C++ - Sortowanie przez wstawianie

Opis

Sortowanie przypomina proces pobierania karty z talii (zbiór nieuporządkowany) i wstawiania karty w odpowiednie miejsce do kart już trzymanych w ręku (zbiór uporządkowany).

Algorytm ten jest efektywny dla niewielkiej liczby elementów (do 20). Jego wydajność wzrasta, gdy elementy są wstępnie - po części - poukładane ("*nearly sorted*"). Algorytm jest najwolniejszy, gdy elementy są posortowane odwrotnie.

Złożoność czasowa: $O(n^2)$.
Złożoność pamięciowa: O(1).

Jak to działa?

1. Na początku oznaczamy drugą liczbę w tablicy (czyli tę z indeksem [1]) i porównujemy ją z poprzednią (czyli z indeksem [0]). Pierwszą liczbę w tablicy traktujemy jako jednoelementowy zbiór uporządkowany (każdy jednoelementowy zbiór jest uporządkowany). Liczby znajdujące się po tym pierwszym elemencie traktujemy jako zbiór nieuporządkowany. Dlatego właśnie będziemy zaczynać porządkowanie od liczby drugiej i stopniowo powiększać zbiór uporządkowany.

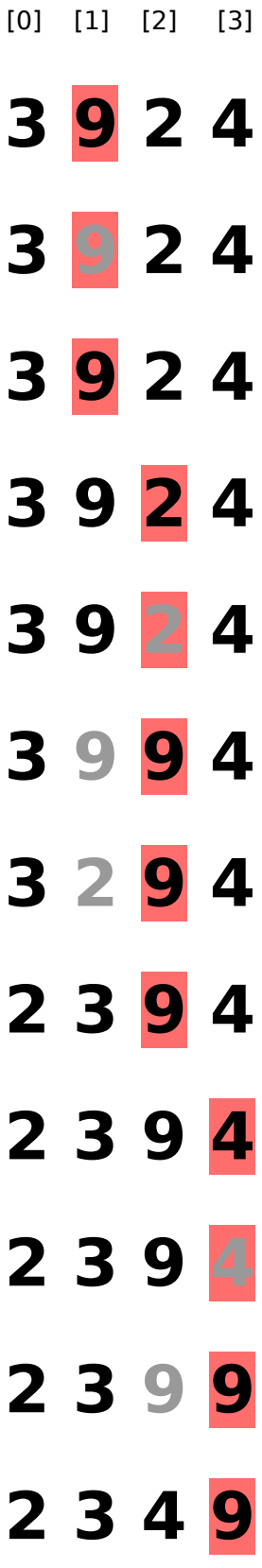
2. *WybranaLiczba* pozostawia jakby „puste miejsce” w tablicy. Na to „puste miejsce” przesunięta zostanie większa liczba lub też *wybranaLiczba* powróci na to samo miejsce (gdy to ona okaże się większą). W rzeczywistości nie jest ono puste, lecz można tak je traktować, gdyż liczba, która się tam znajdowała została zapamiętana w zmiennej, więc jej nie utracimy nadpisując to puste miejsce.

3. Jeśli pierwsza liczba (ta z indeksem [0]) okaże się większa, to przesuwamy ją na drugą pozycję (w to rzekomo „puste miejsce”).

4. Sprawdzamy następną liczbę (z indeksem [2]) w ten sam sposób... i każdą następną.

5. W przeciwieństwie do sortowania bąbelkowego, sprawdzanie liczb występuje tak długo, aż wybrana liczba od razu znajdzie się na właściwym miejscu (już nie zmieni swojego położenia w ciągu, co nie znaczy, że nie zmieni się jej indeks w tablicy).

Podsumowując: Pętla zewnętrzna **wybiera** ze zbioru kolejne elementy, pętla wewnętrzna **sortuje**, szuka dla wybranych elementów miejsca na liście uporządkowanej, porównuje „*wybranąLiczbę*” z kolejnymi liczbami w tabeli posortowanej. Gdy pętla zewnętrzna przebiegnie przez wszystkie elementy, zbiór będzie posortowany.



Uwaga: Nigdy nie wolno kopiować kodu z PDF-a, gdyż zawiera on niewidoczne znaki końca linii, twarde odstępy, odmienne cudzysłowy, odmienne apostrofy, odmienne myślniki. To godziny dodatkowej pracy na wykrywanie błędów i ich poprawianie. Kod należy przepisać ze zrozumieniem.

