Osnove inženjerske informatike II.

Uvod u programiranje

Nizovi

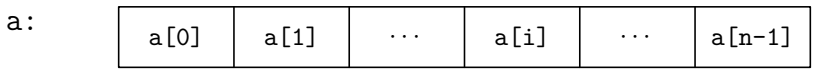
K. F. & V. B.

U rješavanju mnogih netrivijalnih problema pojavljuju se pojmovi koji se ne mogu preslikati u osnovne, „u jezik ugrađene" tipove kao što su **`int`** ili **`double`** i u operacije s njima. Jezik *C++* pruža različite mehanizme povezivanja elementarnih podataka u složen(ij)e *strukture* i mogućnost oblikovanja novih tipova koje ćemo nazivati *izvedenim tipovima*. Izvedeni su tipovi sadržani u standardnoj biblioteci i u drugim bibliotekama (namijenjenima linearnoj algebri i numeričkoj analizi, statističkoj obradi podataka, grafici, ...), a možete ih i sami oblikovati. Programsku realizaciju izvedenoga tipa nazivamo *razredom*.

Najjednostavnija je struktura podataka niz. *Niz* je uređeni skup koji sadrži određeni broj elemenata istoga tipa `T`. Formalno matematički, niz je funkcija koja svakom elementu konačnoga uređenog skupa indeksa $\mathcal{I}$ pridružuje element skupa $\mathcal{T}$:

$$a : \mathcal{I} \to \mathcal{T}.$$

S nizom se može baratati kao sa cjelinom, ali se također može i pristupati pojedinim njegovim elementima. Nekom se elementu niza pristupa navođenjem njegovoga indeksa; pritom vrijeme pristupa ne ovisi o vrijednosti indeksa. U jezicima *C* i *C++* za pristup i–tom elementu niza `a` pišemo `a[i]`. Važno je upamtiti da prvi element niza koji sadrži n elemenata ima indeks 0, a posljednji indeks $n-1$. (U jezicima *Pascal* i *Fortran* možete granice indeksa zadati po volji. Jezici *C* i *C++* nisu tako fleksibilni — granice indeksâ uvijek su 0 i $(n-1)$.) Elementi niza se u memoriji računala pohranjuju u n uzastopno smještenih varijabli određenoga tipa `T`:



Iako u *C++*-u postoje „u jezik ugrađeni" nizovi, u našim ćemo programima za prikaz nizova upotrebljavati znatno prilagodljiviji standardni tipski predložak **`vector`**<>. *Tipski predložak* je (meta)tip parametriziran tipom (ili tipovima). U slučaju **`vector`**a, parametar je tip elemenata koje niz sadrži;[1] tako je **`vector`**<**`double`**> tip kojim se prikazuju nizovi elemenata tipa **`double`**, dok je **`vector`**<**`int`**> tip kojim se prikazuju nizovi elemenata tipa **`int`**. Kao što vidimo, konkretni ćemo tip iz tipskoga predloška oblikovati tako da iza naziva predloška između znakova < i > kao argument navedemo naziv odgovarajućega tipa. I taj argument može biti tip koji je nastao iz predloška: tipom **`vector`**<**`complex`**<**`double`**>> prikazujemo nizove čiji su elementi tipa **`complex`** s realnim i imaginarnim dijelovima tipa **`double`**.

(Unatoč nazivu, predložak **`vector`**<> nema baš mnogo veze s linearnoalgebarskim vektorima. Programskom prikazu tih vektora namijenjen je standardni predložak **`valarray`**<> za koji su definirane i osnovne operacije vektorske algebre.)

Da bi se predložak **`vector`**<> mogao upotrijebiti u programu, u datoteku s izvornim kodom treba uključiti standardno zaglavlje vector.

Varijable za prikaz nizova definiramo tako da iza njihova naziva između oblih zagrada navedemo broj elemenata koje sadrže:

```
vector<double> vd(10);
vector<complex<double>> vc(n);
```

gdje je `n` varijabla cjelobrojnoga tipa.

[1] Predložak **`vector`**<> ima, u stvari, dva tipska parametra. No, drugi je parametar prilično duga priča koju, na sreću, zasad ne moramo ispričati — ako se pripadni argument ne navede, prevodilac će upotrijebiti prešutno podrazumijevani.

U znanstvenim i tehničkim primjenama često se pojavljuju velike količine podataka (poput rezultata raznih mjerenja) koje treba srediti i klasificirati. Najčešće su to nizovi podataka istog tipa (realni ili cijeli brojevi), pa su nizovi prikladne strukture za njihov prikaz.

