Anagramy w języku C++

Źródło: Wokandapix, domena publiczna.

Dwa słowa są anagramami, jeżeli po przestawieniu liter jednego słowa można utworzyć drugie słowo. Przykładami są ekran – nerka, alergia – galeria. Anagramy często wykorzystuje się w różnego rodzaju grach czy zabawach słownych. Jak sprawić, by program komputerowy wyszukiwał pary wyrazów i określał, czy są one anagramami?

Więcej na temat anagramów znajdziesz w e-materiale Anagramy.

Implementację w pozostałych językach programowania znajdziesz w e-materiałach:

- Anagramy w języku Java,
- Anagramy w języku Python.

Więcej zadań? Przejdź do e-materiału Anagramy – zadania maturalne.

> **Twoje cele**
>
> - Przeanalizujesz implementację algorytmu weryfikującego, czy podana para wyrazów to anagramy.
> - Napiszesz w języku C++ program sprawdzający, czy słowa są anagramami.
> - Wykonasz kilka prostych zadań związanych z anagramami.

Film samouczek

Problem 1

Napisz program, który sprawdzi, czy podane słowa są anagramami. Sprawdź słowo „ABAB” i potencjalny anagram „BBAA”.

Specyfikacja

Dane:

- `napis, jegoAnagram` – zmienne typu *string*

Wynik:

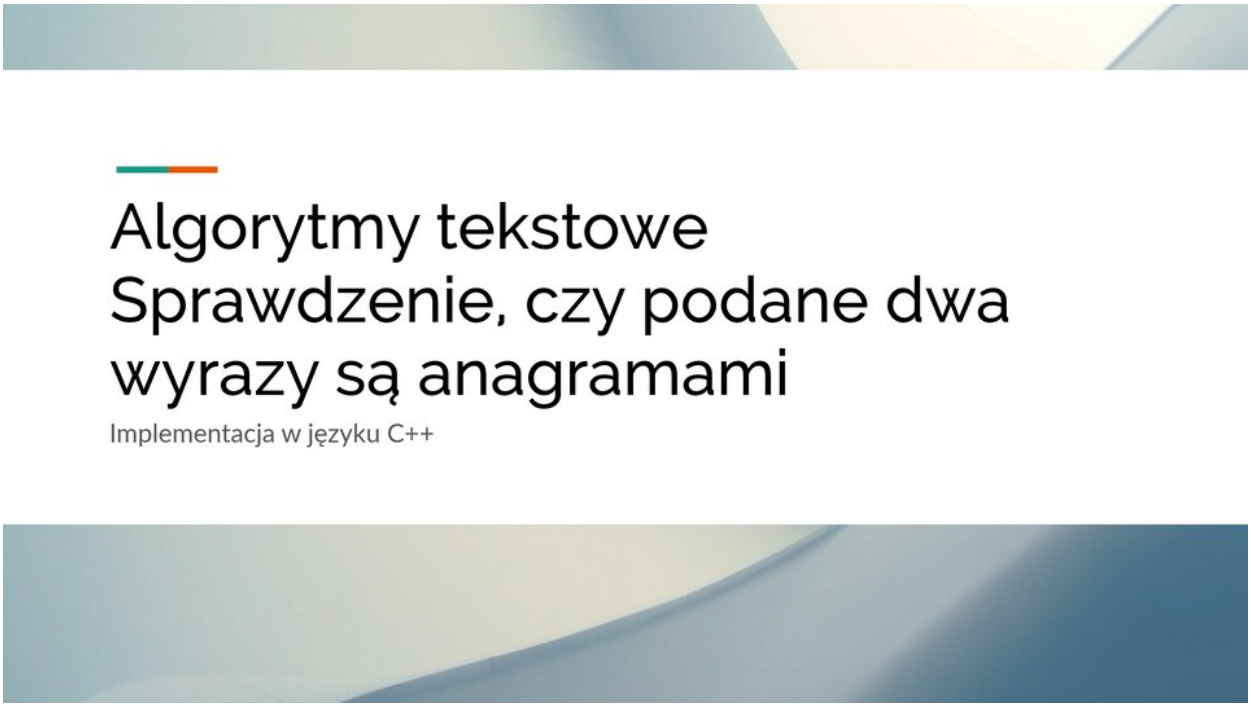
Komunikat „Słowa nie są anagramami” lub „Słowa są anagramami”.

```
1
```

```
1
```

Polecenie 1

Porównaj swoje rozwiązanie z animacją.