Kod - wersja podstawowa

```
#include <iostream>

using namespace std;

void wypiszTablice(int tablicaLiczb[], int rozmiarTablicy) {
    for (int wartoscLicznika = 0; wartoscLicznika < rozmiarTablicy; wartoscLicznika++) {
        cout << tablicaLiczb[wartoscLicznika] << " ";
    }
}

void sortowaniePrzezWstawianie(int tablicaLiczb[], int rozmiarTablicy) {
    for (int licznikGlowny = 1; licznikGlowny < rozmiarTablicy; licznikGlowny++) {
        int wybranaLiczba = tablicaLiczb[licznikGlowny];
        int licznikWewnetrzny = licznikGlowny - 1;
        while ((licznikWewnetrzny >= 0) && (tablicaLiczb[licznikWewnetrzny] > wybranaLiczba))
        {
            tablicaLiczb[licznikWewnetrzny + 1] = tablicaLiczb[licznikWewnetrzny];
            licznikWewnetrzny--;
        }
        tablicaLiczb[licznikWewnetrzny + 1] = wybranaLiczba;
    }
}


int main() {
    int tablicaLiczb[4] = {9, 7, 5, 2};
    int rozmiarTablicy = sizeof(tablicaLiczb) / sizeof(tablicaLiczb[0]); // W byte'ach

    cout << "Tablica przed sortowaniem: ";
    wypiszTablice(tablicaLiczb, rozmiarTablicy);
    cout << endl << endl;

    // Sortujemy:
    sortowaniePrzezWstawianie(tablicaLiczb, rozmiarTablicy);

    cout << "Dane posortowane przez wstawianie: ";
    wypiszTablice(tablicaLiczb, rozmiarTablicy);
    cout << endl << endl;

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

Kod - wersja rozbudowana (*Windows*)

```
#include <iostream>
#include <vector>
#include <sstream>
#include <regex>
#include <windows.h> // Potrzebne do aktywacji kolorów w konsoli systemu Windows

using namespace std;

// Deklarujemy kontener na tablicę, żeby był widoczny w poszczególnych funkcjach:
vector<int> tablicaLiczb;

// ====================== WYPISZ TABLICĘ ====================
void wypiszTablice(vector<int> tablicaLiczb) {
    // Zerujemy z poprzednich wyników:
    string tablicaJakoString = "";
    // Funkcja size() może być użyta tylko w stosunku do danych będących w wektorze:
    for (int wartoscLicznika = 0; wartoscLicznika < tablicaLiczb.size(); wartoscLicznika++) {
        // Tradycyjnie użylibyśmy tego:
        // cout << tablicaLiczb[licznik] << " ";
        // Ale tym razem zamieniamy tablicę na stringa po to,
        // aby potem wyciąć jej dwa ostatnie znaki (przecinek i spację).
        // A wszystko to po to, aby na końcu zdania ładnie postawić kropkę.
      tablicaJakoString += to_string(tablicaLiczb[wartoscLicznika]);
      tablicaJakoString += ", ";
    }
    // Wycinamy ze stringa dwa ostatnie znaki (przecinek i spację):
    tablicaJakoString.erase(tablicaJakoString.find_last_not_of(" \n\r\t")); // Znajdź ostatni znak
    // nie należący do podanego zbioru znaków
    cout << tablicaJakoString;
}

// ================ SORTOWANIE PRZEZ WSTAWIANIE ===============
// Standardowo, przekazując samą tablicę jako argument, nie przekazujemy jej rozmiaru.
// Rozmiar tablicy musimy przekazać jako osobny argument do funkcji, np.:
// int sortowaniePrzezWstawianie(int tablicaLiczb[], int aktualnyRozmiarTablicy) {...}
// No chyba że tablicę umieścimy wewnątrz konteneru "vector", a w funkcji zrobimy referencję
// do tego konteneru:
void sortowaniePrzezWstawianie(vector<int> &tablicaLiczb) {
    // Elementy większe będziemy przesuwać w prawo. Pętla FOR symuluje pobieranie kart z
    // talii.
    for (int licznikGlowny = 1; licznikGlowny < tablicaLiczb.size(); licznikGlowny++) {
        // "WybranaLiczba" to aktualnie analizowany element; liczba, która będzie wstawiana w
        // odpowiednie miejsce. Najpierw to jest liczba o pierwszym indeksie (druga od lewej) a
        // potem każda następna w tablicy.
        int wybranaLiczba = tablicaLiczb[licznikGlowny];
        // Poniżej, indeks liczby znajdujący się przed rozpatrywaną liczbą (czyli pierwsza liczba w
```

