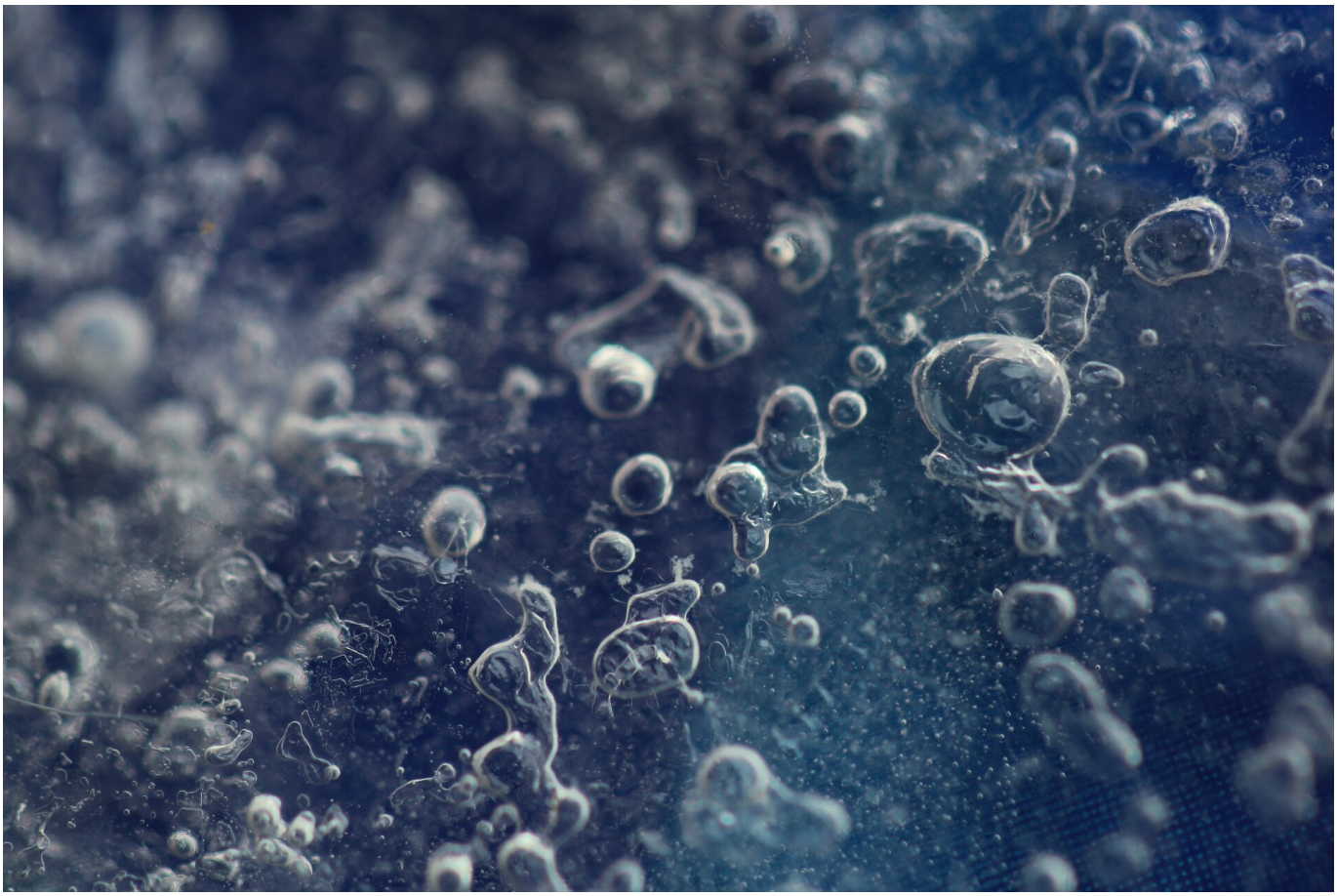
