Strukture

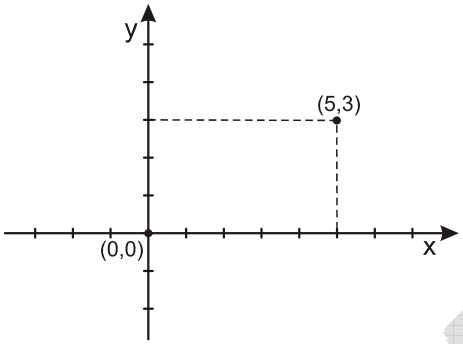
Struktura (slog) podataka je skup više promenljivih koje su grupisane pod zajedničkim imenom radi lakše manipulacije. Strukture pomažu u organizovanju složenih podataka tako što omogućavaju grupisanje međusobno povezanih promenljivih na takav način da se one posmatraju kao celina, a ne kao pojedinačni entiteti.

Primer strukture može biti dosije studenta koji sadrži ime, prezime, adresu, broj indeksa itd. Elementi struktura mogu biti takođe strukture. Na primer, dosije studenta može da sadrži i podatke o položenim ispitima i ostvarenim ocenama. Drugi primer može biti tačka u ravni koja sadrži *x* i *y* koordinatu ili pravougaonik koji sadrži dve tačke.

Osnovno o strukturama

Da bismo pokazali neke osnovne osobine struktura kreiraćemo nekoliko struktura pogodnih za korišćenje u grafici.

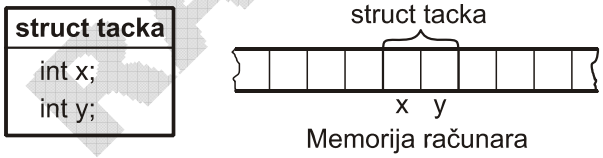
Osnovni objekat u grafici je tačka (posmatraćemo tačku u ravni), za koju ćemo pretpostaviti da je definisana celobrojnim koordinatama *x* i *y*.



Slika ### Tačka u ravni

Ovi podaci mogu biti smešteni u strukturu koja je definisana kao

```
struct tacka
{
   int x;
   int y;
}
```



Slika ### Shematski prikaz strukture *tacka*

Struktura se deklariše korišćenjem rezervisane reči **struct** iza čega, unutar vitičastih zagrada, sledi spisak promenljivih koje pripadaju strukturi (u ovom slučaju celobrojna *x* i *y* koordinata). Neposredno iza reči struct može stajati i naziv strukture, takozvani **strucure tag** (u našem slučaju *tacka*). Ovaj naziv se može kasnije koristiti kao skraćenica za deklaraciju strukture.

Promenljive navedene unutar strukture se nazivaju **promenljive članicama** ili samo **članice**. S obzirom da su promenljive članice zatvorene unutar strukture, dve članice unutar različitih struktura mogu imati iste nazive.

Rezervisana reč **struct** definiše složeni tip podatka. Međutim, iza zatvorene vitičaste zagrade može biti navedena i lista promenljivih tog tipa, baš kao što je to slučaj i sa deklarisanjem promenljivih bilo kog drugog tipa. Na primer:

```
struct ...
{
   ...
} a, b, c;
```

deklariše *a*, *b* i *c* kao promenljive navedenog tipa i obezbeđuje memorijski prostor za njih, isto kao što linija

```
int a, b, c;
```

deklariše *x*, *y* i *z* kao promenljive tipa int.

Definicija strukure koja nije praćena promenljivama ne zauzima nikakav memorijski prostor, već samo opisuje sadržaj navedene strukture. Ukoliko definicija sadrži tag, ona se kasnije može koristiti za deklarisanje instanci te strukture. Tako se neka konkretna tačka može deklarisati kao:

```
struct tacka t;
```

čime je naznačeno da je promenljiva *t* struktura tipa *struct tacka*. Struktura može biti inicijalizovana tako što se iza njene deklaracije navodi znak dodele, a zatim u vitičastim zagradama lista inicijalnih vrednosti promenljivih unutar strukture:

```
struct tacka max_t = {1024, 768};
```

Struktura se takođe može inicijalizovati dodeljivanjem vrednosti neke druge strukure istog tipa.

Članicama neke strukure se pristupa tako što se navodi naziv strukture iza koje sledi naziv promenljive članice odvojen tačkom:

naziv_strukture.clanica

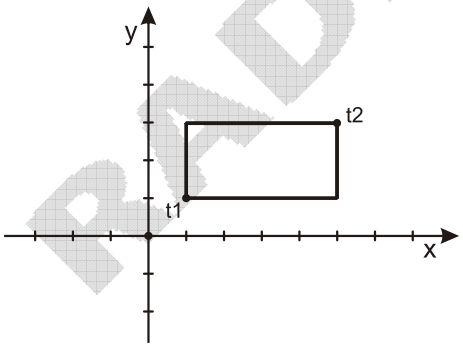
Na primer, sledeća linija štampa koordinate tačke *t*:

```
printf("%d,%d", t.x, t.y);
```

a rastojanje između dve tačke bi se računalo kao:

```
rastojanje = sqrt( (double)t.x*t.x + (double)t.y*t.y );
```

Kao što smo ranije pomenuli, članice strukture mogu biti i druge strukture. Kao primer navodimo pravougaonik koji je definisan pomoću dve tačke (donje levo i gornje desno teme).



