se može iskoristiti već definirana funkcija *InterpLin(double V[], double H[], int nTocaka, double VUlaz)* s odgovarajućim ulaznim argumentima. Problem se riješi tako da se definira funkcija navedene jednakosti $f(\dot{V}) = H - H_{sustav}$ za koju je potrebno pronaći nul-točku $f(\dot{V}) = 0$. Nultočka je pronađena metodom polovljenja intervala, pri čemu je funkcija *MetodaPolovljenjaFun()* prilagođena danom problemu.

```
double FunRadneTocke(double visinaH[], double protokV[], int n, double V) {

	return InterpLin(protokV, visinaH, n, V) - Sustav(V);
}
double MetodaPolovljenjaFun(double visinaH[], double protokV[], int n, double a, double b)
{
	double s = (a + b) / 2, fa, fb, fs, tol = 0.00001;
	fa = FunRadneTocke(visinaH, protokV, n, a);
	fb = FunRadneTocke(visinaH, protokV, n, b);
	if (fa * fb > 0)
	{
		printf("Nema nultocke !\n");
		return s;
	}
	do
	{
		fa = FunRadneTocke(visinaH, protokV, n, a);
		fb = FunRadneTocke(visinaH, protokV, n, b);
		s = (a + b) / 2; fs = FunRadneTocke(visinaH, protokV, n, s);

		if (fa * fs <= 0.0)
			b = s;
		else
			a = s;
	} while (fabs(fs) > tol);

	return s;
}
```

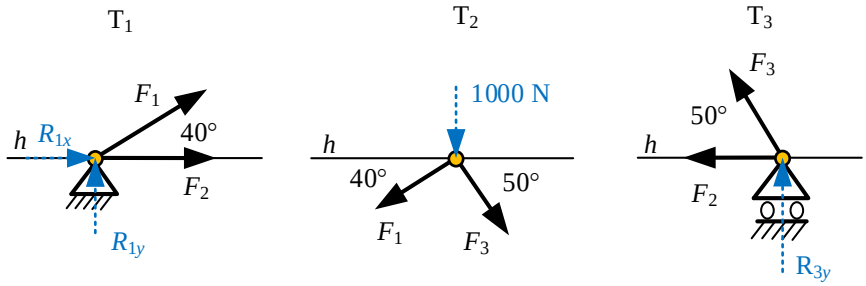
Glavna funkcija:

```
int main(){

	double visinaDobH[] = { 98, 96, 94,  92, 86, 78, 69, 62, 56, 49, 40,  32, 26, 18, 9 };
	double protokV[] = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30 };
	int nTocaka = sizeof(visinaDobH) / sizeof(double);
	double c1 = 0.105, c2 = 2.54, c3 = -3.32;

	double radniProtok = MetodaPolovljenjaFun(visinaDobH, protokV, nTocaka, protokV[0],
protokV[nTocaka - 1]);
	double radnaDobava = InterpLin(visinaDobH, protokV, nTocaka, radniProtok);

	printf("Radna tocka sustava\nProtok=%.5lf [lit/s]\nVisina dobave=%.5lf [m]",
radniProtok, radnaDobava);

	return 0;
}
```

PRIMJER 7 Zadana je štapna konstrukcija prema donjoj slici. Potrebno je izračunati sile u štapovima (F_1, F_2 i F_3).



Rješenje Jednadžbe ravnoteže za horizontalne i vertikalne sile mogu se postaviti u točkama T_1, T_2 i T_3 [5], kao što je prikazano na slici ispod. Osim sila u štapovima, nepoznanice su i reakcije u osloncima (R_{1x}, R_{1y} i R_{3y}). Za svaku se točku može postaviti po jedna jednadžba za horizontalni i vertikalni smjer, što sve skupa daje 6 jednadžbi. Nepoznanice su tri sile u štapovima i tri reakcije, što znači da sustav ima i 6 nepoznanica.



Postavljene su jednadžbe ravnoteže za sve tri točke:

$$R_{1x} + F_2 + \cos(40°) \cdot F_1 = 0$$
$$R_{1y} + \sin(40°) \cdot F_1 = 0$$
$$-\cos(40°) F_1 + \cos(50°) \cdot F_3 = 0$$
$$-\sin(40°) F_1 - \sin(50°) \cdot F_3 - 1000 = 0$$
$$-F_2 - \cos(50°) \cdot F_3 = 0$$
$$R_{3y} + \sin(50°) \cdot F_3 = 0$$

Ovaj sustav jednadžbi može se matrično zapisati u obliku ***Ax*=*b***:

$$\begin{bmatrix} \cos(40°) & 1 & 0 & 1 & 0 & 0 \\ \sin(40°) & 0 & 0 & 0 & 1 & 0 \\ -\cos(40°) & 0 & \cos(50°) & 0 & 0 & 0 \\ -\sin(40°) & 0 & -\sin(50°) & 0 & 0 & 0 \\ 0 & -1 & -\cos(50°) & 0 & 0 & 0 \\ 0 & 0 & \sin(50°) & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ R_{1x} \\ R_{1y} \\ R_{3y} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1000 \\ 0 \\ 0 \end{bmatrix}$$