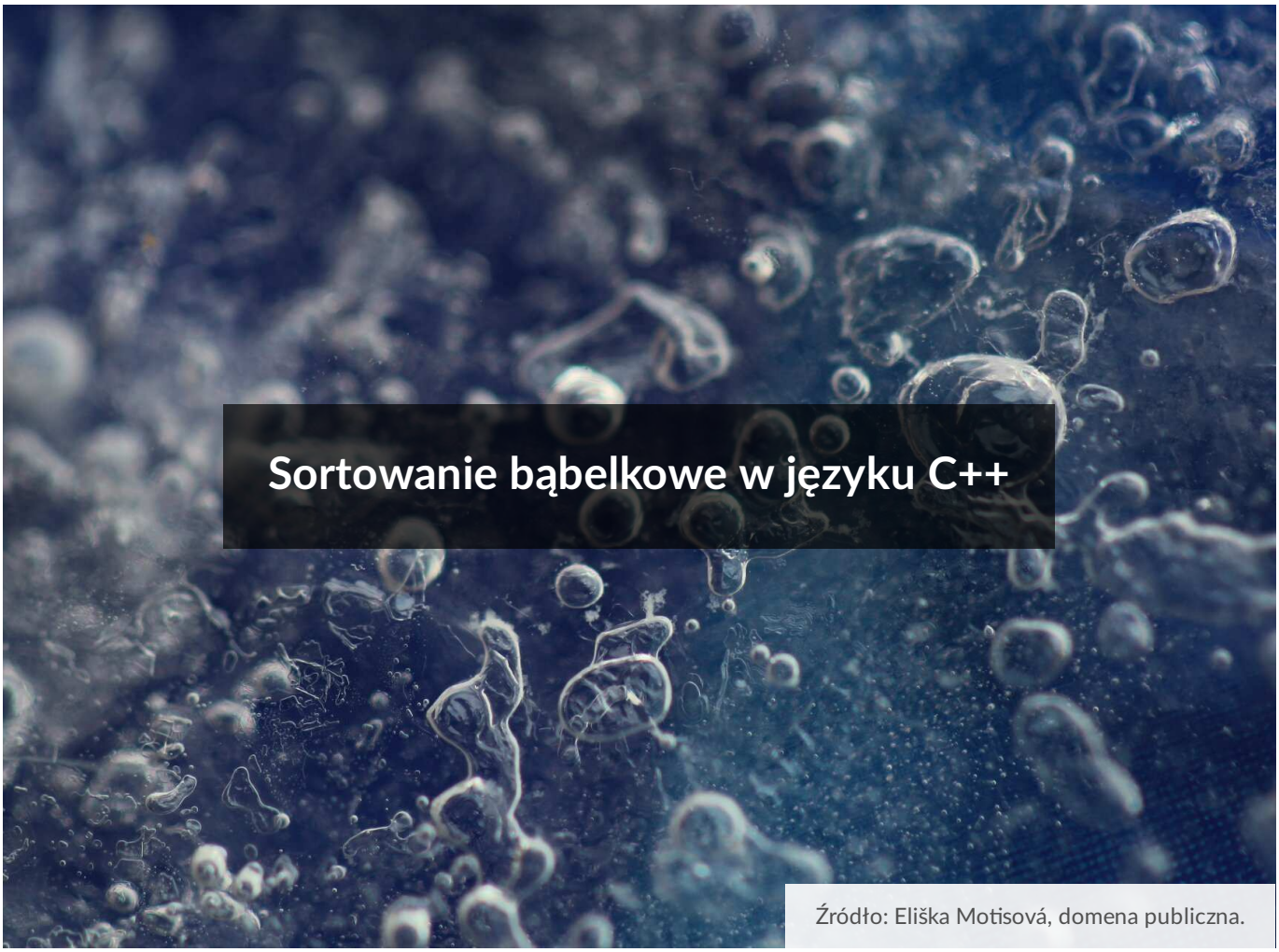