```
        // tablicy). Innymi słowy, "licznikWewnętrzny" wskazuje na liczbę, która poprzedza w
        // tablicy "wybranąLiczbę". Cofamy się o jedno miejsce w tył. Nawet jeśli w poprzedniej
        // pętli while "licznikWewnetrzny" zyskal wartość -1, to teraz ulega zresetowaniu i zyskuje
        // nową wartość opartą o licznikGłówny pętli FOR. "LicznikWewnętrzny" to inaczej liczba, z
        // którą będziemy porównywać naszą "wybranąLiczbę".
        int licznikWewnetrzny = licznikGlowny - 1; // Po kolejnych przejściach: 0, 1, 2, 3...
        // Pętla WHILE szuka miejsca dla "wybranaLiczba", przesuwa elementy większe od
        // "wybranaLiczba" o jedną pozycję w prawo, aby utworzyć miejsce dla "wybranaLiczba".
        // Będzie to robić, dopóki "licznikWewnętrzny" nie osiągnie mniejszej wartości niż 0,
        // co oznacza, że wybranaLiczba znalazła się na pierwszym miejscu, czyli na indeksie [0]
        // (w tablicy posortowanej nie ma już mniejszej liczby od niej, a tym samym nie można
        // przesunąć jej bardziej w lewo). Jeśli na liście uporządkowanej nie ma elementu
        // większego od "wybranejLiczby", to trafia ona na koniec tablicy uporządkowanej,
        // czyli tak naprawdę wraca na swoje miejsce (nadpisuje sama siebie).
        // Pętlę wewnętrzną przerywamy w dwóch przypadkach - gdy licznik pętli wyjdzie poza
        // indeks pierwszego elementu w zbiorze [-1] lub gdy "wybranaLiczba" jest większa lub
        // równa testowanemu elementowi listy uporządkowanej.
        while ((licznikWewnetrzny >= 0) && (tablicaLiczb[licznikWewnetrzny] > wybranaLiczba))
        {
            // Przesuwanie elementów większych o jedno miejsce w prawo, np.: jeśli 12 > 11, to
            // niech 12 = 11:
            tablicaLiczb[licznikWewnetrzny + 1] = tablicaLiczb[licznikWewnetrzny];
            // Poniżej, jeśli "wybranaLiczba" została umieszczona na indeksie [0],
            // to [0] - 1 = -1 (czyli while zostanie przerwane);
            licznikWewnetrzny--;
        }
        // Jeśli "wybranaLiczba" nie jest mniejsza niż jej poprzednik, to zostaje nadpisana sama
        // sobą:
        tablicaLiczb[licznikWewnetrzny + 1] = wybranaLiczba;
    }
}


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 ==============

    string suroweDaneUzytkownika, oczyszczoneDaneUzytkownika;

