Možemo i nešto bolje od ovoga. Pošto je pobrojani tip cijeli broj, možemo deklarisati niz sa imenima dana te koristiti pobrojani tip kao indeks u nizu:

```
1 enum DanUSedmici { Ponedjeljak, Utorak, Srijeda, Cetvrtak, Petak, Subota,
     Nedjelja };
2 char imena_dana[7][12] = { "Ponedjeljak", "Utorak", "Srijeda", "Cetvrtak",
      "Petak", "Subota", "Nedjelja" };
3 ...
4 enum DanUSedmici dan = Subota;
5 printf("%s", imena_dana[dan]);
```

Najprije obratimo pažnju na deklaraciju niza `imena_dana` u liniji 2. U pitanju je niz stringova, odnosno matrica tipa `char`↑ pri čemu je svaki red matrice dužine 12 znakova, što je dovoljno za najduže ime dana (Ponedjeljak). Ukupno matrica ima 7 redova. Ovaj niz smo inicijalizirali sa 7 stringova, zbog čega smo u inicijalizaciji koristili dvostruke navodnike. U liniji 5 ispisali smo na ekranu ime dana sadržanog u promjenljivoj `dan` pobrojanog tipa `DanUSedmici`. Znamo da ova promjenljiva ima vrijednost cijelog broja iz [0,6], a ako to nije sigurno možemo pisati:

↑ Poglavlje C.3.2, str. 386.

```
if (dan < Ponedjeljak || dan > Nedjelja)
  printf("Ilegalan dan!");
else
  printf("%s", imena_dana[dan]);
```

Vidimo u liniji 5 da je korišten format `%s` jer na ekranu ispisujemo string, te da je promjenljiva `dan` korištena kao indeks u nizu `imena_dana`, dakle dobili smo odgovarajući red matrice što je string sa nazivom dana "Subota" kojeg ispisujemo na ekranu.

9.3 *Strukturni tip podataka (`struct`)*

Do sada smo se upoznali sa ključnim riječima `typedef` i `enum` koje nam ustvari ne nude ništa suštinski novo, nego samo da damo preglednija imena postojećim tipovima podataka i vrijednostima. Za razliku od njih, *strukturni tip podataka*, koji se kraće naziva i **struktura**, omogućuje nam da definišemo novi tip podataka koji se sastoji od više elemenata nekog od poznatih tipova.

Npr. u mnogim programima potrebno je raditi sa datumima. Primijetili smo da, za razliku od nekih drugih programskih jezika, C ne posjeduje tip podataka *Date*, pa smo umjesto toga koristili tri promjenljive tipa `int`. Npr. ako smo koristili u programu tri datuma, moglo se desiti da imamo promjenljive koje se zovu *dan_rodjenja, mjesec_rodjenja, godina_rodjenja, dan_upisa, mjesec_upisa...* što je prilično nepregledno. Umjesto toga, možemo definisati novi strukturni tip pod nazivom "Datum", čiji su elementi cjelobrojne promjenljive "dan", "mjesec" i "godina".

9.3.1 *Definicija strukturnog tipa i deklaracija promjenljivih*

Definicija strukture započinje ključnom riječi `struct`, nakon koje slijedi *naziv strukture* koji se još naziva i *tag*. U vitičastim zagradama navodimo *članove ili elemente strukture* koristeći istu sintaksu kao da deklarišemo promjenljive. Nakon zatvorene vitičaste zagrade nalazi se znak tačka-zarez, kao i kod `enum`. Ovo bi bila naša struktura `Datum`:

```
struct Datum {
  int dan;
  int mjesec;
  int godina;
};
```

Elementi strukture su tri cjelobrojne promjenljive koje smo nazvali dan, mjesec i godina. Mogli smo koristiti i kraću deklaraciju:

```
struct Datum { int dan, mjesec, godina; };
```

Obratite pažnju da se nakon deklarisanja svakog elementa (ili grupe elemenata istog tipa) nalazi znak tačka-zarez, te da se znak nalazi i nakon zatvorene vitičaste zagrade. Pravila za imenovanje struktura i članova struktura su kao i za ostale C identifikatore:↑ naziv se može sastojati od velikih slova, malih slova, cifara i znaka donja crtica (undescore), pri čemu ne može počinjati cifrom. Ipak, mi ćemo kod imenovanja struktura koristiti CamelCase notaciju kod koje svaka riječ počinje velikim i nastavlja se malim slovima, a riječi međusobno nisu razdvojene, dok članove strukture imenujemo malim slovima kao i promjenljive.

↑ Poglavlje 1.2.3, str. 29.

Da bismo shvatili šta se tačno dešava u primjeru, odgovorimo na pitanje: koliko memorije je zauzeto ovom naredbom?