Sortowanie bąbelkowe w języku C++



Źródło: Eliška Motisová, domena publiczna.

Znasz już różne algorytmy sortowania. Jednym z najprostszych algorytmów porządkujących dane jest algorytm sortowania bąbelkowego (*bubble sort*). Podstawowe informacje na temat tego zagadnienia zostały omówione w e-materiale Sortowanie bąbelkowe. Tutaj natomiast zajmiemy się jego implementacją w języku C++.

Implementację algorytmu w innych językach programowania znajdziesz w e-materiałach:

- Sortowanie bąbelkowe w języku Python
- Sortowanie bąbelkowe w języku Java.

Więcej zadań? Przejdź do: Sortowanie bąbelkowe – zadania maturalne.

> **Twoje cele**
>
> - Prześledzisz informacje dotyczące sortowania danych.
> - Przeanalizujesz krok po kroku implementację algorytmu sortowania bąbelkowego w języku C++.
> - Zaimplementujesz program realizujący algorytm sortowania bąbelkowego.

Animacja

> **Polecenie 1**
>
> Zapoznaj się z animacją, w której omówiono sortowanie bąbelkowe w języku C++.

Film dostępny pod adresem /preview/resource/R1MTHLr42ZLGj

Film nawiązujący do treści materiału: Sortowanie bąbelkowe w języku C++.

Plik o rozmiarze 457.00 B w języku polskim

> **Problem 1**
>
> Napisz program sortujący wyniki ankiety dotyczącej problemu tortur w kontekście przestrzegania praw człowieka, wykorzystując sortowanie bąbelkowe (*bubble sort*).

Specyfikacja problemu:

Dane:

- ankietę przeprowadzono w grudniu 2008 roku przez CBOS,
- zadano pytanie: „Jaki jest pana/pani stosunek do tortur?”,

- 62% odpowiedziało: „Tortury powinny być zawsze zakazane”, 20% odpowiedziało: „Akceptacja ograniczonego stosowania tortur wobec terrorystów dla ocalenia niewinnych ludzi”, 11% nie miało zdania, 7% odpowiedziało: „Tortury powinny być ogólnie dozwolone”.

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/Rl2cVd3cybTAb

Film nawiązujący do treści materiału: Sortowanie tablicy metodą bąbelkową.

Kod programu zaprezentowanego w filmie:

Plik o rozmiarze 548.00 B w języku polskim

Przeczytaj

Przypomnijmy kilka aspektów związanych ze składnią języka C++:

- Tablicę o typie całkowitym (`Integer`) i wielkości 7 deklarujemy w następujący sposób: `int tab[7]`.
- W celu uzyskania konkretnego elementu tablicy należy użyć notacji `tab[i]`, gdzie `tab` to nazwa tablicy, a `i` to indeks elementu, do którego się odwołujemy.

> **Polecenie 1**
>
> Przeszukaj dostępne źródła i odpowiedz na pytania:

- W jaki sposób zamienia się elementy tablicy miejscami przy założeniu, że nie można wykorzystać funkcji `zamień(ciąg[i], ciąg[i+1])`?
- Czy rozwiązanie zaprezentowane w prezentacji jest w pełni optymalne?
- Jakie usprawnienia można wprowadzić, aby uniknąć niepotrzebnych iteracji i skrócić działanie algorytmu?

Przykładowa implementacja algorytmu sortowania bąbelkowego w języku C++

Przeanalizujemy przykładową implementację algorytmu sortowania bąbelkowego. Przedstawiony program posortuje niemalejąco zadany ciąg liczb całkowitych zawartych w tablicy. Implementacja będzie bazować na porównywaniu i ewentualnej zamianie kolejności elementów.

1

Prześledźmy kolejne kroki implementacji algorytmu sortowania bąbelkowego sortującego niemalejąco tablicę liczb naturalnych o rozmiarze `rozmiar`.

Program przetestujemy dla ciągu siedmiu liczb całkowitych zapisanych w tablicy:

`tablica = [ 7, 5, 3, 4, 8, 9, 1 ]`.

Specyfikacja problemu

Dane:

- `tablica` – tablica liczb całkowitych o liczbie elementów równej zmiennej `rozmiar`
- `rozmiar` – rozmiar tablicy; liczba naturalna

Wynik:

- `tablica` – tablica liczb całkowitych posortowanych niemalejąco

2

Na początku implementacji tworzymy zmienną `rozmiar` typu całkowitego, która będzie przechowywała rozmiar tablicy.

3

Następnie deklarujemy tablicę o typie całkowitym z taką liczbą elementów, jaką podamy w zmiennej `rozmiar`.