    cout << "\033[1;36;40m\n===== SORTOWANIE PRZEZ WSTAWIANIE =====\033[0m" <<
    endl;
    // Przykład z kartami może i dobry, ale nie przybliża nas do pomysłu jaki algorytm
    // zastosować w tym przypadku, dlatego dajemy opis:
    cout << "DEFINICJA:" << endl;
    cout << "\033[1;30;40mTablica z liczbami dzielona jest logicznie (w naszym umyśle) na
dwie tablice:\n-\033[0m \033[1;37;40mtablicę posortowaną\033[0m \033[1;30;40mskładająca
się początkowo z jednej liczby --> pierwszy element w tablicy;\n(należy uzmysłowić sobie, że
każdy zbiór składający się z jednego elementu jest zbiorem posortowanym);\n-\033[0m \
033[1;37;40mtablicę nieposortowaną\033[0m \033[1;30;40mskładającą się z pozostałych
elementów w tablicy.\n\nSortowanie zaczynamy od wybrania drugiej liczby w tablicy i
wstawienia jej w odpowiednie miejsce\nw tablicy posortowanej (tej składającej się z jednego
elementu). Mamy dwie możliwości:\n"
"- wstawimy ją przed pierwszym elementem tablicy uporządkowanej (jeśli jest mniejsza);\n"
"- lub wstawimy ją po pierwszym elemencie tablicy uporządkowanej (jeśli jest większa).\n"
"W tej drugiej sytuacji, liczba nadpisze samą siebie (czyli pozornie nic się nie zmieni), ale stanie
się ona\nw tym momencie częścią tablicy posortowanej (tablica posortowana będzie już miała
dwa elementy).\n\nW kolejnym przebiegu pętli FOR, bierzemy następną liczbę, porównujemy ją
z elementami w tablicy posortowanej\ni wstawiamy w odpowiednie miejsce. Każda kolejna
liczba w tablicy przesuwa się tak długo w lewo,\naż nie napotka liczby mniejszej od siebie lub
równej (wtedy zatrzymuje się) i przemieszczanie zaczyna\nnastępna liczba. Nadrzędna pętla
przechodzi więc po elementach tylko raz.\033[0m\n\n";

    label_wpiszLiczby:
    // Użytkownik wpisuje liczby do sortowania:
    while (suroweDaneUzytkownika == "") {
        cout << "\n\nPodaj ciąg liczb całkowitych oddzielonych spacją: ";
        getline(cin, suroweDaneUzytkownika);
    }

    // Czyścimy wpis użytkownika z niepoprawnych znaków za pomocą 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 ])" }, "");

    // Jeśli użytkownik nie wprowadził żadnej liczby, to po oczyszczeniu ciąg będzie pusty:
    if (oczyszczoneDaneUzytkownika == "") {
        cout << "\n\033[1;31;40mWprowadzone przez ciebie dane nie zawierają ani jednej
        liczby...\033[0m" << endl;
        // Resetujemy suroweDaneUzytkownika, aby ponownie uruchomić pętle while umieszczoną
```

```
        // pod labelem.
        suroweDaneUzytkownika = "";
        goto label_wpiszLiczby;
    }

    cout << "\n\033[1;30;40mZawartość danych po oczyszczeniu: " <<
    oczyszczoneDaneUzytkownika << ".\033[0m\n" << 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);
    int kolejnaLiczbaZeStrumienia;

    // Wpisywanie liczb ze strumienia do tablicy aż do napotkania spacji:
    while (strumienDanych >> kolejnaLiczbaZeStrumienia) {
        tablicaLiczb.push_back(kolejnaLiczbaZeStrumienia);
    }

    // Wypisujemy liczby nieposortowane:
    cout << "\033[1;30;40mDane po zamianie na tablicę (przed sortowaniem): ";
    wypiszTablice(tablicaLiczb);
    cout << ".\033[0m" << endl << endl;

    // Sortujemy:
    sortowaniePrzezWstawianie(tablicaLiczb);

    // Wypisujemy liczby posortowane:
    cout << "Dane posortowane przez wstawianie: \033[1;32;40m";
    wypiszTablice(tablicaLiczb);
    cout << "\033[0m." << endl << endl;