Film dostępny pod adresem /preview/resource/R1S5Wdb2hxN6J

Film nawiązujący do treści materiału przedstawia program zajmujący się anagramem

Polecenie 2

Przeanalizuj poniższe kroki implementacji algorytmu zapisanego w języku C++, który sprawdza, czy podaną parę słów można nazwać anagramami. Następnie zastanów się nad innymi algorytmami, które rozwiązują ten problem.

1

Twoim zadaniem jest napisanie programu sprawdzającego, czy podane dwa wyrazy spełniają omówione w poprzedniej sekcji warunki nazwania ich anagramami.

Napiszemy program, który przedstawiony został w poprzedniej sekcji, a następnie dokonamy kilku modyfikacji.

Do napisania tego programu wykorzystamy umiejętności nabyte w poprzednich lekcjach oraz informacje omówione w poprzedniej sekcji.

2

Zacznijmy od zbudowania szkieletu programu, czyli zadeklarowania klasy i głównej funkcji programu – `main`. To w niej będzie się znajdować reszta naszego programu. Na samym końcu dodamy i pokażemy potrzebne biblioteki.

```
int main() {
    return 0;
}
```

3

Następnie stworzymy szkielet funkcji `czyAnagramy()`. Jej typem będzie zmienna logiczna `bool`, która zwróci wartość `true`,

jeżeli podana para słów to anagramy, a w przeciwnym wypadku `false`.

```
bool czyAnagramy() {

}

int main() {
    return 0;
}
```

4

Nasza funkcja będzie przyjmowała na wejściu dwa parametry – `slowo1`, czyli pierwsze słowo ze sprawdzanej pary oraz `slowo2`, czyli drugie słowo z pary. Parametry będą typu `string`.

```
bool czyAnagramy(string slowo1, string slowo2) {

}

int main() {
    return 0;
}
```

Zastanów się, o czym musimy pamiętać...

5

Czy już wiesz, o co chodzi?

Musimy pamiętać o dodaniu biblioteki `<string>`. Zrobimy to na samym końcu.

6

Teraz możemy przejść do pisania pierwszego warunku. W funkcji sprawdzamy, czy podane

wyrazy są równej długości. Zwracamy `false`, jeżeli nie są.

```
bool czyAnagramy(string slowo1, string slowo2) {

    if (slowo1.length() != slowo2.length()) {
        return false;
    }
}

int main() {
    return 0;
}
```

7

Załóżmy, że nasza para słów przeszła pierwszą weryfikację, czyli podane słowa są równej długości. Teraz posortujemy oba słowa funkcją `sort()`. Pamiętajmy o dodaniu biblioteki `algorithm`

```
bool czyAnagramy(string slowo1, string slowo2) {
    if (slowo1.length() != slowo2.length()) {
        return false;
    }

    sort(slowo1.begin(), slowo1.end());
    sort(slowo2.begin(), slowo2.end());

    if (slowo1 == slowo2) {
        return true;
    } else {
        return false;
    }
}
```

```
15
16 int main() {
17     return 0;
18 }
```

8

Nasza funkcja `czyAnagramy()` jest gotowa. Teraz musimy wywołać ją w funkcji `main()`. Potrzebujemy najpierw słów wejściowych oraz zmiennej `boolanagram`, która przyjmie wartość `true` lub `false` w zależności od wyniku funkcji `czyAnagramy()`.

```
1  bool czyAnagramy(string slowo1, string slowo2) {
2      if (slowo1.length() != slowo2.length()) {
3          return false;
4      }
5
6      sort(slowo1.begin(), slowo1.end());
7      sort(slowo2.begin(), slowo2.end());
8
9      if (slowo1 == slowo2) {
10         return true;
11     } else {
12         return false;
13     }
14 }
15
16 int main() {
17     string slowo1 = "hiacynt";
18     string slowo2 = "chityna";
19     bool anagram = czyAnagramy(slowo1,
```

```
    slowo2);
20
21      return 0;
22  }
```

9

Teraz określimy komunikaty tekstowe, jakie mają się pokazać użytkownikowi w zależności od tego, czy podana para słów to anagramy czy nie.

```
1  bool czyAnagramy(string
   slowo1, string slowo2) {
2      if (slowo1.length() !=
   slowo2.length()) {
3          return false;
4      }
5
6      sort(slowo1.begin(),
   slowo1.end());
7      sort(slowo2.begin(),
   slowo2.end());
8
9      if (slowo1 == slowo2)
   {
10         return true;
11     } else {
12         return false;
13     }
14 }
15
16 int main() {
17     string slowo1 =
   "hiacynt";
18     string slowo2 =
   "chityna";
19     bool anagram =
   czyAnagramy(slowo1,
   slowo2);
20
21     if (anagram == true) {
22         cout << "Podana
   para wyrazów to anagramy";
```

```
    } else {
        cout << "Podana para wyrazów to nie są anagramy";
    }

    return 0;
}
```

10

Możemy teraz dokonać pewnej modyfikacji programu, która umożliwi użytkownikowi wpisywanie własnych wyrazów w celu sprawdzenia, czy są anagramami.