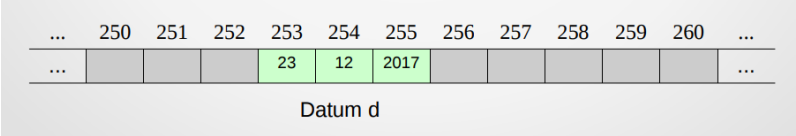
Odgovor glasi: nimalo, jer nije ništa deklarisano. Kada smo naveli definiciju tipa `Datum`, mi smo u biti samo dali opis od čega se jedan datum sastoji, ali još uvijek nismo deklarisali niti jednu promjenljivu tog tipa. Da bismo deklarisali tu promjenljivu, navodimo ponovo ključnu riječ `struct` (nije potrebno u C++u), naziv tipa, te zatim naziv jedne ili više promjenljivih tog tipa, npr:

```
struct Datum { int dan, mjesec, godina; };
struct Datum d; // u C++ moze samo: Datum d
```

Promjenljiva `d` sastoji se od tri cjelobrojna člana: `dan`, `mjesec` i `godina`. Pošto je svaki od tih članova tipa `int`, što znači da na PC platformi tipično zauzima 4 bajta, za promjenljivu `d` možemo reći da zauzima 12 bajta memorije. No memorijsko zauzeće strukturne promjenljive nije uvijek jednako zbiru veličina članova strukture.[2] Naime, na PC arhitekturi podaci u memoriji moraju biti *poravnati* (eng. aligned) na veličinu *riječi* (eng. word), koja na PC arhitekturi iznosi 4 bajta. Tako npr. strukturni tip koji se sastoji od promjenljivih tipa `float` i `short` neće zauzimati 6 nego 8 bajta, dok recimo niz od 10 elemenata tipa `char` kao član strukture će zauzeti 12 bajta.

[2] Na poravnanje utiče čak i redoslijed kojim su navedeni članovi strukture u njenoj definiciji. Npr. ako u strukturi imate elemente tipa `char`, `int` i `short`, biće zauzeto 12 bajta jer će svaki od njih biti zasebno poravnat, ali ako promijenite redoslijed u `int`, `short`, `char` struktura će zauzimati 8 bajta.

Kako to ustvari izgleda u memoriji? Na slici 9.1 možemo vidjeti tri cjelobrojne vrijednosti koje se nalaze uzastopno u memoriji, a sve zajedno čine promjenljivu `d`.



Slika 9.1: Izgled promjenljive `struct Datum d` u memoriji. Ovdje "adrese" predstavljaju apstraktne memorijske lokacije pri čemu svaka promjenljiva zauzima jednu lokaciju.

Kada jednom definišemo strukturu, možemo deklarisati koliko želimo promjenljivih tog tipa. Zbog toga se strukture u pravilu definišu u globalnom opsegu da bi se mogle koristiti u svim funkcijama u programu. Ako bismo posebno definisali strukturu u svakoj funkciji, moglo bi se desiti da se te definicije suptilno razlikuju, i generalno ne želimo da tjeramo programere da provjeravaju da li je definicija na svim mjestima jednaka. Pored toga, funkcije ne mogu primati niti vraćati promjenljive strukturnog tipa ako one nisu definisane u globalnom opsegu. Kada strukturu definišete na samom početku programa (ispod `#include` direktiva) time obavještavate programera da je definicija jedinstvena za program i da je može slobodno koristiti bilo gdje.

Kao i kod pobrojanog tipa, uočavamo da se nakon zatvorene vitičaste zagrade nalazi znak tačka-zarez, što je izuzetak od pravila datog na samom početku knjige. Razlog je to što ključna riječ `struct` omogućuje da odmah nakon definicije strukture deklarišemo i neke promjenljive. Sljedeća linija koda sadrži definiciju strukture `Datum`, koju možete koristiti u svim narednim linijama koda, te odmah deklariše i promjenljivu `d` koja je tipa `Datum`.

```
struct Datum { int dan, mjesec, godina; } d;
```

Česta greška prilikom definisanja struktura je da zaboravimo ovaj posljednji tačka-zarez:

```
struct Datum {
  int dan;
  int mjesec;
  int godina;
}
int x;
```

Za ovaj kod prevodilac će nam prijaviti grešku da riječ `int` nije validan naziv za promjenljivu jer je u pitanju rezervisana riječ. Naime, pošto nakon definicije strukture `Datum` nema tačka-zareza, očekuje se da je sljedeća riječ naziv promjenljive tog tipa, a rekli smo da za programski jezik C nema velike razlike između novih redova i razmaka.

9.3.2 *Anonimne strukture*

Jedna interesantna osobina je da, ako prilikom definisanja strukture odmah navedemo i promjenljivu tog tipa, ne moramo davati naziv

strukturi. Ovakva struktura se naziva *anonimna struktura*. Npr. možemo pisati:

```
struct { int dan, mjesec, godina; } d;
```

Ovim smo definisali anonimnu strukturu koja se sastoji od tri cjelobrojne promjenljive dan, mjesec i godina, te smo deklarisali promjenljivu tog tipa pod nazivom `d`. Ako bismo željeli kasnije deklarisati strukturnu promjenljivu sa istim elementima, morali bismo ponovo pisati kompletnu definiciju strukture:

```
struct { int dan, mjesec, godina; } d2;
```

Zbog ovoga se anonimne strukture koriste samo kada želimo opisati neki strukturni tip koji se nikada više neće koristiti osim na tom mjestu.

9.3.3 *Imenovanje strukturnog tipa pomoću* `typedef`

U literaturi i na Internetu ćete često vidjeti da se strukturni tipovi imenuju koristeći ključnu riječ `typedef` koju smo opisali u poglavlju 9.1. Čak je na mnogim mjestima to i jedini način imenovanja struktura, pa bi oni koji su naučili raditi na taj način mogli zaključiti da je opis dat u ovoj knjizi zbunjujući. Imenovanje strukturnih tipova pomoću `typedef` omogućuje nam da u kasnijim deklaracijama promjenljivih ne moramo svaki put ponavljati ključnu riječ `struct`.