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

Kod - wersja rozbudowana (*Linux*)

```
#include <iostream>
#include <vector>
#include <sstream>
#include <regex>

using namespace std;
```

```
// Deklarujemy kontener na tablicę, żeby był widoczny w poszczególnych funkcjach:
vector<int> tablicaLiczb;

// ==================== WYPISZ TABLICĘ ====================
void wypiszTablice(vector<int> tablicaLiczb) {
    // Zerujemy z poprzednich wyników:
    string tablicaJakoString = "";
    // Funkcja size() może być użyta tylko w stosunku do danych będących w wektorze:
    for (int wartoscLicznika = 0; wartoscLicznika < tablicaLiczb.size(); wartoscLicznika++) {
        // Tradycyjnie użylibyśmy tego:
        // cout << tablicaLiczb[licznik] << " ";
        // Ale tym razem zamieniamy tablicę na stringa po to,
        // aby potem wyciąć jej dwa ostatnie znaki (przecinek i spację).
        // A wszystko to po to, aby na końcu zdania ładnie postawić kropkę.
      tablicaJakoString += to_string(tablicaLiczb[wartoscLicznika]);
      tablicaJakoString += ", ";
    }
    // Wycinamy ze stringa dwa ostatnie znaki (przecinek i spację):
    tablicaJakoString.erase(tablicaJakoString.find_last_not_of(" \n\r\t")); // Znajdź ostatni znak
    // nie należący do podanego zbioru znaków
    cout << tablicaJakoString;
}

// ================ SORTOWANIE PRZEZ WSTAWIANIE ===============
// Standardowo, przekazując samą tablicę jako argument, nie przekazujemy jej rozmiaru.
// Rozmiar tablicy musimy przekazać jako osobny argument do funkcji, np.:
// int sortowaniePrzezWstawianie(int tablicaLiczb[], int aktualnyRozmiarTablicy) {...}
// No chyba że tablicę umieścimy wewnątrz konteneru "vector", a w funkcji zrobimy referencję
// do tego konteneru:
void sortowaniePrzezWstawianie(vector<int> &tablicaLiczb) {
    // Elementy większe będziemy przesuwać w prawo. Pętla FOR symuluje pobieranie kart z
    // talii.
    for (int licznikGlowny = 1; licznikGlowny < tablicaLiczb.size(); licznikGlowny++) {
        // "WybranaLiczba" to aktualnie analizowany element; liczba, która będzie wstawiana w
        // odpowiednie miejsce. Najpierw to jest liczba o pierwszym indeksie (druga od lewej) a
        // potem każda następna w tablicy.
        int wybranaLiczba = tablicaLiczb[licznikGlowny];
        // Poniżej, indeks liczby znajdujący się przed rozpatrywaną liczbą (czyli pierwsza liczba w
        // tablicy). Innymi słowy, "licznikWewnętrzny" wskazuje na liczbę, która poprzedza w
        // tablicy "wybranąLiczbę". Cofamy się o jedno miejsce w tył. Nawet jeśli w poprzedniej
        // pętli while "licznikWewnetrzny" zyskal wartość -1, to teraz ulega zresetowaniu i zyskuje
        // nową wartość opartą o licznikGłówny pętli FOR. "LicznikWewnętrzny" to inaczej liczba, z
        // którą będziemy porównywać naszą "wybranąLiczbę".
        int licznikWewnetrzny = licznikGlowny - 1; // Po kolejnych przejściach: 0, 1, 2, 3...
        // Pętla WHILE szuka miejsca dla "wybranaLiczba", przesuwa elementy większe od
        // "wybranaLiczba" o jedną pozycję w prawo, aby utworzyć miejsce dla "wybranaLiczba".
        // Będzie to robić, dopóki "licznikWewnętrzny" nie osiągnie mniejszej wartości niż 0,
        // co oznacza, że wybranaLiczba znalazła się na pierwszym miejscu, czyli na indeksie [0]
```

```
        // (w tablicy posortowanej nie ma już mniejszej liczby od niej, a tym samym nie można
        // przesunąć jej bardziej w lewo). Jeśli na liście uporządkowanej nie ma elementu
        // większego od "wybranejLiczby", to trafia ona na koniec tablicy uporządkowanej,
        // czyli tak naprawdę wraca na swoje miejsce (nadpisuje sama siebie).
        // Pętlę wewnętrzną przerywamy w dwóch przypadkach - gdy licznik pętli wyjdzie poza
        // indeks pierwszego elementu w zbiorze [-1] lub gdy "wybranaLiczba" jest większa lub
        // równa testowanemu elementowi listy uporządkowanej.
        while ((licznikWewnetrzny >= 0) && (tablicaLiczb[licznikWewnetrzny] > wybranaLiczba))
        {