4

Teraz należy wypełnić utworzoną tablicę danymi. Można to zrobić na dwa sposoby. Pierwszym z nich jest bezpośrednie dodanie elementów przy deklaracji tablicy.

Drugim sposobem jest dodawanie elementów już po uruchomieniu programu. Umożliwia to

wielokrotne testowanie programu dla różnych danych wprowadzanych przez użytkownika.

Ponieważ mamy z góry określone dane, wykorzystamy pierwszy z wymienionych sposobów.

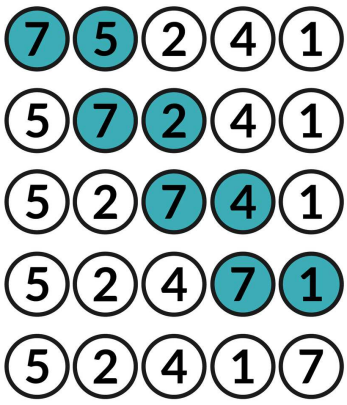
5

Kiedy już mamy gotową tablicę, możemy przejść do implementacji algorytmu sortowania bąbelkowego. Jako pierwszą zapisujemy zewnętrzną pętlę `for`. Funkcja ta będzie się wykonywać `rozmiar - 1` razy, co jest wystarczającą liczbą przejść po tablicy do posortowania jej elementów.

6

Kolejnym krokiem będzie zapisanie pętli wewnętrznej umożliwiającej przejście po elementach, czyli wyznaczenie elementów do porównywania. Ponieważ algorytm sortowania bąbelkowego bazuje na kolejnym porównywaniu i ewentualnej zamianie elementów miejscami, wyznaczenie elementów do porównania jest jedną z kluczowych operacji. W implementacji porównywać będziemy element wyznaczony przez wartość iteratora pętli wewnętrznej i element po nim następujący. W związku z tym najmniejszą dopuszczoną wartością iteratora pętli wewnętrznej będzie 0, a największą wartością będzie `rozmiar - 1`.

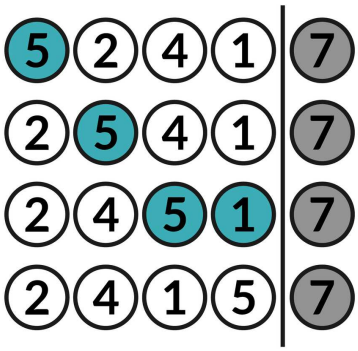
7



Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Zanim przejdziemy do kolejnego etapu, przeanalizujmy ilustrację przedstawiającą proces sortowania elementów przykładowej tablicy pięcioelementowej `[7, 5, 2, 4, 1]`. Szczególną uwagę zwróćmy na to, co stało się z największym elementem.

8



Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Największy element znalazł się na końcu tablicy. Oznacza to, że po pierwszej fazie (iteracji pętli zewnętrznej) sortowania znalazł się on na odpowiednim miejscu. Nie musimy go więc uwzględniać w kolejnym iteracjach algortmu. Zauważmy też, że kolejny największy element po zakończeniu drugiej fazy sortowania również znalazł się na odpowiednim dla siebie miejscu. Możemy zatem wysnuć wniosek, że po n-tej fazie sortowania n największych (lub najmniejszych w przypadku sortowania nierosnącego) elementów jest posortowanych.

9

Uwzględniając tę obserwację, możemy zatem dalej ograniczyć górną granicę wartości iteratora pętli wewnętrznej o wartość operatora pętli zewnętrznej. W związku z tym wartość iteratora pętli wewnętrznej ograniczona będzie z góry przez wartość `rozmiar - i - 1` (ponieważ jest to przedział prawostronnie otwarty, największą dozwoloną wartością iteratora pętli wewnętrznej będzie `rozmiar - i - 2`)

10

Przejdźmy do stworzenia mechanizmu zamiany elementów. Zaczniemy od określenia, kiedy taka zamiana ma nastąpić. Wykorzystujemy do tego celu instrukcję warunkową `if` z odpowiednim warunkiem logicznym. Skoro sortujemy elementy niemalejąco, to chcemy by były one zamienione miejscami, gdy element o mniejszym indeksie jest większy. W związku z tym w wyrażeniu logicznym instrukcji warunkowej będziemy sprawdzać, czy element wyznaczony przez wartość iteratora pętli wewnętrznej jest większy od elementu znajdującego się po nim (w przypadku sortowania nierosnącego sprawdzalibyśmy, czy wspomniany element jest mniejszy niż kolejny).

