Lekcja 10
Sortowanie

sortowanie przez wybór – bąbelkowe – pozycyjne – własności algorytmów

Porządkowanie danych jest jedną z najczęściej wykonywanych operacji przez komputery. Po co się porządkuje dane? Po prostu w danych uporządkowanych szybciej się wyszukuje informacje! Warto więc poświęcić czas na porządkowanie, które zwykle wykonujemy jeden raz, by potem wiele razy szybko i sprawnie wśród nich wyszukiwać.

Dane do sortowania przechowujemy z reguły w tablicach. W języku C tablice przesyłane jako parametry funkcji, zawsze traktowane są jak referencje. Projektant C wymyślił tak, żeby zmienne tablicowe zajmowały mniej miejsca w pamięci, ale…! Jednym z problemów jest to, że w C nie można w prosty sposób sprawdzić jak wielka jest tablica – ile ma elementów!

Sortowanie przez wybór (naiwne)

Sortowanie przez wybór jest naturalne. Szukamy w zbiorze liczb tej najmniejszej (największej) i ustawiamy ją na początku zbioru. Czynności szukania i wstawiania powtarzamy z pominięciem już uporządkowanego elementu.

Specyfikacja
Zadanie: uporządkuj rosnąco ciąg N liczb
zapisanych w tablicy TN
Dane: tablica liczb TN
Wynik: tablica liczb TN z uporządkowanymi
liczbami
Algorytm:
1. i=0
2. znajdź minimum w ciągu liczb i..N-1
3. zamień w tablicy TN elementy TN[i] z TN[min]
4. zwiększ wartość i o 1
5. jeśli i<N-1 wróć do 2

W programie korzystamy z dwóch osobnych funkcji ZAMIANA i MINIMUM. W obu funkcjach musimy podawać prócz samej tablicy jako parametr także ile jest w niej elementów (C++ ma z ttym problem). Korzystamy także z funkcji PiszTablica, która wypisuje na ekranie elementy tablicy

```
void ZAMIANA(int &a, int &b){
  int p=a;
  a=b;
  b=p;
}

int MINIMUM(int poc,int kon,int t[]){
  int k=0;
  int mink=999999;
  for (int i=poc;i<=kon;i++)
    if (t[i]<mink){
      mink=t[i];
      k=i;
    }
  return k;
}

void PiszTablica(int n, int t[]){
  for (int i=0;i<n;i++){
    cout.width(4);
    cout << t[i];
  }
  cout <<endl;
}

void SortNaiwne(int n, int t[]){
  int i=0;
  do{
    int min=MINIMUM(i,n-1,t);
    ZAMIANA(t[i],t[min]);
    i=i+1;
  } while (i<n-1);
}
```

Program główny
- deklaracja tablicy i ilości elementów
- aby losowane liczby były różne należy zainicjować licznik pseudolosowy
- wylosowanie 20 liczb do tablicy
- wypisanie tablicy nieposortowanej
- sortowanie
- wypisanie tablicy posortowanej

```
const int N=20;
int tt[N];

srand(time(NULL));

for(int i = 0; i < N; i++) tt[i] = rand() % 100;
PiszTablica(N,tt);
SortNaiwne(N,tt);
PiszTablica(N,tt);
```

```
SORT NAIWNE

  26   8  87  41  36  51   7  14  49  74  15  98   2  89  11  83  40  77  38  94

   2   7   8  11  14  15  26  36  38  40  41  49  51  74  77  83  87  89  94  98
```

Sortowanie bąbelkowe

Porównujemy parami dwie sąsiednie liczby, i jeśli pierwsza jest mniejsza od drugiej (lub większa), to zamieniamy je miejscami. Jednokrotna analiza tablicy przestawia najmniejszy element (największy) na początek (koniec). Tablicę trzeba więc analizować tyle razy ile jest elementów. Aby przyspieszyć sortowanie można pomijać elementy już uporządkowane.

Algorytm:

1. ustaw element początkowy na 0 i zwiększaj go o jeden aż osiągnie wartość N-1
   2. ustaw element analizowany na 0 i zwiększaj go o 1 aż osiągnie wartość N-pocz-1
      3. porównaj elementy tablicy i i i+1 i dokonaj ich zamiany gdy t[i] > t[i+1]
      4. wróć do punktu 3
   5. wróć do punktu 2

```
void SortBuble(int n, int t[]){
  for (int pocz=0; pocz<n-1; pocz++)
    for (int i=0; i<n-pocz-1; i++)
      if (t[i] > t[i+1])
        ZAMIANA(t[i],t[i+1]);
}
for(int i = 0; i < N; i++)
       tt[i] = rand() % 100;
PiszTablica(N,tt);
SortBuble(N,tt);
PiszTablica(N,tt);
```

Biblioteka algorithm

Biblioteka **algorithm** posiada szereg gotowych algorytmów, m.in. algorytm sortowania. Funkcja SORT jest bardzo przydatna, gdyż nie musimy pisać kodu sortującego. W najprostszej postaci wystarczy podać nazwę tablicy i ilość elementów do posortowania. Nie ma też znaczenia, czy tablica zawiera liczby czy teksty.

```
#include <algorithm>
...
int tab[] = {9,8,7,0,8,6,3,1,7,2};
sort(tab, tab+10);
```

- deklaracja i inicjacja tablicy
- sortowanie 10 elementów tablicy tab[]

Sortowanie z własnym porównaniem

Standardowo funkcja SORT porządkuje liczby rosnąco. Jeśli chcemy posortować liczby w inny sposób, np. malejąco musimy przygotować odpowiednią funkcję.

```
bool porownaj(int a, int b){
       return a>b;
}
...
sort(tab, tab+10, porownaj);
```

- sprawdzamy, czy pierwszy element jest większy od drugiego

- sortujemy z trzecim parametrem

O algorytmach teoretycznie - własności algorytmów

Analiza algorytmów jest zadaniem złożonym. Jeśli piszemy jakiś algorytm, to powinien być napisany precyzyjnie i szczegółowo, aby można go było zrozumień i przekształcić na konkretny język programowania.

Najważniejsze własności algorytmów

1. **poprawność** – rozwiązuje problem zgodnie ze specyfikacją (dane i wyniki)
2. **skończoność** – gwarantuje uzyskanie wyniku w skończonym czasie
3. **złożoność** – skończony czas i określone zasoby komputera (pamięć i szybkość)
4. **efektywność** - otrzymam y rozwiązanie w możliwie najkrótszym czasie bez błędów

Zadania

1) *W zmiennej tekstowej znajduje się zestaw losowych znaków. Posortuj go.*
2) *Zapisz do pliku tekstowego 100 losowych liczb. Następnie go wczytaj, liczby z pliku posortuj i zapisz w pliku z nazwą wpisaną z klawiatury.*
3) *Zmienna zawiera tekst, np. treść tego zadania. Policz ile jest każdego ze znaków alfabetu i posortuj je malejąco. Wyniki na ekranie i w pliku tekstowym.*
4) *Wygeneruj plik tekstowy dwojkowe.txt zawierający losowe ciągi zer i jedynek. Wczytaj plikdo programy, zamień liczby dwókowe na dziesiątkowe i posortuj malejąco.*
5) *Wygeneruj plik tekstowy daty.txt zawierający losowe daty w formacie RRRR-MM-DD. Lata w przedziale 1918..2018. Wczytaj plik do programu i posortuj daty.*