            // Przesuwanie elementów większych o jedno miejsce w prawo, np.: jeśli 12 > 11, to
            // niech 12 = 11:
            tablicaLiczb[licznikWewnetrzny + 1] = tablicaLiczb[licznikWewnetrzny];
            // Poniżej, jeśli "wybranaLiczba" została umieszczona na indeksie [0],
            // to [0] - 1 = -1 (czyli while zostanie przerwane);
            licznikWewnetrzny--;
        }
        // Jeśli "wybranaLiczba" nie jest mniejsza niż jej poprzednik, to zostaje nadpisana sama
        // sobą:
        tablicaLiczb[licznikWewnetrzny + 1] = wybranaLiczba;
    }
}


int main() {
    string suroweDaneUzytkownika, oczyszczoneDaneUzytkownika;

    cout << "\033[1;36;40m\n===== SORTOWANIE PRZEZ WSTAWIANIE =====\033[0m" <<
    endl;
    // Przykład z kartami może i dobry, ale nie przybliża nas do pomysłu jaki algorytm
    // zastosować w tym przypadku, dlatego dajemy opis:
    cout << "DEFINICJA:" << endl;
    cout << "\033[1;30;40mTablica z liczbami dzielona jest logicznie (w naszym umyśle) na
dwie tablice:\n-\033[0m \033[1;37;40mtablicę posortowaną\033[0m \033[1;30;40mskładająca
się początkowo z jednej liczby --> pierwszy element w tablicy;\n(należy uzmysłowić sobie, że
każdy zbiór składający się z jednego elementu jest zbiorem posortowanym);\n-\033[0m \
033[1;37;40mtablicę nieposortowaną\033[0m \033[1;30;40mskładającą się z pozostałych
elementów w tablicy.\n\nSortowanie zaczynamy od wybrania drugiej liczby w tablicy i
wstawienia jej w odpowiednie miejsce\nw tablicy posortowanej (tej składającej się z jednego
elementu). Mamy dwie możliwości:\n"
"- wstawimy ją przed pierwszym elementem tablicy uporządkowanej (jeśli jest mniejsza);\n"
"- lub wstawimy ją po pierwszym elemencie tablicy uporządkowanej (jeśli jest większa).\n"
"W tej drugiej sytuacji, liczba nadpisze samą siebie (czyli pozornie nic się nie zmieni), ale stanie
się ona\nw tym momencie częścią tablicy posortowanej (tablica posortowana będzie już miała
dwa elementy).\n\nW kolejnym przebiegu pętli FOR, bierzemy następną liczbę, porównujemy ją
z elementami w tablicy posortowanej\ni wstawiamy w odpowiednie miejsce. Każda kolejna
liczba w tablicy przesuwa się tak długo w lewo,\naż nie napotka liczby mniejszej od siebie lub
równej (wtedy zatrzymuje się) i przemieszczanie zaczyna\nnastępna liczba. Nadrzędna pętla
przechodzi więc po elementach tylko raz.\033[0m\n\n";
```

```
label_wpiszLiczby:
// Użytkownik wpisuje liczby do sortowania:
while (suroweDaneUzytkownika == "") {
    cout << "\n\nPodaj ciąg liczb całkowitych oddzielonych spacją: ";
    getline(cin, suroweDaneUzytkownika);
}

// Czyścimy wpis użytkownika z niepoprawnych znaków za pomocą 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 ])" }, "");

// Jeśli użytkownik nie wprowadził żadnej liczby, to po oczyszczeniu ciąg będzie pusty:
if (oczyszczoneDaneUzytkownika == "") {
    cout << "\n\033[1;31;40mWprowadzone przez ciebie dane nie zawierają ani jednej
    liczby...\033[0m" << endl;
    // Resetujemy suroweDaneUzytkownika, aby ponownie uruchomić pętle while umieszczoną
    // pod labelem.
    suroweDaneUzytkownika = "";
    goto label_wpiszLiczby;
}

cout << "\n\033[1;30;40mZawartość danych po oczyszczeniu: " <<
oczyszczoneDaneUzytkownika << ".\033[0m\n" << 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);
int kolejnaLiczbaZeStrumienia;

// Wpisywanie liczb ze strumienia do tablicy aż do napotkania spacji:
while (strumienDanych >> kolejnaLiczbaZeStrumienia) {
    tablicaLiczb.push_back(kolejnaLiczbaZeStrumienia);
}

// Wypisujemy liczby nieposortowane:
cout << "\033[1;30;40mDane po zamianie na tablicę (przed sortowaniem): ";
wypiszTablice(tablicaLiczb);
cout << ".\033[0m" << endl << endl;

// Sortujemy:
sortowaniePrzezWstawianie(tablicaLiczb);
```

```
    // Wypisujemy liczby posortowane:
    cout << "Dane posortowane przez wstawianie: \033[1;32;40m";
    wypiszTablice(tablicaLiczb);
    cout << "\033[0m." << endl << endl;

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

Ostatnia aktualizacja: 29 stycznia 2024.