SORTOWANIE

Sortowanie bąbelkowe odbywa się następująco. Przypuśćmy, że chcemy posortować wektor w kolejności rosnącej. Porównujemy najpierw dwa pierwsze elementy i jeśli są w złej kolejności, to zamieniamy. Następnie robimy to samo z drugim i trzecim elementem i tak dalej, do końca wektora. Jeżeli w takim pojedynczym przebiegu wszystkie pary były w dobrej kolejności to znaczy, że wektor jest już posortowany. Jeśli natomiast musieliśmy wykonać przynajmniej jedną zamianę, to powtarzamy całą procedurę od początku.

Napisz procedurę bubble sortującą wektor bąbelkowo w kolejności rosnącej. Deklaracja:

```
void bubble (double vec [],int n);
```

Argument	**Wejście**	**Wyjście**
vec	Wektor do posortowania	Wektor posortowany
n	Długość wektora	

Napisz program bubble, który wczytuje ze standardowego wejścia długość wektora, tworzy go w pamięci i wypełnia losowymi liczbami, wypisuje na standardowe wyjście, a następnie sortuje bąbelkowo i wypisuje wektor posortowany.

Pamiętaj, że sortowanie bąbelkowe jest najgorszym znanym algorytmem sortowania.

```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;

void bubble (double vec [],int n) {
   bool flag;
   do {
      flag=false;
      for (int i=0;i<n-1;i++) {
         if (vec [i+1]<vec [i]) {
            double tmp=vec [i];
            vec [i]=vec [i+1];
            vec [i+1]=tmp;
            flag=true; }}}
   while (flag); }
         int main () {
                  srandom (time (0));
                  int n;
```

```
        cin>>n;
        double vec [n];
        for (int i=0;i<n;i++)
        vec [i]=random ()/RAND_MAX;
    for (int i=0;i<n;i++)
        cout<<vec [i]<<endl;
        cout<<endl;
        bubble (vec,n);
    for (int i=0;i<n;i++)
        cout<<vec [i]<<endl;
return 0; }
```

Sortowanie metodą wybierania odbywa się następująco. Przypuśćmy, że chcemy posortować wektor w kolejności rosnącej. Wybieramy najpierw z całego wektora element najmniejszy i wstawiamy go na początek, tzn. zamieniamy miejscami z pierwszym. Następnie powtarzamy tę procedurą dla wektora bez pierwszego elementu itd.

Napisz procedurę selection, która sortuje wektor rosnąco metodą wybierania. Deklaracja:

```
void selection (double vec [],int n);
```

Argument	**Wejście**	**Wyjście**
Vec	Wektor do posortowania	Wektor posortowany
N	Długość wektora	

Napisz program selection, który wczytuje ze standardowego wejścia długość wektora, two-rzy go w pamięci i wypełnia losowymi liczbami, wypisuje na standardowe wyjście, a następnie sortuje metodą wybierania i wypisuje wektor posortowany.

```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;

void selection (double vec [],int n) {
  for (int i=0;i<n-1;i++) {
    int m=i;
    for (int j=i+1;j<n;j++)
      if (vec [j]<vec [m])
        m=j;
```

```
        double t=vec [i];
        vec [i]=vec [m];
        vec [m]=t; }}
int main () {
   srandom (time (0));
   int n;
   cin>>n;
   double vec [n];
   for (int i=0;i<n;i++)
      vec [i]=random ()/RAND_MAX;
   for (int i=0;i<n;i++)
      cout<<vec [i]<<endl;
   cout<<endl;
   selection (vec,n);
   for (int i=0;i<n;i++)
      cout<<vec [i]<<endl;
   return 0; }
```

Sortowanie metodą wstawiania odbywa się następująco. Przypuśćmy, że chcemy posortować wektor w kolejności rosnącej i że pierwszych i 1 elementów jest już posortowanych. Wyjmujemy element i-ty i wstawiamy go na właściwe miejsce pośród pierwszych i elementów, przesuwając wszystkie większe od niego o jeden w kierunku większych indeksów. W ten sposób posortowanych jest już pierwszych i elementów, co pozwala na powtórzenie procedury dla kolejnego elementu itd.

Napisz procedurę insertion sortującą rosnąco wektor vec o długości n metodą wstawiania.

Deklaracja:

```
void insertion (double vec [],int n);
```

Argument	**Wejście**	**Wyjście**
vec	Wektor do posortowania	Wektor posortowany
n	Długość wektora	

Napisz program insertion, który wczytuje ze standardowego wejścia długość wektora, two-rzy go w pamięci i wypełnia losowymi liczbami, wypisuje na standardowe wyjście, a następnie sortuje metodą wstawiania i wypisuje wektor posortowany.

```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;
void insertion (double vec [],int n) {
   for (int i=1;i<n;i++) {
      double tmp=vec [i];
      int j;
      for (j=i-1;j>=0;j--) {
         if (vec [j]<=tmp)
            break;
         vec [j+1]=vec [j]; }
      vec [j+1]=tmp; }}

int main () {
   srandom (time (0));
   int n;
   cin>>n;
   double vec [n];
   for (int i=0;i<n;i++)
      vec [i]=random ()/RAND_MAX;
   for (int i=0;i<n;i++)
      cout<<vec [i]<<endl;
   cout<<endl;
   insertion (vec,n);

   for (int i=0;i<n;i++)

      cout<<vec [i]<<endl;

   return 0; }
```

W języku C++ łańcuchy, czyli zmienne tekstowe, reprezentujemy zazwyczaj przy pomocy typu string. Można jednak postępować jak w języku C, gdzie używa się do tego celu zwykłych tablic znaków. Znaki są to zmienne typu char, które zajmują jeden bajt i mogę być też traktowane jako liczby całkowite. Łańcuch może być krótszy niż zawierająca go tablica. Aby zaznaczyć, gdzie łańcuch się kończy, za jego ostatnim znakiem umieszcza się bajt zerowy. Oznacza to w szczególności, że długość tablicy musi być przynajmniej o jeden większa od liczby znaków zawartego w niej łańcucha, aby w jej ostatnim elemencie zmieścił się bajt zerowy.

Napisz funkcję length znajdującą długość łańcucha zapisanego w tablicy znaków, czyli liczbę znaków przed pierwszym bajtem zerowym. Deklaracja:

```
int length (char string []);
```

Argument	**Wejście**	**Wyjście**
string	Łańcuch	Niezmienione
length		Długość łańcucha

Napisz program length znajdujący i wypisujący na standardowe wyjście długość łańcucha Vivat scientia physica!.

```
#include <iostream>
using namespace std;
int length (char string []) {
   int i;
   for (i=0;string [i];i++);
   return i; }

int main () {
   char string [30]="Vivat scientia physica!";
   cout<<length(string)<<endl; return 0; }
```

Wykonanie tego programu wypisze na standardowe wyjście liczbę 23.