C++ - Sortowanie bąbelkowe

Ogólne

Jeśli chcemy posortować od najmniejszej do największej liczby, porównujemy najpierw dwie sąsiadujące liczby (3 i 1) i ewentualnie zamieniamy je miejscami. Na przykład:

8
12
5
1
3

W ten sposób, po kilku przebiegach, najmniejsza liczba przesunięta zostanie na samą górę (uniesie się jak bąbelek w szampanie).

Przedstawienie procedury w ujęciu poziomym (pierwszy przebieg pętli FOR):

8, 12, 5, 1, 3
8 ? 12, 5, 1, 3
8, **12 ? 5**, 1, 3
8, 5, **12 ? 1**, 3
8, 5, 1, **12 ? 3**
8, 5, 1, 3, 12

Drugi przebieg pętli FOR:

8, 5, 1, 3, 12
8 ? 5, 1, 3, 12
5, **8 ? 1**, 3, 12
5, 1, **8 ? 3**, 12
5, 1, 3, **8 ? 12**
5, 1, 3, 8, 12

Trzeci przebieg pętli FOR:

5, 1, 3, 8, 12
5 ? 1, 3, 8, 12
1, **5 ? 3**, 8, 12
1, 3, **5 ? 8**, 12
1, 3, 5, **8 ? 12**
1, 3, 5, 8, 12

Uwaga: Nigdy nie wolno kopiować kodu z PDF-a, gdyż zawiera on niewidoczne znaki końca linii i tzw. twarde odstępy. Kod należy przepisać ze zrozumieniem.

Kod - wersja podstawowa (*Windows*)

```
#include <iostream>

using namespace std;

int main() {
    bool czyKolejnyPrzebieg = true; // W języku C++ zmienne logiczne nie mają domyślnej
    // wartości.
    int tablica[6] = {99, -23, 34, 2, 121, 98};

    cout << "===== SORTOWANIE BĄBELKOWE =====" << endl;

    cout << "Zawartość tablicy nieposortowanej: ";
    for (int licznik = 0; licznik < 6; licznik++) {
        cout << tablica[licznik] << " ";
    }
    cout << endl;

    while (czyKolejnyPrzebieg) {
        czyKolejnyPrzebieg = false;
        for (int licznik = 0; licznik < 5; licznik++) {
            if (tablica[licznik] > tablica[licznik + 1]) {
                swap(tablica[licznik], tablica[licznik + 1]);
                czyKolejnyPrzebieg = true;
            }
        }
    }

    cout << "Zawartość tablicy posortowanej: ";
    for (int licznik = 0; licznik < 6; licznik++) {
        cout << tablica[licznik] << " ";
    }
    cout << endl;

    cout << "Naciśnij ENTER, aby zakończyć..." << endl;
    cin.get();
    return 0;
}
```