Oto wersja finalna programu.

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

bool czyAnagramy(string slowo1, string slowo2) {
    if (slowo1.length() != slowo2.length()) {
        return false;
    }

    sort(slowo1.begin(), slowo1.end());
    sort(slowo2.begin(), slowo2.end());

    if (slowo1 == slowo2) {
        return true;
    } else {
        return false;
    }
}

```

```
int main() {
    string slowo1;
    string slowo2;

    cout << "Wprowadź 1. słowo" << endl;
    cin >> slowo1;
    cout << "Wprowadź 2. słowo" << endl;
    cin >> slowo2;

    bool anagram = czyAnagramy(slowo1, slowo2);

    if (anagram == true) {
        cout << "Podana para wyrazów to anagramy";
    } else {
        cout << "Podana para wyrazów to nie są anagramy";
    }

    return 0;
}
```

Przeczytaj

Komputerowa realizacja w języku C++

Zastanówmy się, jak zaimplementować w języku C++ sprawdzenie, czy podana para wyrazów to anagramy. Zanim to nastąpi, przypomnij sobie niektóre zagadnienia:

- **funkcje** – napiszemy osobną funkcję do sprawdzania, czy podana para słów to anagramy. Przez użycie tej formy program stanie się bardziej czytelny i optymalny – będzie można sprawdzić kilka par wyrazów bez powielania kodu,
- **tablice znaków** – to w nich będziemy przechowywać nasze wyrazy. Przypomnij sobie, jak używać `string` i jak wywoływać poszczególne znaki,
- **sortowanie** – za jego pomocą posortujemy litery w danych słowach, a następnie sprawdzimy, czy są one równe. Użyjemy do tego funkcji `sort()`,
- **zmienna logiczna** – taka będzie cała nasza funkcja `czyAnagram()`. W wypadku, gdy podana para słow spełni warunki, zwróci wartość logiczną `true`, w przeciwnym `false`.

Spójrzmy na kod gotowego programu napisanego w języku C++, który w następnej sekcji właśnie będziemy tworzyć:

```
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

bool czyAnagram(string slowo1, string slowo2) {
    if (slowo1.length() != slowo2.length()) {
        return false;
    }

    sort(slowo1.begin(), slowo1.end());
    sort(slowo2.begin(), slowo2.end());

    if (slowo1 == slowo2){
        return true;
    } else {
        return false;
    }
```

```
}

int main() {
    string slowo1 = "hiacynt";
    string slowo2 = "chityna";
    bool anagram = czyAnagram(slowo1, slowo2);

    if (anagram == true) {
        cout << "Podana para wyrazów to anagram";
    } else {
        cout << "Podana para wyrazów nie jest anagramem";
    }