Rekli smo da se `typedef` koristi tako što najprije deklarišemo "promjenljivu" željenog tipa, a zatim ispred dodajemo ključnu riječ `typedef`, čime "promjenljiva" postaje ustvari naziv tipa kojeg možemo koristiti u programu. Kod strukturnog tipa "promjenljiva" je ustvari anonimna struktura a naziv "promjenljive" je naziv tipa. Pogledajmo deklaraciju strukturnog tipa `Datum` pomoću `typedef`:

```
1 typedef struct {
2   int dan, mjesec, godina;
3 } Datum;
4 Datum d;
```

Najprije je deklarisana anonimna struktura (kao u primjerima iznad) pod imenom `Datum`. Dakle, deklarisana je promjenljiva koja se zove `Datum` a koja je tipa anonimne strukture koja se sastoji od dana, mjeseca i godine. Zatim smo ispred ove deklaracije dodali ključnu riječ `typedef` čime je promjenljiva `Datum` ustvari postala novi tip pod imenom `Datum`. Sada se u liniji 4 deklariše promjenljiva `d` tog novog tipa. Obratite pažnju da kod ovakve deklaracije ne moramo ponovo navoditi ključnu riječ `struct`.

9.3.4 *Rad sa strukturnim promjenljivim*

Nakon što smo deklarisali promjenljivu `d` tipa `Datum`, šta možemo raditi s njom? Direktno ne možemo ništa osim međusobno dodjelji-

vati promjenljive, jer je to tip podataka koji nije kao takav poznat programskom jeziku C ni njegovoj standardnoj biblioteci. Međutim, možemo pristupati članovima strukture koji su ustvari obične promjenljive nekog od poznatih tipova, u našem slučaju tipa `int`. Za pristup članovima strukture koristi se *operator tačka*. Način korištenja operatora biće jasniji na primjeru:

```
1  struct Datum {
2    int dan, mjesec, godina;
3  };
4  struct Datum d;
5  d.dan = 21;
6  scanf("%d", &d.mjesec);
7  d.godina = d.dan * 100 - 83;
```

Promjenljiva `d` je strukturnog tipa `Datum`. Pomoću operatora tačka pristupamo članovima strukture navedenim u definiciji strukturnog tipa (linija 2).[3] Dakle, kada smo u liniji 5 napisali `d.dan` dobili smo običnu promjenljivu tipa `int`. Isto važi i za `d.mjesec` i `d.godina` u linijama 6 i 7. Pošto su ovo obične promjenljive tipa `int`, možemo im pridruživati vrijednost (linija 5), unositi ih koristeći `scanf` (linija 6), ili ih koristiti u računskim operacijama (linija 7).

[3] Obratite pažnju na prioritet i asocijativnost operatora tačka date u tabeli u prilogu A (str. 351). Operator tačka ima najviši prioritet, te je jednak prioritetu operatora [] i () (pristup članu niza i poziv funkcije), a svi ovi operatori su lijevo asocijativni.

Koju vrijednost imaju dan, mjesec i godina nakon linije 4 tj. nakon što smo tek deklarisali promjenljivu strukturnog tipa? Kao i sa običnim promjenljivim, članovi strukture *nisu inicijalizirani* dok to izričito ne uradimo kao u liniji 5. Možemo istovremeno inicijalizirati sve članove strukture prilikom deklaracije promjenljive strukturnog tipa na sličan način kao što smo to ranije radili sa nizovima:↑

↑ Poglavlje 5.1.2, str. 134.

```
struct Datum d = {21, 12, 2017};
```

Kao što smo rekli, standardna C biblioteka ne poznaje naš tip `Datum`, tako da ni funkcije `printf` i `scanf` ne znaju direktno raditi sa ovim tipom podataka. Jedino rješenje je da radimo sa članovima strukture pojedinačno:

```
printf("Unesite datum: ");
scanf("%d%d%d", &d.dan, &d.mjesec, &d.godina);
printf("Unijeli ste datum: %d. %d. %d\n", d.dan, d.mjesec, d.godina);
```

Kod složenijih struktura preporučujemo da odmah napravite funkcije za unos i ispis na ekran promjenljivih ovog tipa, što će bitno skratiti vaš program.

9.3.5 *Strukture i funkcije*

Pa kad smo već tu, pogledajmo kako bi izgledale te funkcije. Prije svega, napominjemo da se promjenljive strukturnog tipa mogu *slati u funkcije i vraćati iz funkcija* podjednako kao i promjenljive bilo kojeg drugog tipa. Sjetite se da se iz funkcije može vratiti samo jednu vrijednost.↑ Strukture nam omogućuju da zaobiđemo to ograničenje:

↑ Poglavlje 6.2.1, str. 183.