Kod - wersja rozbudowana (*Windows*)

```
#include <iostream>
#include <sstream> // Potrzebne do klasy istringstream
#include <regex>
#include <windows.h> // Potrzebne do aktywacji kolorów w konsoli

using namespace std;
```

```
string suroweDaneUzytkownika, oczyszczoneDaneUzytkownika;
int tablica[100];
int aktualnyRozmiarTablicy = 0; // Wartość początkowa
bool czyKolejnyPrzebieg = true; // W języku C++ zmienne logiczne nie mają domyślnej
// wartości.


void wypiszTablice() {
   // Licznik jest na początku zerowany:
   for (int licznik = 0; licznik < aktualnyRozmiarTablicy; licznik++) {
      cout << tablica[licznik] << " ";
   }
   cout << endl;
}


void sortujBabelkowo() {
   while (czyKolejnyPrzebieg) {
      // Zakładamy, że nie będzie potrzebny kolejny przebieg:
      czyKolejnyPrzebieg = false;
      // Licznik jest na początku zerowany:
      for (int licznik = 0; licznik < aktualnyRozmiarTablicy - 1; licznik++) {
         // Porównujemy każdą kolejną liczbę z jej następnikiem.
         // Jeśli jest większa niż następnik, zamieniamy je miejscami:
         if (tablica[licznik] > tablica[licznik + 1]) {
            // Funkcja "swap" zamienia swoje wartości miejscami:
            swap(tablica[licznik], tablica[licznik + 1]);
            // Skoro nastąpiła jakaś zamiana, to jednak jest potrzebny kolejny przebieg:
            czyKolejnyPrzebieg = true;
         }
      }
   }
}

// W poniższej wersji użyto podwójnej pętli FOR bazując na fakcie, że jesteśmy w stanie
// przewidzieć jaka jest MAKSYMALNA ilość przebiegów, aby tablica liczb była uporządkowana.
// Wadą tej metody jest wykonanie wszystkich przebiegów nawet wtedy, gdy liczby były już
uporządkowane.
// void sortujBabelkowo() {
//       for (int licznik = 0; licznik < aktualnyRozmiarTablicy - 1; licznik++) {
//          // Licznik jest na początku zerowany:
//          for (int licznikB = 0; licznikB < aktualnyRozmiarTablicy - 1; licznikB++) {
//             // Porównujemy każdą kolejną liczbę z jej następnikiem.
//             // Jeśli jest większa niż następnik, zamieniamy je miejscami:
//             if (tablica[licznikB] > tablica[licznikB + 1]) {
//                // Funkcja "swap" zamienia swoje wartości miejscami:
//                swap(tablica[licznikB], tablica[licznikB + 1]);
```

```
//             }
//           }
//       }
// }



int main() {
    // ==================== KOLOROWANIE NAPISÓW ==================
    // Tworzymy tzw. uchwyt do tego, co będzie pojawiać się na konsoli (do bufora konsoli):
    HANDLE konsola = GetStdHandle(STD_OUTPUT_HANDLE);
    // Aktywujemy virtualny terminal i to, co będzie się na nim pojawiać:
    #ifndef ENABLE_VIRTUAL_TERMINAL_PROCESSING // Jeśli nie jest zdefiniowany, to:
    #define ENABLE_VIRTUAL_TERMINAL_PROCESSING 0x0004
    #endif
    // Aby powyższe działało, musimy aktywować i to, co poniżej:
    #ifndef ENABLE_PROCESSED_OUTPUT // Jeśli nie jest zdefiniowany, to:
    #define ENABLE_PROCESSED_OUTPUT 0x0001
    #endif

    // Wartość trybu (intup lub output). Słowo "dw" to skrót od "Display Window",
    // jest to jednak nazwa zmiennej, i może być inna:
    DWORD dwMode = 0;
    dwMode |= ENABLE_PROCESSED_OUTPUT | ENABLE_VIRTUAL_TERMINAL_PROCESSING;
    SetConsoleMode(konsola, dwMode);
    // ================ KOLOROWANIE NAPISÓW - KONIEC ==============

    cout << "\n\n\033[1;34;40m===== SORTOWANIE BĄBELKOWE =====\033[0m" << endl;

    while (suroweDaneUzytkownika == "") {
        cout << "\n\nPodaj ciąg liczb całkowitych oddzielonych spacją: " << endl;
        getline(cin, suroweDaneUzytkownika);
    }

    // Regex = Regular Expression
    // Regex usuwa wszelkie znaki alfabetyczne (które nie są liczbami bądź spacjami).
    // Wyrażenie "[^\d]" oznacza "nie liczba" ("not a digit"). Czyli: wszystko, co nie jest liczbą,
    // zamień na element pusty. Aby uwzględnić także liczby ujemne: [^-\d ].
    // Działanie regex można przetestować na stronie: https://regex101.com
    oczyszczoneDaneUzytkownika = regex_replace(suroweDaneUzytkownika, regex{ R"([^-\d ])" }, "");
     cout << "\n\nZawartość danych po oczyszczeniu: " << oczyszczoneDaneUzytkownika << endl;

    // Teraz czas na wydobycie liczb ze strumienia danych (z pominięciem spacji).
    // strumienDanych to obiekt klasy istringstream (jej instancja).
    // Działa podobnie do "cin", ale czyta dane z "obiektu string" zamiast z wejścia
    // standardowego.
    istringstream strumienDanych(oczyszczoneDaneUzytkownika);
```

```
    // Wpisanie liczb ze strumienia do tablicy do napotkania spacji:
    while (strumienDanych >> tablica[aktualnyRozmiarTablicy]) {
        aktualnyRozmiarTablicy++;
    }

    // W przypadku tablicy dynamicznej (wektora) byłoby to:
    // vector<int> tablica;
    // while (strumienDanych >> tablica) {
    //     tablica.push_back(tablica);
    // }

    // W celach diagnostycznych:
    // cout << "Ilość wprowadzonych liczb do tablicy: " << aktualnyRozmiarTablicy << endl;

    cout << "\n\nZawartość tablicy nieposortowanej: ";
    wypiszTablice();
    sortujBabelkowo();

    cout << "\n\nZawartość tablicy posortowanej: \033[1;32;40m";
    wypiszTablice();
    cout << "\033[0m\n" << endl;

    cout << "Naciśnij ENTER, aby zakończyć..." << endl;
    cin.get();
    return 0;
}
```

