Drużyna, w szeregu zbiórka!

#sortowanie #sort
#STL #wskaźnik

Jeśli do posortowania mamy większą ilość elementów, wtedy warto skorzystać z jakiegoś gotowego algorytmu sortującego – wszakże to zbyt często spotykana operacja, aby nie było w systemie gotowego narzędzia. Wykorzystanie gotowej procedury sortującej ma tę zaletę, że jest ona zoptymalizowana, przetestowana i na pewno działa poprawnie – oczywiście jeśli wykorzystujemy ją we właściwy sposób.

W zadaniach algorytmicznych często spotykamy dane wejściowe podane w następującej formie: najpierw podawana jest ilość danych – jest to liczba naturalna, dajmy na to N, a następnie podane jest N liczb (lub danych innego typu), które stanowią właściwe dane do przetworzenia.

Nasz program powinien działać poprawnie w zasadzie dla dowolnej rozsądnej wartości N. Zwykle z góry wiadomo, jakiego zakresu N należy się spodziewać – jest to podawane w treści zadania. Na przykład my przyjmiemy, że $1 \leqslant N \leqslant 1000$, czyli maksymalnie spodziewać się będziemy 1000 danych, dla ustalenia uwagi: liczb całkowitych (`int`).

Takiej ilości liczb nie wtłoczymy w zwykłe zmienne, jak używane w poprzednim przykładzie a, b oraz c. Trzeba użyć struktury danych, gdzie poszczególne elementy (komórki) mają swoje numery – przerabialiśmy to już w przypadku `string`-u, nieprawdaż? Prawdaż.

Potrzebujemy struktury o nazwie *tablica*, która już pojawiła się w podrozdziale *Kot Leonarda*.

Poniższa instrukcja to deklaracja 1000-elementowej tablicy A, której elementami będą liczby całkowite:

```
int A[1000];
```

Kolejne jej elementy to $A[0]$, $A[1]$, …, $A[999]$.* Elementy o innych numerach nie są dostępne, a próba odwołania się na przykład do elementu $A[2000]$ skończy się zapewne błędem wykonania programu. *Notabene*, systemy operacyjne Windows i Linux mogą zareagować w odmienny sposób, takie ich prawo.

Nasz program musimy rozpocząć od wczytania wszystkich danych: ilości (N) oraz wartości poszczególnych elementów tablicy A:

```
int N, A[1000];
cin >> N;
for(int i = 0; i < N; i++)
  cin >> A[i];
```

*Wynika stąd, między innymi, że jeśli potrzebowalibyśmy elementów numerowanych od 1 do 1000, to opłacałoby się zadeklarować tablicę A o jeden element większą, wtedy będziemy mieć do dyspozycji element $A[1000]$, a elementu $A[0]$ możemy po prostu nie używać.

Jak widać wykorzystamy tylko początkową część tablicy: komórki o numerach od 0 do $N-1$, ale jesteśmy przygotowani także na maksymalny rozmiar danych ($N = 1000$).

Teraz czas na uruchomienie procedury sortującej. Jest to funkcja biblioteczna, zdefiniowana w powszechnie używanej bibliotece STL.[†] Nazwa funkcji nie zaskakuje – `sort()` – ale jej argumenty wymagają komentarza. Otóż wywołuje się ją w następujący sposób:[‡]

```
sort(odkąd, dokąd);
```

Argumenty funkcji to *wskaźniki* (ang. **pointers**), czyli adresy komórki tablicy, od której należy zacząć sortowanie (*odkąd*), oraz komórki, *przed* którą należy skończyć sortowanie (*dokąd*). Tak się składa, że w tym języku programowania zmienna tablicowa (czyli A) jest zarazem wskaźnikiem do początkowej komórki, czyli do $A[0]$. Natomiast wskaźniki do kolejnych komórek (czyli ich adresy) obliczamy dodając ich numery do A. I tak, $A+1$ to wskaźnik do $A[1]$, $A+2$ wskazuje na $A[2]$, i tak dalej. Wyrażenie $A+N-1$ jest wskaźnikiem do $A[N-1]$, czyli do ostatniej komórki, którą należy objąć sortowaniem. Zatem $A+N$ (o jedno miejsce dalej) oznacza adres komórki, przed którą należy skończyć.[§]

Zatem nasze sortowanie uruchomimy przez wydanie prostej i zwięzłej instrukcji:

```
sort(A, A + N);
```

Jednak żeby to zadziałało, potrzebny jest mały myk: na początku programu powinna znaleźć się poniższa linijka, zapewniająca dostęp do definicji funkcji `sort()` (i wielu innych użytecznych funkcji i struktur danych):

```
#include <algorithm>
```

Cały program, który wczytuje opisane dane, sortuje je i wypisuje na ekranie, wygląda tak:

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

int main()
{
  int N, A[1000];
  cin >> N;
  for(int i = 0; i < N; i++)
    cin >> A[i];
  sort(A, A + N);
  for(int i = 0; i < N; i++)
    cout << A[i] << " ";
  cout << endl;
}
```

[†]Standard Template Library – Standardowa Bilioteka Szablonów.

[‡]Czasem dodaje się jeszcze trzeci argument: informację o specjalnej funkcji stosowanej do porównywania sortowanych elementów.

[§]Nie ma tu znaczenia, czy ta komórka formalnie jest dostępna czy nie. Dla $N = 1000$ adres $A+N$ sięga *poza* zadeklarowaną tablicę, ale nie jest to problemem, bo funkcja sortująca odpowiednio operuje wskaźnikami i nie narozrabia w systemie.

Na pewno nie uszło Waszej uwadze dodanie spacji pomiędzy wypisywanymi liczbami. (Dzięki temu prezentowane są one jako oddzielne liczby, a nie długi ciąg cyfr.) Trochę może niepokoić fakt, że po ostatniej wypisywanej liczbie także pojawia się spacja, ale wszystkie znane testerki ignorują takie nadprogramowe białe znaki,[¶] o ile znajdują się one na końcu wiersza.

Po uruchomieniu programu i wpisaniu przykładowych danych (liczba 10 oraz 10 liczb):

```
10
3 1 -7 8 2 3 17 5 4 3
```

otrzymujemy posortowany ciąg liczb:

```
-7 1 2 3 3 3 4 5 8 17
```

Jak widać, powtarzające się wartości znalazły się obok siebie, co nie powinno specjalnie dziwić.

A gdybyśmy chcieli otrzymać ciąg posortowany malejąco (*pardon*, nierosnąco)? Najprościej chyba posortować to jak dotąd i przeglądać od tylca, przy użyciu poniższej pętli:

```
for(int i = N - 1; i >= 0; i--)
  cout << A[i] << " ";
```

Tablice to jedne z najstarszych struktur danych, używano ich już w latach czterdziestych ubiegłego wieku. Występują praktycznie we wszystkich językach programowania. Mają wiele zalet, choć na przykład w nowoczesnym C++ są zwykle zastępowane przez *kontenery* `vector<>` lub `array<>`.

[¶]Przykładowe białe znaki to spacja, znak tabulacji i koniec wiersza.