jednostavno definišemo strukturu koja sadrži sve što želimo da vratimo iz funkcije i zatim vratimo promjenljivu tog strukturnog tipa.

```
1  #include <stdio.h>
2  struct Datum {
3    int dan,mjesec,godina;
4  };
5  struct Datum unesi_datum(void) {
6    struct Datum d;
7    printf("Unesite datum: ");
8    scanf("%d%d%d", &d.dan, &d.mjesec, &d.godina);
9    return d;
10 }
11 void ispisi_datum(struct Datum d) {
12   printf("%d. %d. %d", d.dan, d.mjesec, d.godina);
13 }
14 int main() {
15   struct Datum d1 = unesi_datum();
16   struct Datum d2 = unesi_datum();
17   struct Datum d3;
18   /* Racunamo d3... */
19   ispisi_datum(d3);
20   printf("\n");
21   return 0;
22 }
```

Najprije uočavamo da je strukturni tip `Datum` definisan u globalnom opsegu (iznad funkcija), kako bi isti bio dostupan iz svih funkcija. Funkcija `unesi_datum` (linije 5-10) služi za unos sa tastature jedne promjenljive tipa `Datum`. Ona nam skraćuje kod na način da ne moramo svaki put pisati `printf("Unesite datum: ");` te komplikovani `scanf` koji smo vidjeli na prethodnoj stranici. Zbog toga ova funkcija nema parametara (ne prima ništa), a vraća jednu promjenljivu strukturnog tipa.

Svaki put kada navodimo strukturni tip pišemo i ključnu riječ `struct` (što se može izbjeći primjenom `typedef`), dakle funkcija vraća vrijednost tipa `struct Datum`. U funkciji je najprije deklarisana pomoćna promjenljiva `d` (linija 6), elementi strukture su uneseni kao pojedinačne vrijednosti tipa `int` (linija 8), te je ova promjenljiva jednostavno vraćena iz funkcije naredbom `return` (linija 9).

Poziv funkcije možemo vidjeti u linijama 15 i 16 gdje je funkcija pozvana odmah prilikom deklarisanja promjenljivih `d1` i `d2`, te su ove promjenljive inicijalizirane onim što je funkcija vratila.

Program smo također pojednostavili funkcijom `ispisi_datum`, jer ne moramo svaki put kod ispisa datuma pisati kompleksan `printf` koji vidimo u liniji 12. Ova funkcija prima kao parametar datum koji treba ispisati na ekran, a ne vraća ništa jer je sav posao koji ona obavlja ispis na ekran (funkcija je tipa `void`). Primjer poziva te funkcije vidimo u liniji 19.

Zadatak 1. *Napisati program koji omogućuje unos dva datuma, a zatim na ekranu ispisuje broj dana koliko je proteklo između ta dva datuma.*

Već imamo program koji služi za unos dva datuma. Sada trebamo

izračunati traženi broj dana. Računanje razlike između dva datuma je dosta kompleksno jer svaki mjesec ima različit broj dana, a moramo uzeti u obzir i prestupne godine. Prilično jednostavno rješenje je napraviti funkciju koja izračunava broj dana između primljenog datuma i nekog referentnog datuma u prošlosti, a zatim ta dva broja oduzmemo. Kako ne bismo rukovali sa negativnim razlikama, možemo kao referentni datum uzeti 1. 1. 1. godine (mada Gregorijanski kalendar tada nije važio). Pokazuje se da brojevi sa kojim rukujemo neće biti preveliki.

Dakle, napravićemo funkciju `broj_dana` koja prima jedan datum, a vraća broj dana između 1. 1. 1. i tog datuma:

```
int broj_dana(struct Datum d);
```

Najprije ćemo izračunati broj dana koji odgovara godinama, zatim mjesecima, te ćemo na kraju sabrati dane. Godina je prestupna ako je djeljiva sa 4, a nije djeljiva sa 100, osim godina koje su djeljive sa 400. Napravićemo funkciju koja vraća 1 ako je godina prestupna, a 0 ako nije:

```
int prestupna(int godina) {
  if (godina%4 == 0 && godina%100 != 0 || godina%400 == 0)
    return 1;
  return 0;
}
```

Sada ćemo petljom na ukupan broj dana sabrati po 365 dana za svaku godinu, s tim što ćemo za prestupne sabrati još 1:

```
int ukupno=0, i;
for (i=1; i<d.godina; i++)
  ukupno += 365 + prestupna(i);
```

Slijede mjeseci. Za broj dana u mjesecu ne postoji neka jednostavna formula, pa možemo napraviti niz `bdum` (skraćeno od "broj dana u mjesecu") koji za svaki indeks `i` daje broj dana u i-tom mjesecu. Pošto indeksi u nizovima u C-u kreću od nule, deklarisaćemo niz od 13 elemenata pri čemu je prvi element nula.

```
int bdum[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Ali ako je godina prestupna, u februaru treba biti 29 dana.

```
if (prestupna(d.godina)) bdum[2]++;
```

Sada trebamo na ukupan broj dana sabrati dane za mjesece od januara do trenutnog mjeseca, dodati dane datuma i možemo vratiti tako izračunat ukupan broj dana.

