29. Komputer porządkuje liczby, czyli układamy programy sortujące

NA TEJ LEKCJI:

- ułożysz programy sortujące na podstawie poznanych algorytmów;
- zweryfikujesz poprawność programów;
- poznasz działanie funkcji swap.

Wiesz już, jak przebiega sortowanie bąbelkowe i przez wstawianie. Algorytmy nie są skomplikowane i jeśli się postarasz, ułożysz na ich podstawie programy w języku C++. Jeśli będziesz mieć z tym problemy, przeczytaj ten rozdział.

29.1. Bąbelki w C++, czyli układamy program na podstawie algorytmu sortowania bąbelkowego

Algorytm sortowania bąbelkowego (rys. 29.1.) wymaga użycia pętli, ponieważ zakłada wielokrotne powtórzenia tych samych czynności dla kolejnych liczb. Robi to $n-1$ razy, gdzie n jest liczbą liczb, które mają być sortowane. W przykładowym programie wykorzystano pętlę `for`. Wewnątrz pętli działania zależą od pewnych warunków, a więc konieczne będzie użycie instrukcji warunkowej `if`.

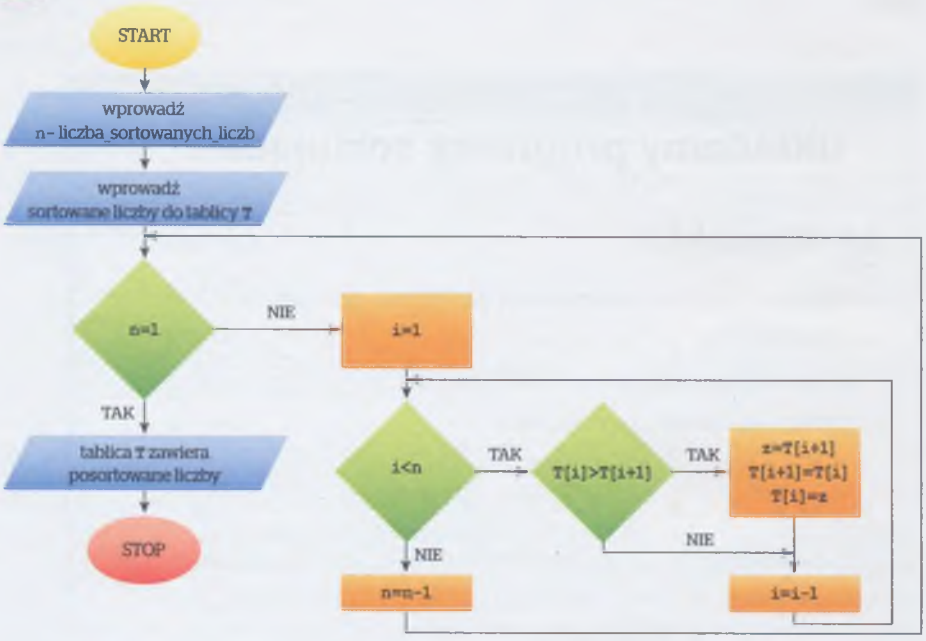
Zanim zaczniesz analizować przykładowy program, spróbuj ułożyć go samodzielnie. Nie zrażaj się początkowymi trudnościami. Przez cały czas tworzenia programu miej wgląd w algorytm i porównuj go ze swoim kodem. Wykorzystaj funkcję `swap`, która wykona zamianę liczb na wskazanych pozycjach, jeśli podasz jej odpowiednie argumenty:

funkcja swap

```
swap (T[i], T[i+1]);
```

Ta podpowiedź powinna ci pomóc w programowaniu kluczowej operacji w algorytmie sortowania bąbelkowego.

Zwróć uwagę na to, że algorytm przewiduje wprowadzenie na wstępie liczby liczb do sortowania. Ponieważ liczbę najlepiej jest umieścić w tablicy jednowymiarowej, jej deklaracja może zostać wykonana dopiero po tej czynności. Zastanów się, w którym miejscu programu ją umieścić i jak zapisać. W obu przykładach występuje podobny problem, jednak rozwiązany został w różny sposób.