11

Gdy warunek pętli wewnętrznej zostanie spełniony, konieczna jest zamiana elementów miejscami. Użyjemy w tym celu zmiennej pomocniczej `pom`. Najpierw zapiszemy do niej

wartość przechowywaną w tablicy pod indeksem wyznaczonym przez iterator pętli wewnętrznej, żeby uniknąć jej nadpisania przez wartość, z którą ma być zamieniona. Następnie do uwolnionej w ten sposób komórki tablicy zapisujemy wartość przechowywaną w tablicy pod następnym indeksem. Na koniec do komórki przechowującej dotychczas element następujący po elemencie wyznaczonym przez iterator pętli wewnętrznej zapisujemy wartość przechowywaną w zmiennej pomocniczej.

12

Wyświetlamy zawartość posortowanej tablicy, żeby sprawdzić, czy algorytm zadziałał. Używamy w tym celu pętli `for`. Pętla iteruje po tablicy i wypisuje jej elementy:

13

Napisany dotychczas kod prezentuje się następująco:

14

Warto dodać, że w języku C++ istnieje wbudowana funkcja zamiany – `swap`, którą możemy wykorzystać. Unikamy tym samym tworzenia zmiennej pomocniczej `pom`. Jednak by móc zastosować funkcję `swap`, musimy pamiętać o dołączeniu do programu biblioteki `<algorithm>`.

Potrzebną bibliotekę dodajemy w następujący sposób:

```
#include <algorithm>
```

15

Kod działa poprawnie, jest on jednak daleki od ideału. Jesteśmy w stanie go zoptymalizować, dodając do niego kilka modyfikacji.

16

Zauważmy, że program wykonuje taką samą liczbę iteracji niezależnie od tego, czy ciąg liczb zapisany w tablicy jest już posortowany, czy nie (pod warunkiem, że są one tej samej długości). Zmieńmy zatem algorytm tak, by w przypadku otrzymania na wejściu lub uzyskaniu w trakcie działania posortowanej tablicy algorytm kończył działanie.

17

Użyjemy do tego celu zmiennej `zmieniono` typu `bool`. Jej zadaniem będzie przechowywanie informacji o wykonaniu w danej iteracji pętli zewnętrznej operacji zamiany elementów miejscami. Na początku każdej iteracji pętli zewnętrznej ustawimy jej wartość na `false`. W przypadku wystąpienia operacji zamiany elementów ustawimy jej wartość `true`.

18

Jeżeli po zakończeniu wykonywania pętli wewnętrznej wartością tej zmiennej będzie `false` oznacza to, że tablica jest już posortowana i możemy zakończyć wykonywanie algorytmu. Zrobimy to za pomocą instrukcji warunkowej `if` sprawdzającej wartość zmiennej `zmieniono` i przerywającej wykonywanie pętli instrukcją `break` w przypadku przechowywania w niej wartości `false`.

Kompletny kod algorytmu wraz z deklaracjami i funkcją `main()` wygląda następująco:

19

jego wykonanie daje następujący wynik:

Alternatywna implementacja algorytmu sortowania bąbelkowego w języku C++

Ponownie stworzymy program sortujący niemalejąco ciąg liczb całkowitych zapisanych w tablicy. Tym razem algorytm zaimplementujemy, bazując na porównywaniu wybranego elementu z elementem go poprzedzającym zamiast porównania z elementem następującym po nim.

1

Przestawiona wcześniej implementacja algorytmu sortowania bąbelkowego nie jest jedyną poprawną wersją. W tej prezentacji przeanalizujmy inny wariant programu i

omówimy różnicę między tymi dwoma rozwiązaniami.

Program przetestujemy dla ciągu siedmiu liczb całkowitych zapisanych w tablicy:

`tablica = [ 7, 5, 3, 4, 8, 9, 1 ]`.