```
for (i=1; i<d.mjesec; i++)
  ukupno += bdum[i];
ukupno += d.dan;
return ukupno;
```

Time je završena funkcija `broj_dana`. Ostaje da je pozovemo iz maina koji sada izgleda ovako:

```
int main() {
  struct Datum d1 = unesi_datum();
  struct Datum d2 = unesi_datum();
  int razlika = broj_dana(d2) - broj_dana(d1);
  printf("Izmedju ovih datuma je proteklo %d dana.\n", razlika);
  return 0;
}
```

Ovaj program će davati negativn broj dana ako je drugi datum prije prvog, što možemo riješiti koristeći funkciju `abs` da odredimo apsolutnu vrijednost razlike. Za dva susjedna datuma ćemo dobiti rezultat 1. Postavka zadatka nije precizno definisala da li "broj dana između dva datuma" uključuje spomenuta dva datuma ili ne, tako da smo usvojili uobičajenu konvenciju da je uključen jedan od dva datuma. Npr. ako računamo broj dana između 10. januara i 12. januara uobičajen odgovor je "dva dana" pri čemu smo uračunali 12. a nismo uračunali 10.

9.3.6 Strukture i nizovi, podstrukture

Članovi strukture mogu biti promjenljive svih poznatih tipova, pa tako i nizovi. Pogledajmo sljedeći primjer:

```
1  struct Student {
2    char ime[20];
3    char prezime[20];
4    int broj_indeksa;
5  } s;
```

Definisana je struktura `Student`, te je odmah deklarisana i promjenljiva `s` tog tipa. Elementi strukture su ime, prezime i broj indeksa. Kao što znamo, string u C-u se realizuje kao niz tipa `char`, pa je element strukture `ime` niz tipa `char` od 20 elemenata.

Kada iskoristimo *operator tačka*, dobijamo član strukture koji je kao obična promjenljiva onog tipa koji je naveden u definiciji. Dakle, kada napišemo `s.ime`, u ovom slučaju će to biti običan niz tipa `char` s kojim radimo kao i sa bilo kojim nizom tog tipa. Sljedeća naredba će na ekran ispisati ime studenta `s`:

```
printf("%s", s.ime);
```

Ako želimo da pristupimo članu niza, npr. prvom slovu imena, koristimo operator uglastih zagrada nakon imena niza, a kako je `s.ime` u cjelosti ime niza, zagrade se stavljaju nakon toga:

```
printf("%c", s.ime[0]); /* Prvo slovo imena */
```

U poznate tipove spadaju i neki drugi strukturni tipovi, pod uslovom da su definisani iznad tekuće strukture. Drugim riječima, član strukture može biti druga struktura. Takav slučaj možemo vidjeti na sljedećem primjeru:

```
1  struct Datum { int dan,mjesec,godina; };
2  struct Student {
3    char ime[20];
4    char prezime[20];
5    int broj_indeksa;
6    struct Datum datum_rodjenja;
7  } s;
```

U liniji 1 definisana je struktura `Datum`, a zatim slijedi definicija strukture `Student`. Jedan od članova ove strukture je i `datum_rodjenja` koji je tipa `struct Datum` (linija 6). Konačno, deklarisan je jedan `Student` pod nazivom `s` (linija 7).

Kada u našem programu napišemo `s.datum_rodjenja`, ovaj izraz se ponaša kao promjenljiva tipa `struct Datum`. Šta možemo raditi sa promjenljivom tog tipa? Ništa direktno (osim dodjele), ali možemo pristupiti njenim članovima koristeći operator tačka. Ovaj operator se navodi desno od naziva strukturne promjenljive, nakon čega navodimo član strukture `Datum`. Dakle, možemo pisati `s.datum_rodjenja.dan` čime pristupamo elementu `dan` strukturne promjenljive `datum_rodjenja` koji je obična promjenljiva tipa `int`:

```
printf("Unesite datum rodjenja studenta: ");
scanf("%d", &s.datum_rodjenja.dan);
```

Moguće je da se struktura nalazi unutar druge strukture, koja se nalazi unutar treće strukture i tako do u beskonačnost. Jednostavno koristimo operator tačka kako bismo se "spustili jedan nivo" u ovoj hijerarhiji struktura. Jedna česta greška je da se kod pristupa članu strukture navodi tip člana – nije ispravno pisati `s.datum_rodjenja.int` ili `s.Datum.dan` u ovom primjeru, jer su `int` i `Datum` tipovi.

Interesantna osobina strukturnih promjenljivih je da ih možemo sigurno međusobno dodjeljivati, za razliku od recimo nizova, i svi članovi strukture će se korektno kopirati – pa čak i nizovi! Npr. znamo da ako imamo dva stringa koji se zovu `ime1` i `ime2`, bilo bi neispravno pisati `ime1=ime2`. Također, ako imamo dva studenta `s1` i `s2`, nije ispravno pisati `s1.ime=s2.ime` iz istog razloga. Ali sasvim je ispravno pisati `s1=s2` čime će se kopirati između ostalog i ime.[4] No, promjenljive strukturnog tipa se ipak ne mogu međusobno upoređivati koristeći operator ==.

[4] Ako je član strukture pokazivač, kôd tipa `s1=s2` će kopirati adresu na koju pokazivač pokazuje, što u vašem konkretnom programu može biti "plitka kopija".

Zadatak 2. *Pomoću struktura definisati osnovne geometrijske oblike u Dekartovom dvodimenzionalnom prostoru. Zatim napraviti program koji omogućuje osnovne operacije nad ovim oblicima.*

Ovaj program može biti proizvoljno obiman, tako da ćemo ovdje samo dati neke osnovne principe rješavanja. Prije svega potrebno je da definišemo tačku u prostoru, a zatim preko tačke možemo definisati ostale oblike. Tačka je opisana preko koordinata x i y:

```
struct Tacka {
  double x, y;
};
```

Za koordinate naravno koristimo realne brojeve. Jedna od korisnih funkcija koja će nam zatrebati je udaljenost između tačaka. Funkcija prima dvije tačke, a vraća realan broj koji predstavlja udaljenost.