Slika ### Pravougaonik definisan pomoću dve tačke

Struktura koja opisuje pravougaonik se može definisati na sledeći način:

```
struct pravougaonik
{
   struct tacka t1;
   struct tacka t2;
}
```

Struktura *pravougaonik* sadrži dve strukture *tacka*. Ukoliko deklarišemo promenljivu *slika* kao

```
struct pravougaonik slika;
```

onda

```
slika.t1.x
```

označava *x* koordinatu donjeg levog temena slike.

Strukture i funkcije

Jedine dopuštene operacije nad strukturom su kopiranje, pridruživanje kao celine, dobijanje njene adrese pomoću operatora & i pristupanje njenim članicama. Kopiranje i pridruživanje uključuje takođe i prosleđivanje parametara funkcijama i vraćanje vrednosti kao rezultat funkcije. Upoređivanje struktura nije moguće. Struktura može biti inicijalizovana navođenjem liste konstantnih vrednosti članica ili pridruživanjem neke druge strukture.

Razmotrimo ponašanje struktura na primerima nekoliko funkcija koje manipulišu tačkama i pravougaonicima. Postoji najmanje tri moguća pristupa: prosleđivanje pojedinačnih komponenata, prosleđivanje čitave strukture ili prosleđivanje pokazivača na strukturu. Svaki od ovih pristupa ima dobre i loše strane.

Prva funkcija, *napravi_tacku*, uzima dva cela broja, a vraća strukturu *tacka*:

```
/* napravi_tacku: kreira tacku od x i y komponente */
struct tacka napravi_tacku(int x, int y)
{
   struct tacka pom;

   pom.x = x;
   pom.y = y;

   return pom;
}
```

Obratite pažnju na to da ne postoji konflikt između argumenata funkcije i članica strukture, iako imaju iste nazive. Naprotiv, isti nazivi naglašavaju povezanost ovih promenljivih.

Funkcija *napravi_tacku* se sada može iskoristiti za dinamičku inicijalizaciju ili za prosleđivanje struktura funkciji:

```
struct pravougaonik slika;
struct tacka centar;

slika.t1 = napravi_tacku(0, 0);
slika.t2 = napravi_tacku(XMAX, YMAX);
centar = napravi_tacku( (slika.t1.x+slika.t2.x)/2,
                        (slika.t1.y+slika.t2.y)/2 );
```

Sledeći korak je definisanje funkcija za aritmetičke operacije nad tačkama. Na primer:

```
/* saberi_tacke: sabira dve tacke */
struct tacka saberi_tacke(struct tacka t1, struct tacka t2)
{
   t1.x += t2.x;
   t1.y += t2.y;
   return t1;
}
```

U ovom slučaju i argumenti i vrednost koju funkcija vraća su strukture. Umesto da koristimo pomoćnu promenljivu, u ovom primeru smo uvećavali komponente strukture *p1*, kako bismo nagalsili da se strukture prosleđuju funkciji preko vrednosti kao i svi ostali parametri.

Ukoliko je potrebno velike strukture proslediti funkciji, onda je znatno efikasnije funkciji proslediti pokazivač na strukturu, kako bi se izbeglo kopiranje čitave strukture u lokalnu promenljivu funkcije. Pokazivači na strukture se deklarišu kao i pokazivači na bilo koji drugi tip promenljive. Na primer:

```
struct tacka *pt;
```

deklariše *pt* kao pokazivač na strukturu tipa *struct tacka*. S obzirom da *pt* pokazuje na strukturu *tacka*, onda **pt* predstavlja strukturu, a *(*pt).x i (*pt).y* članice strukture. Sledeći primer pokazuje korišćenje pokazivača na strukturu:

```
struct tacka centar, *pt;

pt = &centar;
printf("Centar je (%d,%d)\n", (*pt).x, (*pt).y);
```

Korišćenje zagrada u izrazu *(*pt).x* je obavezno zato što operator . ima veći prioritet od operatora *. Kako bi se skratilo pisanje ovakvog izraza uvedena je notacija:

pokazivac->clanica

pa se poslednji red prethodnog primera može kraće zapisati kao:

```
printf("Centar je (%d,%d)\n", pt->x, pt->y);
```

Nizovi struktura

Pretpostavimo da želimo da napišemo program koji će praviti rang-listu studenata prema proseku ocena. U tom slučaju mogli bismo da deklarišemo nekoliko nizova koji bi čuvali osnovne podatke o studentima na sledeći način:

```
int br_indeksa[MAX_STUD];
char *ime[MAX_STUD];
float prosek[MAX_STUD];
```

Međutim, postojanje ovakvih paralelnih nizova, čiji su elementi u međusobnoj relaciji, ukazuje na potrebu za drugačijom organizacijom podataka, kojom bi se međusobno povezani elementi grupisali u vidu struktura. Nizovi struktura se deklarišu kao i nizovi bilo kog drugog tipa, pa bi u ovom slučaju odgovarajuća deklaracija bila:

```
struct student
{
   int br_indeksa;
   char *ime;
   float prosek;
} studenti[MAX_STUD];
```

Na ovaj način definisana je struktura *student*, koja sadrži osnovne podatke o studentu (*br_indeksa*, *ime* i *prosek*), a zatim deklarisan niz *studenti* čiji su elementi tipa *struct student*. Ovakva organizacija podataka je potpuno prirodna, jer se na ovakav način grupišu međusobno povezani podaci, čime se olakšava rad i smanjuje mogućnost greške.