2

Pierwszą niezbędną modyfikacją algorytmu jest zmiana indeksu komórki tablicy, w której przechowywany jest element porównywany z elementem wyznaczonym przez wartość iteratora pętli wewnętrznej z `j + 1` na `j - 1`. Porównajmy odpowiednie fragmenty obu implementacji.

Pierwsza implementacja:

Druga implementacja:

3

Konieczne jest również wprowadzenie zmian w instrukcji warunkowej, w której zawarty jest przedstawiony fragment kodu. Jednak nie wystarczy zamienić wartości `j + 1` na `j - 1`. Konieczne jest również dopasowanie wyrażenia logicznego w taki sposób, by badało, czy element wyznaczony przez wartość iteratora pętli wewnętrznej jest mniejszy niż element go poprzedzający, żeby zachować kolejność sortowania.

Pierwsza implementacja:

Druga implementacja:

4

Ostatnią konieczną modyfikacją jest dostosowanie przedziału wartości iteratora pętli wewnętrznej. Skoro porównujemy element wyznaczony przez wartość iteratora pętli wewnętrznej z elementem go poprzedzającym, to najmniejsza wartość wspomnianego iteratora musi wskazywać na drugi element tablicy, czyli przyjmować wartość 1. Z tego samego powodu jego przedział wartości musi być z góry ograniczony w taki sposób, by jego największa wartość wskazywała ostatni element tablicy. Jego wartości będą więc z góry ograniczone przez wartość `rozmiar - i` (przedział prawostronnie otwarty).

Pierwsza implementacja:

Druga implementacja:

5

Pozostałe elementy algorytmu nie wymagają zmiany.

Cały kod tej implementacji algorytmu sortowania bąbelkowego wraz z deklaracjami i

funkcją `main()` wygląda następująco:

> **Ważne!**
>
> Obie przedstawione w tej sekcji implementacje są poprawnymi implementacjami algorytmu sortowania bąbelkowego.
> **Ciekawostka**
>
> Nazwa sortowanie bąbelkowe nawiązuje do faktu, że dane z każdą iteracją są wyrzucane do góry, podobnie jak bąbelki w napojach gazowanych.

Słownik

> **implementacja**
>
> zapis abstrakcyjnego opisu algorytmu jako programu w wybranym języku programowania
> **iteracja**
>
> wielokrotne wykonywanie tych samych operacji (konkretną liczbę powtórzeń lub do spełnienia danego warunku); stosowana najczęściej w pętlach
> **zagnieżdżenie pętli**
>
> wykonywanie pętli wewnątrz innej istniejącej pętli

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Zaprezentowany kod jest implementacją algorytmu sortowania bąbelkowego. Program sortuje elementy tablicy nierosnąco. Zmodyfikuj go tak, aby sortował elementy tablicy niemalejąco.

Przetestuj swój program dla tablicy składającej się z następujących elementów: $7, 8, 5, 6, 3, 2, 1, 9, 4, 0$.

Specyfikacja problemu:

Dane:

- `liczby` – tablica liczb naturalnych
- `rozmiar` – liczba naturalna; rozmiar tablicy

Wynik:

- `liczby` – tablica liczb naturalnych posortowana niemalejąco

Twoje zadania

1. Zmodyfikuj program tak, aby sortował elementy tablicy niemalejąco.

Ćwiczenie 2

Zaprezentowany kod jest implementacją algorytmu sortowania bąbelkowego. Program do zamiany miejscami elementów tablicy wykorzystuje zmienną pomocniczą `pom`. Zaproponuj inny sposób zamiany elementów miejscami. Pamiętaj o niezbędnych bibliotekach.