```
double udaljenost(struct Tacka t1, struct Tacka t2) {
  return sqrt( (t1.x-t2.x)*(t1.x-t2.x) + (t1.y-t2.y)*(t1.y-t2.y) );
}
```

Za funkciju `sqrt` treba nam biblioteka `math.h`.

Sada možemo definisati kružnicu preko tačke koja predstavlja centar kružnice i poluprečnika.

```
struct Kruznica {
  struct Tacka centar;
  double poluprecnik;
};
```

Kvadrat možemo definisati preko jedne tačke (npr. donjeg lijevog ugla) i dužine stranice. Ovako definisan kvadrat ima sve stranice paralelne osama koordinatnog sistema.

```
struct Kvadrat {
  struct Tacka dolje_lijevo;
  double stranica;
};
```

Pravougaonik možemo definisati na više načina. Jedan je da zadamo dvije dijagonalne tačke (npr. donji lijevi ugao i gornji desni), a drugi je preko jedne tačke i dvije dužine stranica. Ovo je opet pravougaonik čije su stranice paralelne osama. Pravougaonik i Kvadrat čije stranice nisu paralelne osama bi bilo nešto teže zadati. Jedan način je da u strukturu dodamo i ugao rotacije oblika (u radijanima) u odnosu na polazno stanje.

Pogledajmo i neke funkcije. Recimo možemo napraviti funkciju koja prima kružnicu i tačku, te provjerava da li se tačka nalazi unutar kružnice:

```
int unutar_kruznice(struct Kruznica k, struct Tacka t) {
  if (udaljenost(t, k.centar) < k.poluprecnik) return 1;
  return 0;
}
```

Ako bismo željeli provjeriti da li se tačka nalazi *na kružnici*, morali bismo uzeti u obzir nepreciznost realnih brojeva↑ jer funkcija `sqrt` može vratiti brojeve sa dosta velikim brojem decimala.

↑ Poglavlje 3.1.6, str. 72.

```
int na_kruznici(struct Kruznica k, struct Tacka t) {
  if (udaljenost(t, k.centar) - k.poluprecnik < EPSILON) return 1;
  return 0;
}
```

Provjera da li se dvije kružnice preklapaju bi izgledala ovako:

```
int kruznice_preklapanje(struct Kruznica k1, struct Kruznica k2) {
  if (udaljenost(k1.centar, k2.centar) < k1.poluprecnik + k2.poluprecnik)
   return 1;
  return 0;
}
```

9.3.7 *Nizovi struktura*

Vratimo se na nizove. Možemo deklarisati niz vrijednosti bilo kojeg poznatog tipa, pa tako i niz promjenljivih strukturnog tipa. Za primjer posmatraćemo sljedeću deklaraciju:

```
1 struct Datum { int dan,mjesec,godina; };
2 struct Student {
3   char ime[20];
4   char prezime[20];
5   int broj_indeksa;
6   struct Datum datum_rodjenja;
7 };
8 struct Student studenti[100];
```

U liniji 8 deklarisan je niz od 100 vrijednosti čije je ime `studenti`, a tip svakog člana niza je `struct Student`. Kako radimo sa ovakvim nizom? Kao i kod svakog niza, najprije trebamo pristupiti nekom članu niza koristeći uglaste zagrade. Recimo, prvi član ovog niza bi bio `studenti[0]`. Ovo je sada promjenljiva tipa `struct Student` sa kojom radimo kao i sa bilo kojom promjenljivom tog tipa. Recimo, ako bismo željeli na ekran ispisati ime studenta koji je prvi član niza, mogli bismo pisati:

```
printf("%s", studenti[0].ime);
```

Obratite pažnju da je kompletan kod `studenti[0]` jedna promjenljiva tipa `struct Student`, pa na tu promjenljivu primjenjujemo operator tačka koji se navodi desno od nje (nakon uglastih zagrada). Zatim desno od tačke navodimo član strukture `ime`. Sada je `studenti[0].ime` promjenljiva tipa string (niz tipa `char`), pa je možemo ispisati koristeći format `%s`. Ako bismo željeli ispisati prvo slovo imena, pisali bismo `student[0].ime[0]`. Ako bismo željeli pomoću tastature unijeti mjesec rođenja sedmog studenta, kod bi glasio:

```
scanf("%d", &studenti[6].datum_rodjenja.mjesec);
```

> ***Zadatak 3.*** *Napisati funkciju "stariji_od" koja prima niz studenata i neki datum, te na ekran ispisuje imena i prezimena studenata (u zasebnim redovima) koji su rođeni prije primljenog datuma a broj indeksa im je paran broj.*

Najprije ćemo napraviti jednu pomoćnu funkciju koja poredi dva datuma, te vraća 1 ako je prvi datum prije drugog, a 0 ako nije.