Prethodne matrice ***A*** i ***b*** se zatim unesu u C, te rješavaju prethodno razvijenom funkcijom za rješavanje linearnih sustava metodom Gaussove eliminacije s *pivotiranjem*.

```
int main()
{
	int n = 6;
	double A[SIZE][SIZE] = {
		{cos(40/57.3),	1,	0,		1,	0,	0},
		{sin(40/57.3),	0,	0,		0,	1,	0},
		{-cos(40/57.3),	0,	cos(50/57.3),	0,	0,	0},
		{-sin(40/57.3),	0,	-sin(50/57.3),	0,	0,	0},
		{0,		-1,	-cos(50/57.3),	0,	0,	0},
		{0,		0,	sin(50/57.3),	0,	0,	1}
		};
	double x[SIZE], b[SIZE] = { 0, 0, 0, 1000, 0, 0 };
	RjesiLinSustav(n, A, b, x);
	printf("Sile iznose F1: %.1f N, F2: %.1f N, F3: %.1f N\n", x[0], x[1], x[2]);
	printf("Reakcije: R1x: %.1f N, R1y: %.1f N, R3y: %.1fN\n", x[3], x[4], x[5]);
	return 0;
}
```

Pokretanjem programa dobije se uvid u rezultate prikazane u nastavku:

```
Sile iznose F1: -642.8 N, F2: 492.5 N, F3: -766.1 N
Reakcije: R1x: 0.0 N, R1y: 413.2 N, R3y: 586.8N
```

Za provjeru točnosti može se koristiti reakcija u smjeru *x* u točci T_1 koja treba biti jednaka nuli zato što nema vanjskih sila koje djeluju u horizontalnom smjeru koje bi mogle uzrokovati tu reakciju. Osim toga zbroj dviju reakcija u smjeru *y* treba iznositi ukupno 1000 N.

LITERATURA

[1] G. Magazinović, *Primjena elektroničkih računala: podloge za laboratorijske vježbe - programski jezik C*. Split: Fakultet elektrotehnike, strojarstva i brodogradnje, 2003.

[2] D. Vučina, *Primjena računala u inženjerskoj analizi : s programskim primjerima u jezicima C i MATLAB*. Split: Fakultet elektrotehnike, strojarstva i brodogradnje, 2007.

[3] D. Vučina, *Metode inženjerske numeričke optimizacije: s primjerima primjene u programskom jeziku C i MATLAB*. Split: Fakultet elektrotehnike, strojarstva i brodogradnje, 2005.

[4] Ž. Lozina, *Uvod u programiranje*. Split: Fakultet elektrotehnike strojarstva i brodogradnje, 2006.

[5] R. Pavazza, *Tehnička mehanika - Statika*. Split: Fakultet elektrotehnike, strojarstva i brodogradnje, 2007.

[6] I. Alfirević, *Nauka o čvrstoći I*. Zagreb: Tehnička knjiga, 1995.

[7] I. Alfirević, *Nauka o čvrstoći II*. Zagreb: Golden marketing, 1999.

[8] B. Kraut, *Strojarski priručnik*. Zagreb: SAJEMA D.O.O., 2009.

[9] N. Ninić, *Uvod u termodinamiku i njene tehničke primjene*. Split: Fakultet elektrotehnike, strojarstva i brodogradnje, 2008.

[10] N. Ninić, *Elementi prijenosa topline i tvari*. Split: Fakultet elektrotehnike, strojarstva i brodogradnje, 2002.

[11] J. Damir, *Elementi strojeva - I dio*. Split: Sveučilište u Splitu, Fakultet elektrotehnike, strojarstva i brodogradnje, 2007.

[12] A. Šegota, *Financijska matematika*. Rijeka: Ekonomski fakultet Sveučilišta u Rijeci, 2012.

[13] R. Deželić, *Metali u strojogradnji*. Split: Fakultet elektrotehnike, strojarstva i brodogradnje, 1975.

[14] I. Vinko, M. Franz, Đ. Španiček, and L. Ćurković, *Materijali I, Novo izmijenjeno izdanje*. Zagreb: Fakultet strojarstva i brodogradnje, 2014.

[15] Ž. Lozina, *Dinamika*. Split: Fakultet elektrotehnike, strojarstva i brodogradnje, 2005.

[16] F. M. White, *Fluid Mechanichs, 7th Edition*. 2011.

[17] Y. A. Cengel and M. A. Boles, *Thermodynamics: an Engineering Approach 8th Edition*. 2015.