Specyfikacja problemu:

Dane:

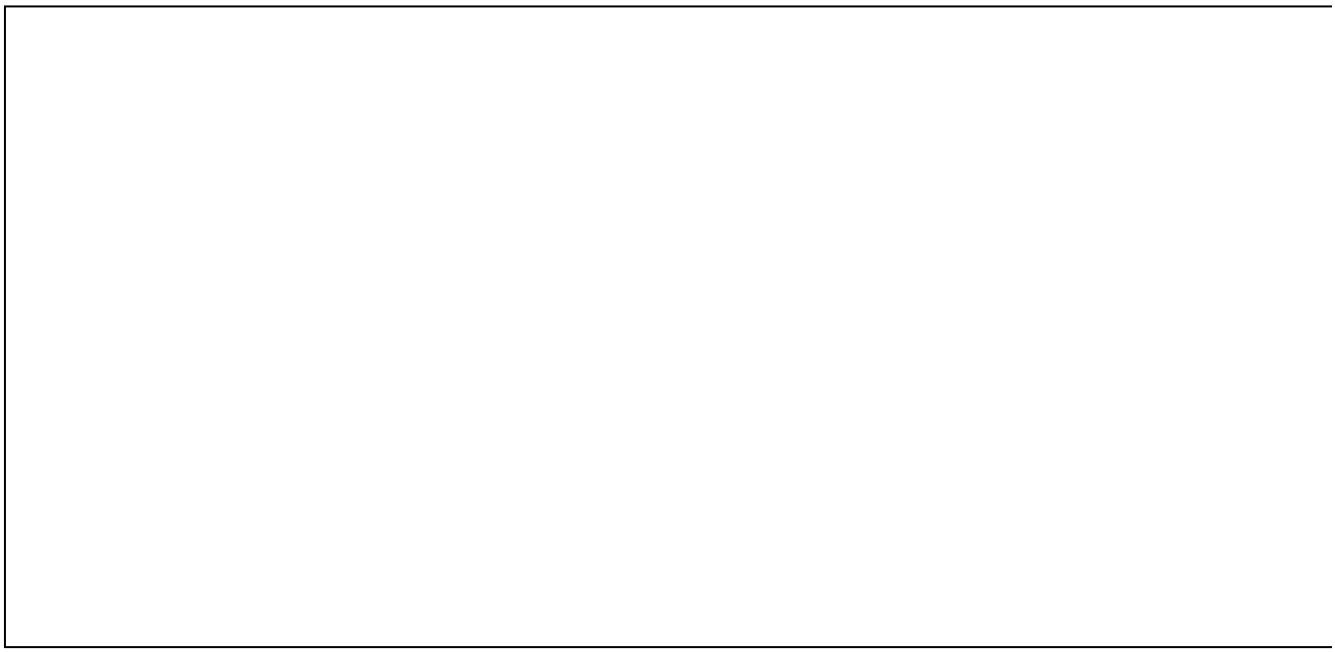
- `liczby` – tablica liczb naturalnych
- `rozmiar` – rozmiar tablicy; liczba całkowita

Wynik:

- `liczby` – tablica liczb naturalnych posortowana niemalejąco

Twoje zadania

1. Zmodyfikuj program tak, aby do zamiany elementów stosował inną metodę i sortował liczby niemalejąco.



Ćwiczenie 3

W pewnej kręgielni zapisywano najlepsze dzienne wyniki. Po tygodniu przyszedł czas wyznaczenia rankingu graczy. Użyj sortowania bąbelkowego, aby wydrukować posortowaną rosnąco listę wyników. Użyj wersji zoptymalizowanej algorytmu.

Specyfikacja problemu:

Dane:

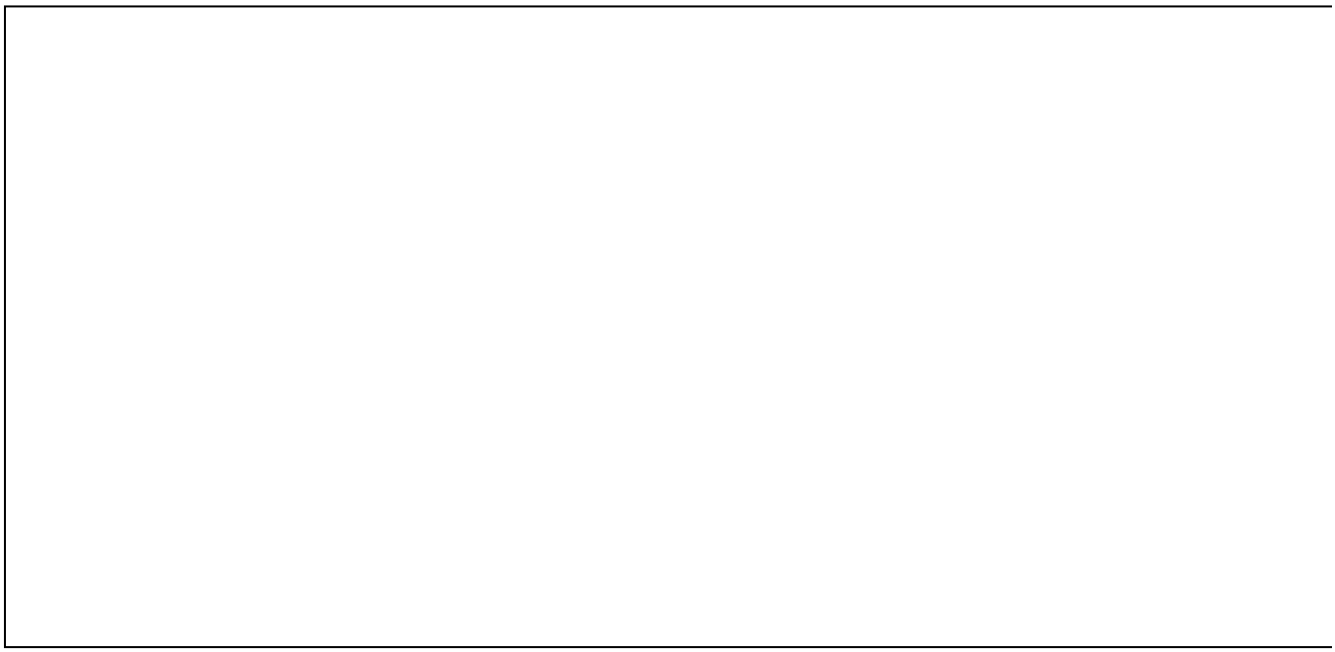
- `wyniki` – tablica liczb naturalnych
- `rozmiar` – rozmiar tablicy; liczba całkowita

Wynik:

- `wyniki` – tablica liczb naturalnych

Twoje zadania

1. Program drukuje na standardowe wyjście posortowaną tablicę `{84, 102, 77, 121, 111, 190, 138}`, każdy element w osobnej linii.



Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie bąbelkowe w języku C++

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

2) stosuje przy rozwiązywaniu problemów z różnych dziedzin algorytmy poznane w szkole podstawowej oraz algorytmy:

c) porządkowania ciągu liczb: przez wstawianie i metodą bąbelkową,

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Prześledzisz informacje dotyczące sortowania danych.

- Przeanalizujesz krok po kroku implementację algorytmu sortowania bąbelkowego w języku C++.
- Zaimplementujesz program realizujący algorytm sortowania bąbelkowego.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Sortowanie bąbelkowe w języku C++”. Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Animacja”.

Faza wstępna:

1. Nauczyciel wyświetla uczniom temat zajęć oraz cele. Prosi, by na ich podstawie uczniowie sformułowali kryteria sukcesu.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel sprawdza przygotowanie uczniów do zajęć, prosząc wybrane osoby o przedstawienie najważniejszych informacji przedstawionych w sekcji „Animacja".
2. **Praca ztekstem.** Uczniowie pracując w parach, analizują krok po kroku implementację algorytmu sortowania bąbelkowego przedstawioną w sekcji „Przeczytaj". Następnie testują omówione rozwiązania na swoich komputerach. Na forum klasy dzielą się swoimi spostrzeżeniami.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1, a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. W kolejnym etapie uczniowie dobierają się w pary i wykonują ćwiczenia nr 2 i 3. Następnie konsultują swoje rozwiązania z inną parą uczniów i ustalają jedną wersję odpowiedzi.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat i cele lekcji zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji następuje omówienie ewentualnych problemów z rozwiązaniem ćwiczeń z sekcji „Sprawdź się".

Praca domowa:

1. Uczniowie wykonują problem z sekcji „Animacja". Ich zadaniem jest napisanie programu sortującego wyniki ankiety dotyczącej problemu tortur w kontekście przestrzegania praw człowieka, wykorzystując sortowanie bąbelkowe.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Animacja", „Przeczytaj", „Sprawdź się" jako materiał do lekcji powtórkowej.