Neka se bitna svojstva niza statističkih podataka mogu izraziti pomoću nekoliko *statističkih parametara*: za dani se niz podataka $\{x_i\}_{i=0}^{n-1}$ *aritmetička sredina* ili *prosjek* definira izrazom

$$\bar{x} = \frac{1}{n}\sum_{i=0}^{n-1} x_i\,,$$

varijanca ili *disperzija* izrazom

$$s = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^2,$$

a *standardno odstupanje* ili *standardna devijacija* izrazom

$$\sigma = \sqrt{s}.$$

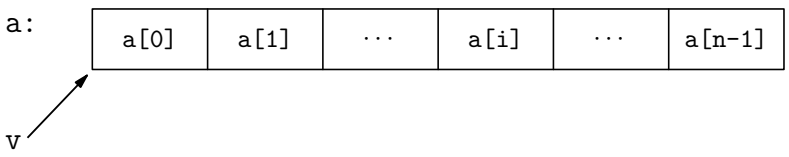
Za izračunavanje aritmetičke sredine napisali smo funkciju koju ćemo nazvati `mean()`:

```
double mean (std::vector<double> const& v) {
  int n = v.size();
  double sum = 0.0;
  for (int i = 0; i < n; ++i)
    sum += v[i];
  return sum / n;
}
```

Ponajprije treba objasniti pomalo kriptičnu definiciju parametra funkcije.

Znamo da je parametar funkcije nova varijabla koja se upotrebljava u njezinu tijelu. Navedemo li pri pozivu funkcije kao argument naziv neke varijable, njezina se vrijednost pridružuje parametru — funkcija, prema tome, barata s kopijom „vanjske" varijable. Takvo se prenošenje argumenta u funkciju naziva *prenošenjem vrijednosti*. Međutim, za nizove je, treba li u funkciji samo pročitati vrijednosti njihovih elemenata, kopiranje u novi niz nepotrebno produljenje vremena izvođenja programa i gubljenje memorijskoga prostora. Stoga niz u funkciju prenosimo *upućivanjem*, što znači da se, u stvari, prenosi samo njegova adresa.

Sintaktički oblik poziva funkcije uz prenošenje upućivanjem ne razlikuje se od poziva uz prenošenje vrijednosti: u oba ćemo slučaja pisati `mean(a)`. No, u definiciji (i u deklaraciji) funkcije treba naznačiti da je riječ o prenošenju upućivanjem — iza naziva tipa parametra stavlja se znak `&`. Nakon poziva `mean(a)` naziv `v` će se unutar funkcije odnositi na „vanjski" niz `a`:



Ključna pak riječ **`const`** prije znaka `&` naglašava da se u funkciji vrijednosti elemenata niza samo čitaju — da je niz unutar funkcije nepromjenjiv.

Petlja **for** u tijelu funkcije prolazi svim vrijednostima indeksa od 0 do $n-1$, gdje je n broj elemenata niza. Svaka varijabla tipa **vector<double>** (i, općenitije, tipa **vector<T>** za neki tip T) „zna" koliko elemenata sadrži. Taj nam broj daje funkcija *size()* koja „pripada" tipskom predlošku **vector<>**. Funkcije koje pripadaju tipovima ili tipskim predlošcima pozivamo tako da iza naziva varijable napišemo točku i naziv funkcije.

U i–tom prolazu kroz petlju varijabli sum pribrajamo vrijednost i–toga elementa niza. Budući da joj uzastopce pribrajamo vrijednosti elemenata, početna vrijednost varijable sum mora biti 0,0. Iz funkcije vraćamo njezinu konačnu vrijednost (koja je jednaka $\sum_{i=0}^{n-1} v_i$) podijeljenu s brojem elemenata niza.

Funkcija variance() izračunava varijancu:

```
double variance(std::vector<double> const& v) {
  int n = v.size();
  double sum = 0.0;
  double m = mean(v);
  for (int i = 0; i < n; ++i) {
    double b = v[i] - m;
    sum += b * b;
  }
  return sum / n;
}
```

I, napokon, funkcija std_dev() izračunava standardno odstupanje:

```
double std_dev(std::vector<double> const& v) {
  return std::sqrt(variance(v));
}
```

(Podsjećamo vas da u datoteku s ovim definicijama morate uključiti i standardnu zaglavnu datoteku cmath u kojoj je deklarirana funkcija *sqrt()*.)

Na kraju priče možemo napisati i probni programčić u kojem ćemo pozvati naše funkcije:

```
int main() {
  int n;
  std::cout << "broj podataka -> ";
  std::cin >> n;

  std::vector<double> v(n);
  for (int i = 0; i < n; ++i) {
    std::cout << "v[" << i << "] -> ";
    std::cin >> v[i];
  }

  std::cout << "aritm. sred.: " << mean(v) << std::endl;
  std::cout << "varijanca:    " << variance(v) << std::endl;
  std::cout << "stand. dev.:  " << std_dev(v) << std::endl;
  return 0;
}
```

Nadamo se da dodatna objašnjenja nisu potrebna.