Rys. 29.1. Algorytm sortowania bąbelkowego

iteracje w sortowaniu

Przejdźmy do przykładowego programu (rys. 29.2.). Zastosowano w nim konstrukcję iteracyjną.

```
1 #include<iostream>
2 #include<cstdlib>
3 using namespace std;
4
5 void sortb(int T[],int n)   //funkcja sortująca
6 {
7     for (int i=1;i<n-1;i++)
8     {
9     if (i<n and T[i]>T[i+1])
10        {
11        swap(T[i], T[i+1]);
12        i=i+1;
13        }
14    }
15 }
16
17 int main()
18 {
19     int* T, n;
20     cout<<"Podaj liczbę sortowanych liczb  ";
21     cin>>n;
22
23     T = new int [n];          //zarezerwowanie miejsca w pamięci
24                               //dla n-elementowej tablicy T
25
26     for(int i=0;i<n;i++)      //wczytanie liczb do tablicy T
27         cin>>T[i];
28
29     sortb(T,n);               //wywołanie funkcji sortującej liczby
30
31     for(int i=0;i<n;i++)
32           cout<<T[i]<<" ";    //Wyświetlenie w konsoli
33                               //posortowanej tablicy T
34
35   cout<<endl;
36   system("pause");
37   return 0;
38 }
```

Funkcja **sortb** oparta na algorytmie z rys. 29.1. Funkcja **swap** zamienia wartości dwóch zmiennych, co jest podstawą działania algorytmu sortowania bąbelkowego.

Efekt działania programu dla przykładowych liczb użytych do testowania algorytmu w rozdziale 27.2.

```
Podaj liczbę sortowanych liczb 5
3
10
2
1
4
1 2 3 4 10
Press any key to continue . . .
```

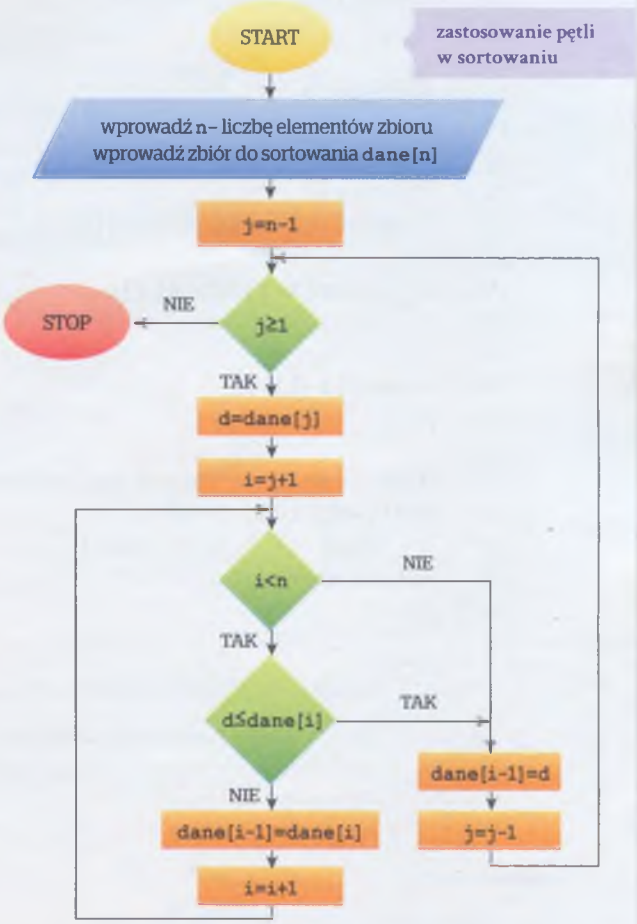
Rys. 29.2. Program sortujący liczby metodą bąbelkową

Zdefiniowano w nim funkcję `sortb`, która wykonuje część algorytmu odpowiedzialną za badanie relacji pomiędzy dwiema liczbami i ewentualną zamianę. Jest ona wywoływana *n–1* razy, a więc tyle, ile zakłada algorytm. W poniższym programie zmienna *i* występuje w funkcji i w programie głównym. Nie przechowuje ważnych danych, a jedynie odgrywa rolę pomocniczą.

W programie występuje jeszcze jeden element wymagający wytłumaczenia. Otóż, na początku program „nie wie", ile elementów będzie sortował. „Dowie się", gdy użytkownik wprowadzi tę informację, a on zapamięta tę liczbę pod nazwą zmiennej *n*. Dopiero wtedy można rezerwować w pamięci odpowiednią ilość miejsca na tablicę *n*-wymiarową. Osiągamy to w linii 23.