Kod - wersja rozbudowana (*Linux*)

```
#include <iostream>
#include <sstream> // Dla klasy istringstream
#include <regex>

using namespace std;

string suroweDaneUzytkownika, oczyszczoneDaneUzytkownika;
int tablica[100];
int aktualnyRozmiarTablicy = 0; // Wartość początkowa
bool czyKolejnyPrzebieg = true; // W języku C++ zmienne logiczne nie mają domyślnej wartości.


void wypiszTablice() {
    // Licznik jest na początku zerowany:
    for (int licznik = 0; licznik < aktualnyRozmiarTablicy; licznik++) {
        cout << tablica[licznik] << " ";
    }
```

```
    cout << endl;
}

void sortujBabelkowo() {
    while (czyKolejnyPrzebieg) {
        // Zakładamy, że nie będzie potrzebny kolejny przebieg:
        czyKolejnyPrzebieg = false;
        // Licznik jest na początku zerowany:
        for (int licznik = 0; licznik < aktualnyRozmiarTablicy - 1; licznik++) {
            // Porównujemy każdą kolejną liczbę z jej następnikiem.
            // Jeśli jest większa niż następnik, zamieniamy je miejscami:
            if (tablica[licznik] > tablica[licznik + 1]) {
                // Funkcja "swap" zamienia swoje wartości miejscami:
                swap(tablica[licznik], tablica[licznik + 1]);
                // Skoro nastąpiła jakaś zamiana, to jednak jest potrzebny kolejny przebieg:
                czyKolejnyPrzebieg = true;
            }
        }
    }
}
// W poniższej wersji użyto podwójnej pętli FOR bazując na fakcie, że jesteśmy w stanie
// przewidzieć jaka jest MAKSYMALNA ilość przebiegów, aby tablica liczb była uporządkowana.
// Wadą tej metody jest wykonanie wszystkich przebiegów nawet wtedy, gdy liczby były już
// uporządkowane.
// void sortujBabelkowo() {
//         for (int licznik = 0; licznik < aktualnyRozmiarTablicy - 1; licznik++) {
//             // Licznik jest na początku zerowany:
//             for (int licznikB = 0; licznikB < aktualnyRozmiarTablicy - 1; licznikB++) {
//                 // Porównujemy każdą kolejną liczbę z jej następnikiem.
//                 // Jeśli jest większa niż następnik, zamieniamy je miejscami:
//                 if (tablica[licznikB] > tablica[licznikB + 1]) {
//                     // Funkcja "swap" zamienia swoje wartości miejscami:
//                     swap(tablica[licznikB], tablica[licznikB + 1]);
//                 }
//             }
//         }
// }

int main() {
    cout << "\n\n\033[1;34;40m===== SORTOWANIE BĄBELKOWE =====\033[0m" << endl;

    while (suroweDaneUzytkownika == "") {
        cout << "\n\nPodaj ciąg liczb całkowitych oddzielonych spacją: " << endl;
        getline(cin, suroweDaneUzytkownika);
    }

    // Regex = Regular Expression
    // Regex usuwa wszelkie znaki alfabetyczne (które nie są liczbami bądź spacjami).
```

```
    // Wyrażenie "[^\d]" oznacza "nie liczba" ("not a digit"). Czyli: wszystko, co nie jest liczbą,
    // zamień na element pusty. Aby uwzględnić także liczby ujemne: [^-\d ].
    // Działanie regex można przetestować na stronie: https://regex101.com
    oczyszczoneDaneUzytkownika = regex_replace(suroweDaneUzytkownika, regex{ R"([^-\
    d ])" }, "");
    cout << "\n\nZawartość danych po oczyszczeniu: " << oczyszczoneDaneUzytkownika <<
    endl;

    // Teraz czas na wydobycie liczb ze strumienia danych (z pominięciem spacji).
    // strumienDanych to obiekt klasy istringstream (jej instancja).
    // Działa podobnie do "cin", ale czyta dane z "obiektu string" zamiast z wejścia
    // standardowego.
    istringstream strumienDanych(oczyszczoneDaneUzytkownika);

    // Wpisanie liczb ze strumienia do tablicy do napotkania spacji:
    while (strumienDanych >> tablica[aktualnyRozmiarTablicy]) {
        aktualnyRozmiarTablicy++;
    }

    // W przypadku tablicy dynamicznej (wektora) byłoby to:
    // vector<int> tablica;
    // while (strumienDanych >> tablica) {
    //     tablica.push_back(tablica);
    // }

    // W celach diagnostycznych:
    // cout << "Ilość wprowadzonych liczb do tablicy: " << aktualnyRozmiarTablicy << endl;

    cout << "\n\nZawartość tablicy nieposortowanej: ";
    wypiszTablice();
    sortujBabelkowo();

    cout << "\n\nZawartość tablicy posortowanej: \033[1;32;40m";
    wypiszTablice();
    cout << "\033[0m\n" << endl;

    cout << "Naciśnij ENTER, aby zakończyć..." << endl;
    cin.get();
    return 0;
}
```

Ostatnia aktualizacja: 6 listopada 2023.