Seznami

Ko si želimo podatke shraniti tako, da jih bomo lahko spreminjali in na koncu nekaj z njimi naredili (npr. da z njimi računamo, jih izpišemo, itd.), za to uporabimo spremenljivke. Vsaka spremenljivka hrani en podatek – eno številko, en znak, ..., razen nizov, ki lahko hranijo več zaporednih znakov. Nizi so posebni na zanimiv način. Med pisanjem programa nismo vedeli, točno kako dolg bo niz, ki ga bo uporabnik vnesel; rekli smo le, da mora biti krajši od neke največje dolžine niza, ki smo jo vnesli v program. Niz s kapaciteto 201 je tako lahko shranil do 200 znakov; namesto dvestotih spremenljivk smo vse te znake shranili v eno. Pogosto si želimo tudi števila shraniti tako, da bomo kasneje lahko dostopali do njih, ampak med pisanjem programa ne vemo točno, s koliko števili bo program moral delati. Problem rešimo s seznami.

1 Sintaksa

Osnovna sintaksa seznamov (angl. *array*) je zelo podobna sintaksi nizov. Da ustvarimo nov seznam, napišemo ime tipa (`int`, `long long`, itd.), ime spremenljivke in nato največjo možno dolžino seznama:

```
int seznam_stevil[300];
long long seznam_velikih_stevil[5000];
```

Prav tako kot pri nizih tudi do posamičnih elementov seznama dostopamo z oglatimi oklepaji:

```
// nastavi četrti (indeks 3) element na 7
seznam_stevil[3] = 7;

// izpiši element na petem mestu
printf("%lld\n", seznam_velikih_stevil[4]);
```

Opazimo da, prav tako kot pri nizih, tudi pri seznamih šteti začnemo pri 0. Spomnimo se, da smo pri nizih imeli posebni znak NULL, ki je programu sporočal, da se naš niz na tistem mestu konča. Za sezname števil takega znaka ne poznamo; ko delamo s števili, moramo drugače vedeti, kako dolg naš seznam dejansko je. Naloge so običajno narejene tako, da prvo število na vhodu pove, koliko števil (ki jih želimo shraniti v seznam) bo sledilo. V večini primerov je to ravno dolžina našega seznama, če pa delamo kakšne posebne trike, pa moramo za dolžino skrbeti sami.

Primer

Poglejmo si primer preproste naloge: Na vhodu je podano število N, ki mu sledi N števil. Program naj izpiše vsoto teh N-tih števil. Velja $N \leq 10^5$.

```
#include <stdio.h>

// seznam je malce daljši od 10^5 - glej spodaj
int seznam[100002];

int main() {
    int N;  // dolžina seznama
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        // preberemo številko na i-to mesto seznama
        scanf("%d", &seznam[i]);
    }
    // v spremenljivki bomo hranili delno vsoto prvih
    // nekaj mest seznama
    int vsota = 0;
    for (int i = 0; i < N; i++) {
        vsota += seznam[i];
    }
    printf("%d\n", vsota);
    return 0;
}
```

Primer vhoda in izhoda

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

```
15
```

Pri programu opazimo nekaj posebnosti. Seznam števil smo postavili *zunaj* funkcije `main`. To je zmeraj dobro narediti, če uporabljamo zelo dolge sezname, kakor zgornji seznam je. Poleg tega smo največjo dolžino seznama nastavili na $10^5 + 2$. Lahko bi nastavili dolžino na točno 10^5, vendar je dobra ideja, da si pustimo malce praznega prostora, če se kje v programu zmotimo pri indeksiranju. Ta prazen prostor nam lahko v nekaterih primerih prepreči, da se program sesuje.

Primer

Poglejmo si malo bolj zanimiv primer. Naša naloga sedaj je, da uredimo seznam N števil ($N \leq 10^6$) po velikosti od najmanjšega do največjega. Podano imamo tudi informacijo, da bodo ta števila velika med vključno 0 in 100. Naloge se lotimo tako, da preštejemo, kolikokrat se neko število pojavi v danem seznamu, nato pa bomo seznam rekonstruirali tako, da bo urejen. Da preštejemo, kolikokrat se kakšno število pojavi, uporabimo nov seznam, kjer indeks pomeni številko, ki jo štejemo, shranjena vrednost pa kolikokrat smo to številko že prešteli.

```c
#include <stdio.h>

// seznam iz vhoda
int seznam[1000003];
// na indeksu j je zapisano, kolikokrat smo že videli
// število j v seznamu
int stetje[101];
// nov, urejen seznam
int novseznam[1000003];

int main() {
    int N;  // velikost seznama
    scanf("%d", &N);
    for (int i = 0; i < N; i++)
        scanf("%d", &seznam[i]);

    // število na mestu i v seznamu je seznam[i]
    // v eni iteraciji zanke vidimo eno število; zato prištejemo
    // 1 na pravo mesto seznama za štetje
    for (int i = 0; i < N; i++)
        stetje[ seznam[i] ] += 1;

    // sedaj rekonstruiramo seznam
    // shranjujemo si indeks prvega elementa v novem seznamu,
    // ki ga še nismo nastavili
    int indeks = 0;
    for (int j = 0; j <= 100; j++) {
        // stetje[j]-krat moramo zapisati j v nov seznam
        for (int k = 0; k < stetje[j]; k++) {
            novseznam[indeks] = j;
            // povečamo indeks, ker smo ga ravno nastavili
            indeks++;
        }
    }
    // izpišemo nov seznam
    for (int i = 0; i < N; i++) {
        printf("%d ", novseznam[i]);
    }
    printf("\n");
    return 0;
}
```

Ta algoritem za urejanje je zelo znan; imenuje se urejanje s preštevanjem (angl. *counting sort*). Primeren je, kadar imamo zelo majhen razpon možnih vrednosti števil, kakor smo imeli tu ($0 - 100$).

2 Večdimenzionalni seznami

Videli smo, kako ustvariti seznam števil, kaj pa seznam seznamov? Takemu seznamu pravimo *dvodimenzionalen seznam*, ustvarimo pa ga tako, da napišemo dva zaporedna oglata oklepaja z velikostjo;

```
int seznam_seznamov_stevil[velikost1][velikost2];
```

Dvodimenzionalni seznami so uporabni, kadar moramo podatke predstaviti v tabeli. Do posamičnih elementov dostopamo z dvojnimi oglatimi oklepaji, tako kot pri inicializaciji spremenljivke; `tabela[i][j]`. Tabele si običajno predstavljamo tako, da nam prvi indeks pove zaporedno številko vrstice, drugi pa zaporedno številko stolpca.
Druga možna uporaba dvodimenzionalnih seznamov je ustvarjanje seznamov nizov. Sezname smo obravnavali kakor nekaj sorodnega nizom, dejansko pa je pravilna smer razmišljanja tu ravno obratna; nizi so pravzaprav le seznami znakov. Če želimo narediti seznam nizov, ustvarimo dvodimenzionalni seznam znakov, v katerem nam bo prvi indeks predstavljal indeks niza, drugi indeks seznama pa nam bo predstavljal indeks znaka v nizu.