Zapiski s priprav

Urejanje

ACM priprave na tekmovanja

Različica z dne 5. december 2024

Kazalo

V programih pogosto želimo nek seznam števil urediti po vrsti. V ta namen lahko napišemo svojo funkcijo, ki implementira enega od znanih algoritmov za urejanje; npr. *bubble sort*, *insertion sort*, *quick sort*, ipd. Ker pa so učinkovite implementacije pogosto kompliciranе in se pri pisanju hitro zmotimo, je bolje, da uporabimo funkcije, ravno v ta namen vključene v standardno knjižnjico. Za to bomo potrebovali na začetek programa dodati še dve vrstici:

```
#include <algorithm>
using namespace std;
```

Ukaz `#include` že poznamo, opazimo pa, da tokrat za spremembo nima končnice `.h`. To je zato, ker funkcije, ki smo jih uporabljali do sedaj, izvirajo iz jezika C, tokrat pa potrebujemo funkcijo, napisano posebaj za C++. To razloži tudi drugo vrstico; vse funkcije v standardni knjižnjici v C++ so vključene v imenski prostor `std`. Če jih želimo klicati, moramo pred ime funkcije vedno napisati `std::`, ali pa na začetek programa vključiti vrstico `using namespace std`.

Sedaj lahko uporabimo funkcijo `sort`, ki sprejme dva argumenta; kazalec na začetek in tik za konec predela spomina, ki ga želimo urediti. Poglejmo si enostavni primer.

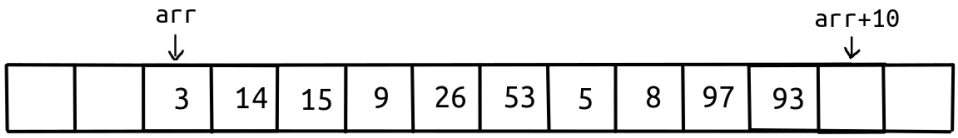
```
#include <algorithm>
#include <stdio.h>
using namespace std;

int arr[1000003];

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    sort(arr, arr+n);
    for (int i = 0; i < n; i++) {
        printf("%d\n", arr[i]);
    }
    return 0;
}
```

Program prebere število n, za njim pa še n števil, jih uredi naraščajoče, in jih izpiše. Funkcijo `sort` smo poklicali tako, da smo kot prvi argument podali seznam `arr` (oz. kazalec na prvi element seznama), kot drugi argument pa kazalec na prvo mesto

v spominu, ki ne spada več v naš seznam (oz. prvo mesto v spominu, ki ga ne želimo urediti). Na primer, če želimo urediti seznam `arr` z 10 elementi, moramo podati kazalca `arr` in `arr+10`, kot je prikazano spodaj:



Optimalna časovna zahtevnost algoritma za urejanje je $O(n \log n)$. To v praksi pomeni, da bo urejanje delovalo dovolj hitro za $n \leq 10^6$ podatkov. Če imamo več podatkov kot toliko, bo urejanje trajalo predolgo in naša rešitev ne bo sprejeta.

1 Primerjalna funkcija

Če želimo urediti seznam padajoče namesto naraščajoče, lahko seznam prvo uredimo naraščajoče, in ga nato obrnemo. Ker s tem dobimo veliko dodatnega dela, je bolje, da funkciji `sort` podamo lastno primerjalno funkcijo. Le-ta mora sprejeti dva argumenta ter vrniti `bool`, in sicer; če mora biti prvi argument v urejenem seznamu levo od drugega, mora funkcija vrniti `true`, sicer pa `false`. Če ne podamo tretjega argumenta, se `sort` obnaša tako, kot da bi podali naslednjo funkcijo:

```
bool compare(int a, int b) {
    return a < b;
}
```

Če želimo urediti seznam padajoče, moramo torej le podati nasprotno funkcijo, kot spodaj:

```
#include <algorithm>
#include <stdio.h>
using namespace std;

int arr[1000003];

int compare_padajoce(int a, int b) {
    return a > b;
}

int main() {
```

```
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    sort(arr, arr+n, compare_padajoce);
    for (int i = 0; i < n; i++) {
        printf("%d\n", arr[i]);
    }
    return 0;
}
```

2 Urejanje sestavljenih podatkov

Recimo, da imamo v nalogi dana imena tekmovalcev ter točke, ki so jih ti tekmovalci dosegli na tekmovanju, naš cilj pa je, da izpišemo imena tekmovalcev po vrsti glede na doseženo število točk. Če bomo prebrali točke in imena v različna seznama, ter uredili seznam točk, bo seznam imen ostal nespremenjen in ne bomo več vedeli, katero ime pripada katerim točkam.

Kako uredimo oba seznama hkrati? Najbolj enostavna možnost je uporaba `struct`, ki pa ga še ne poznamo. Namesto tega si lahko pripravimo seznam indeksov, ki na začetku na i-tem mestu hrani številko i. Potem sestavimo funkcijo `compare` tako, da sprejme dva indeksa, ter ju uredi glede na vrednosti v tabeli s točkami na pripadajočih indeksih. Urejamo pa ne tabele s točkami, temveč novo tabelo indeksov. Na ta način se tabeli s točkami in z imeni ne bosta spreminjali, in bodo točke pripadale imenu na istem indeksu.

Primer implementacije opisane rešitve je prikazan spodaj.

```
#include <algorithm>
#include <stdio.h>
using namespace std;

int tocke[100003];
char imena[100003][30];
int idxs[100003];

int compare(int i, int j) {
    return tocke[i] > tocke[j];
}

int main() {
```

```
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%s%d", imena[i], &tocke[i]);
    }
    // pripravimo tabelo indeksov
    for (int i = 0; i < n; i++)
        idxs[i] = i;

    sort(idxs, idxs+n, compare);

    for (int i = 0; i < n; i++) {
        int idx = idxs[i];
        printf("%s\n", imena[idx]);
    }
    return 0;
}
```

Če programu podamo spodnji vhod,

```
5
France 37
Gregor 34
Julija 38
Matija 29
Urska 8
```

nam bo izpisal imena, urejena padajoče po številu točk:

```
Julija
France
Gregor
Matija
Urska
```