```
1 int datum_prije(struct Datum d1, struct Datum d2) {
2   if (d1.godina < d2.godina) return 1;
```

```
3     if (d1.godina == d2.godina) {
4       if (d1.mjesec < d2.mjesec) return 1;
5       if (d1.mjesec == d2.mjesec && d1.dan < d2.dan) return 1;
6     }
7     return 0;
8   }
```

Sada nam je lako napisati funkciju `stariji_od`:

```
1   void stariji_od(struct Student* studenti, int vel, struct Datum d) {
2     int i;
3     for (i=0; i<vel; i++) {
4       if (datum_prije(studenti[i].datum_rodjenja, d) &&
5         studenti[i].broj_indeksa % 2 == 0)
6           printf("%s %s\n", studenti[i].ime, studenti[i].prezime);
7     }
8   }
```

Funkcija je tipa `void` jer ne vraća ništa, posao koji obavlja je ispis imena i prezimena na ekran. Funkcija prima niz studenata, što znači da su joj parametri pokazivač na prvi član niza i broj članova niza, a treći parametar je datum `d` sa kojim upoređujemo datume rođenja studenata.

U funkciji ćemo napraviti petlju koja prolazi kroz sve članove niza, i-ti član niza je `studenti[i]`, a tipa je `struct Student`. Sada možemo pozvati funkciju `datum_prije` da provjerimo da li je datum rođenja prije datog. Kao prvi parametar funkcije poslaćemo datum rođenja i-tog studenta: `studenti[i].datum_rodjenja`.

Funkcija vraća logičku vrijednost tako da možemo odmah preći i na drugi uslov koji treba provjeriti, a to je da li je broj indeksa paran. Broj indeksa i-tog studenta je `studenti[i].broj_indeksa` pa uzimamo modulo ove vrijednosti sa 2. Konačno, u liniji 6 ispisujemo ime i prezime studenta ako je uslov ispunjen.

9.3.8 Pokazivači na strukture

Možemo deklarisati pokazivač pomoću kojeg pokazujemo na vrijednost bilo kojeg od poznatih tipova. Tako bi deklaracija pokazivača na vrijednost tipa `struct Student` glasila:

```
struct Student* p;
```

Znak zvjezdica se navodi nakon tipa a prije naziva promjenljive.

Podsjetimo se da ime niza predstavlja adresu prvog člana niza, pa tako ovaj pokazivač p sada može pokazivati na prvi član ranije deklarisanog niza studenti:

```
struct Student* p = studenti;
```

Da bismo pristupili nekom članu strukture preko pokazivača, najprije moramo dereferencirati pokazivač. Podsjetimo se da se to radi primjenom operatora dereferenciranja (zvjezdica): `*p`. Ako bismo pisali recimo `*p.broj_indeksa` ne bismo dobili željeni rezultat jer operator tačka ima veći prioritet od operatora zvjezdica, pa bi ovo bilo isto što

i `*(p.broj_indeksa)`. Dakle prevodilac će `p` posmatrati kao neku strukturnu promjenljivu čiji je član pokazivač `broj_indeksa`, a zatim će pokušati da dereferencira ovaj pokazivač. Umjesto toga trebamo pisati ovako:

```
(*p).datum_rodjenja.dan = 23;
```

Pošto je pokazivač na strukturu vrlo česta pojava u programiranju, osmišljen je poseban *operator strelica* koji istovremeno dereferencira pokazivač i pristupa članu strukture, a piše se kao znak minus nakon kojeg slijedi znak veće-od. Ovaj operator ujedno ima vrlo visok prioritet, kao i tačka. Tako je prethodni kod ekvivalentan sa sljedećim:

```
p->datum_rodjenja.dan = 23;
```

Obratite pažnju da strelicu stavljamo nakon pokazivača `p`, ali nakon člana strukture `p->datum_rodjenja` se nalazi tačka jer to nije pokazivač nego obična strukturna promjenljiva.

Zadatak 4. *Ranije napisanu funkciju "stariji_od" prepisati koristeći pokazivačku aritmetiku.*

```
1 void stariji_od(struct Student* studenti, int vel, struct Datum d) {
2   struct Student* p = studenti;
3   while (p < studenti+vel) {
4     if (datum_prije(p->datum_rodjenja, d) && p->broj_indeksa % 2 == 0)
5       printf("%s %s\n", p->ime, p->prezime);
6     p++;
7   }
8 }
```

Pokazivač na strukturu može i sam biti član strukture, pa se na taj način realizuju dinamičke strukture kao što su liste i stabla.

9.3.9 *Bitpolja*

Strukture nam daju još jednu interesantnu mogućnost, a to je da definišemo cjelobrojne tipove promjenljivih čija je veličina (u bitima) proizvoljna. Jedini uslov je da takva promjenljiva bude član strukture. Recimo, na PC platformi tip `int` se sastoji od 32 bita (4 bajta), `short` ima 16 bita, a `char` 8 bita. Ali šta ako želimo cjelobrojnu promjenljivu od recimo 12 bita? To možemo postići sljedećom definicijom:

```
struct NekiTip {
  int x:12;
};
```

Broj iza dvotačke označava broj bita uključujući predznak koje će imati cjelobrojni član strukture `x`. Ako ne želimo predznak, možemo pisati `unsigned int`.

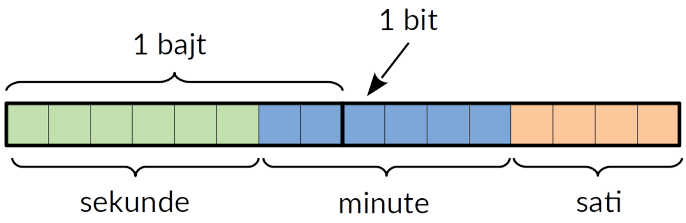
Ova funkcionalnost se naziva **bitpolja** (eng. bitfields).

Jedna od primjena bitpolja je da kreiramo strukture koje zauzimaju vrlo malu količinu memorije, a sadrže veći broj elemenata koji su "up-

akovani" (eng. packed) u manji prostor. Pogledajmo definiciju strukture `Vrijeme`:

```
struct Vrijeme {
  unsigned int sati:4, minute:6, sekunde:6;
};
```

Struktura `Vrijeme` se sastoji od tri cjelobrojne vrijednosti: sati, minuta i sekundi. Da smo koristili obične vrijednosti tipa `int`, svaka promjenljiva tipa `Vrijeme` zauzimala bi 12 bajta odnosno 96 bita. Koristeći bitpolja čini se da smo postigli da promjenljiva zauzima svega 4+6+6=16 bita odnosno dva bajta. Naime, znamo da su sati u opsegu 0-11 što znači da su nam dovoljna 4 bita (vrijednosti 0-15), dok za minute i sekunde koristimo po 6 bita (opseg 0-63) (sl. 9.2).



Slika 9.2: Organizacija strukture Vrijeme u memoriji

No, ipak to neće biti slučaj. Pored ranije spominjanog poravnanja,↑ za bitpolja važi da strukturni tip ne može zauzimati manje memorije od osnovnog tipa, a to je u ovom slučaju `unsigned int` koji zauzima 4 bajta, pa trebamo prepisati strukturu tako da se koristi tip `unsigned char`:

↑ Str. 304.

```
struct Vrijeme {
  unsigned char sati:4, minute:6, sekunde:6;
};
```

Uočavamo da je ipak potrebno sate predstavljati u opsegu 0-23, pa ćemo promijeniti definiciju strukture tako da koristimo 5 bita za sate. Da li sada jedna promjenljiva tipa Vrijeme zauzima 17 bita? Ipak ne, takva promjenljiva bi zauzimala 24 bita jer nije moguće direktno raditi sa memorijskim blokovima manjim od jednog bajta tj. osam bita. Drugim riječima, memorijski zahtjevi jedne strukturne promjenljive će biti zaokruženi na cijeli broj bajta. Neko bi mogao uočiti da isto zauzeće od 3 bajta (24 bita) postižemo i bez bit-polja, koristeći tri obične vrijednosti tipa `unsigned char`.

Obratite pažnju da su operacije nad bitpoljima u pravilu sporije od operacija nad običnim strukturnim promjenljivim jer ih procesor računara direktno ne podržava, pa će prevodilac generisati mašinski kod koji svaki put kada radite sa ovakvom promjenljivom vrši matematičke operacije nad bitima (shift, and, or) kako bi bio postignut očekivani rezultat.

9.4 *Unija (`union`)*

Da bismo zaokružili priču o vlastitim tipovima podataka, upoznaćemo se sa još jednom mogućnošću C-a koja se rijetko koristi, a prividno je vrlo slična strukturama. U pitanju je **unija**.

Deklaracija unije je vrlo slična deklaraciji strukture, osim što se koristi ključna riječ `union`:

```
union Unija {
  int a, b, c;
} u;
u.a = 15;
u.b = 32;
printf("%d", u.a);
```

Na prvi pogled unija se ponaša kao i struktura uz jednu vrlo bitnu razliku: svi članovi unije zauzimaju istu memorijsku lokaciju! To znači da će prethodni program ispisati broj 32, jer članovi strukture `u.a` i `u.b` zauzimaju iste memorijske lokacije tako da kada promijenimo jedan, promijenili smo i drugi.

U prethodnom primjeru promjenljiva `u` zauzima 4 bajta (kao i jedna promjenljiva tipa `int`) jer je zauzeta samo jedna memorijska lokacija a ne tri.

Članovi unije ne moraju biti istog tipa. Ako tipovi članova unije imaju različitu veličinu, sama unija će imati veličinu najvećeg tipa. Npr. u sljedećem primjeru:

```
union Unija {
  char x;
  double y;
  float z;
} u;
```

najveći korišteni tip je double (8 bajta), pa će promjenljiva `u` zauzimati 8 bajta.

Unija se koristi kada želimo da iste podatke posmatramo na više različitih načina. Pogledajmo ovaj primjer:

```
union Unija {
  int i;
  float f;
  unsigned char c[4];
} u;
u.f = 0.1;
printf("%d", u.i);
```

Sva tri člana unije imaju istu dužinu (4 bajta), te zauzimaju istu memorijsku lokaciju. Kada pristupimo članu `u.f` i dodijelimo mu vrijednost 0.1, mi ovu memorijsku lokaciju posmatramo kao realnu promjenljivu kodiranu IEEE-754 standardom. S druge strane, kada joj pristupimo preko člana `u.i` mi isti binarni zapis tumačimo kao predznačnu cijelu vrijednost. Ovu vrijednost možemo dodatno razložiti kao četiri osmobitna cijela broja (4 člana niza tipa `unsigned char`).