29.2. Wstawianie w programie, czyli układamy program sortujący przez wstawianie

Sortowanie przez wstawianie wydaje się prostym do rozwiązania problemem programistycznym. Będzie takim, jeśli dokładnie odwzorujesz algorytm (rys. 29.3.). Jak to zwykle bywa w takich przypadkach, trzeba użyć kilku znanych konstrukcji. Ponieważ działania sortujące według algorytmu należy wykonać *n–2* razy, to i pętla będzie musiała powtórzyć się tyle razy. Spróbuj swoich sił i ułóż program samodzielnie. Możesz zastosować główną pętlę z instrukcją `for` i wewnętrzne z instrukcją `while`. A może wpadniesz na inny pomysł? Pamiętaj o zweryfikowaniu działania programu dla różnych danych. Podobnie jak w przypadku sortowania bąbelkowego, jeśli napotkasz problemy, przeanalizuj przykładowy program (rys. 29.4.)



Rys. 29.3. Algorytm sortowania przez wstawianie

W przykładowym programie kluczowym elementem jest zmienna pomocnicza `d`. Przechowuje ona liczbę, która może w wyniku sortowania ulec „zamazaniu" przez liczbę przesuniętą na jej miejsce. Dzięki temu można ją ponownie umieścić w tablicy `dane [ ]` na miejscu „poniżej" liczby przesuniętej. Zwróć uwagę na operatory w pętli `while` i porównaj je z operatorami w blokach decyzyjnych algorytmu. Na pierwszy rzut oka wydają się błędne. Zwróć jednak uwagę w algorytmie na kierunek linii TAK i NIE oraz operator. Zrozumiesz, dlaczego użyto w przykładzie właśnie takiego zapisu.

Podobnie jak poprzedni program, ten także stosuje konstrukcję iteracyjną.

sortowanie wstawianie.cpp

```
#include <iostream>
#include <cstdlib>

using namespace std;
int d,i,j,n;

int main()
{
    cout << "ile liczb sortujemy?  ";
    cin>>n;
    int dane[n];
    for(int i=0;i<n;i++)
    cin>>dane[i];
    cout<<"dane przed sortowaniem"<<endl;
    for(i=0; i<n; i++)
          cout << " " << dane[i];

  for(j=n-2; j>=0; j--)          //start sortowania
  {
    d=dane[j];
    i=j+1;
    while((i<n)&&(d>dane[i]))
    {
      dane[i-1]=dane[i];
      i++;
    }
    dane[i-1]=d;
  }                               //koniec sortowania

  cout <<endl << "dane po sortowaniu"<<endl;
  for(i=0; i<n; i++)
      cout << " " << dane[i];
  return 0;
}
```

Rys. 29.4. Przykładowy program sortujący metodą „przez wstawianie"

W informatyce zazwyczaj nie ma jednego rozwiązania problemu programistycznego. Programy z rozdziału są jednymi z możliwych. Może uda ci się opracować własne rozwiązania?

ZADANIA DO ROZWIĄZANIA

1. Przeanalizuj program sortujący metodą bąbelkową (rys. 29.2.). Przedstaw działanie funkcji sortującej. Wskaż miejsca w algorytmie, które realizują poszczególne instrukcje.
2. Wykonaj symulację działania programu sortującego metodą bąbelkową (rys. 29.2.) dla 3 zmiennych [4,9,3], zapisując poszczególne działania i wartości zmiennych w odpowiednio zaprojektowanej tabeli. Wzoruj się na tabeli z rozdziałów 27. lub 28.
3. Przeanalizuj programy sortujące metodą przez wstawianie (rys. 29.4.). Przedstaw działanie głównej pętli sortowania. Wskaż miejsca w algorytmie, które realizują poszczególne instrukcje.
4. Wykonaj symulację działania programu sortującego metodą przez wstawianie (rys. 29.4.) dla 3 zmiennych [4,9,3], zapisując poszczególne działania i wartości zmiennych w odpowiednio zaprojektowanej tabeli. Wzoruj się na tabeli rozdziałów 27. lub 28.

5*. Ułóż samodzielnie program sortujący metodą bąbelkową. Omów jego działanie i swój pomysł na realizację algorytmu.

6*. Ułóż samodzielnie program sortujący metodą przez wstawianie. Omów jego działanie i swój pomysł na realizację algorytmu.

PODSUMOWANIE LEKCJI

- Funkcja `swap` wykonuje zamianę wartości zapamiętanych pod nazwami zmiennych `swap(a,b)`. s. 198
- Część główną programu (sortującą) możesz umieścić **w pętli** lub zapisać w funkcji wywoływanej odpowiednią liczbę razy. s. 198
- W programach sortowania metodami bąbelkową i przez wstawianie możesz wykorzystać pętle, ponieważ są to zazwyczaj rozwiązania **iteracyjne**, polegające na wielokrotnym powtarzaniu operacji na elementach sortowanych. s. 199