    return 0;
}
```

W programie zostało użytych kilka funkcji wbudowanych:

- `length()` – należy do biblioteki `<string>` i zwraca liczbę znaków danego łańcucha, czyli jego długość. W programie używamy jej przy sprawdzeniu, czy podana para słów jest równej długości,
- `begin()` – również należy do biblioteki `<string>` i jej wynikiem jest iterator, który znajduje się na samym początku stringa, na którym używamy danej funkcji,
- `end()` – analogicznie do `begin()`, wskazuje na końcowy iterator stringa. Jak reszta, należy do biblioteki `<string>`.
- `sort()` – należy do biblioteki `<algorithm>`. Przymuje dwa parametry: pierwszym jest początkowy iterator danej struktury, którą chcemy posortować, a drugim końcowy iterator tej samej struktury.

Słownik

> **sortowanie**
>
> uporządkowanie elementów danego zbioru według określonych kryteriów
> **tabela ASCII**
>
> przyporządkowuje liczbom z zakresu od 0 do 127 litery alfabetu łacińskiego, cyfry, znaki interpunkcyjne i inne symbole oraz polecenia sterujące; na przykład litera „a” jest kodowana jako liczba 97, a znak spacji jest kodowany jako 32

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Poniższy kod jest niepełną implementacją algorytmu sprawdzającego, czy podane dwa słowa są anagramami. Wewnątrz funkcji `czyAnagram` znajdują się dwa miejsca oznaczone trzema kropkami (`...`), które należy uzupełnić. Następnie dodaj do kodu dwie zmienne typu `string`, o nazwach `slowo1` oraz `slowo2`, i umieść w nich słowa o których wiesz, że na pewno są anagramami.

Specyfikacja:

Dane:

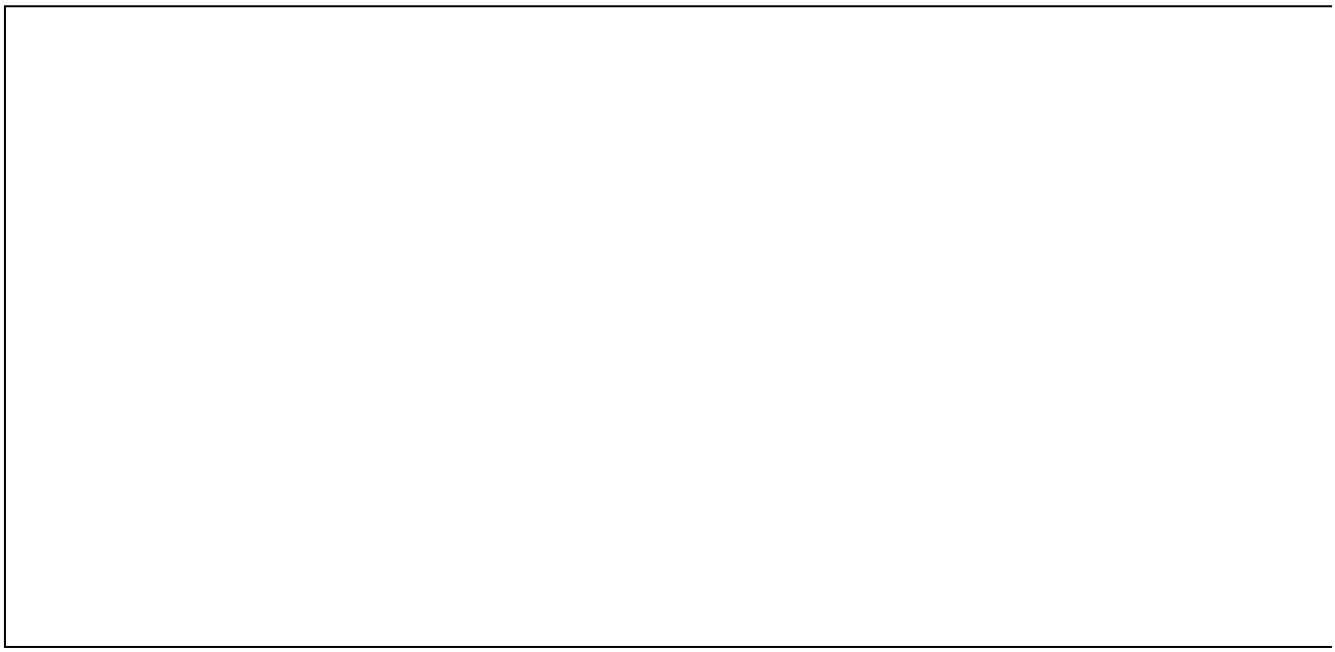
- `slowo1`, `slowo2` – zmienne typu *string*

Wynik:

Program na wyjściu standardowym wyświetli „Podane dwa wyrazy to anagramy” lub „Podane dwa wyrazy nie sa anagramami”.

Twoje zadania

1. Uzupełnij brakujące fragmenty funkcji `czyAnagram`. Następnie dodaj do kodu dwie zmienne typu `string` i umieść w nich słowa, o których wiesz, że na pewno są anagramami.



Ćwiczenie 2

Poniższy kod jest implementacją algorytmu sprawdzającego, czy podane dwa słowa są anagramami. Dodaj kod (wewnątrz funkcji `czyAnagram()`), który będzie weryfikował pierwszy warunek, dotyczący równej długości słów.

Specyfikacja:

Dane:

- `slowo1`, `slowo2` – zmienne typu *string*

Wynik:

Program na wyjściu standardowym wyświetli „Podane dwa wyrazy to anagramy” lub „Podane dwa wyrazy nie sa anagramami”.

Twoje zadania

1. Dodaj do kodu warunek, który sprawdza, czy oba wyrazy są równej długości. Nie zmieniaj przykładowych słów.

Ćwiczenie 3

Nie patrząc na kody z ćwiczenia 1. i 2., spróbuj napisać od początku algorytm sprawdzający, czy podane dwa słowa są anagramami. Użyj funkcji.

Specyfikacja:

Dane:

- `slowo1`, `slowo2` – zmienne typu *string*

Wynik:

Program na wyjściu standardowym wyświetli „Podane dwa wyrazy to anagramy” lub „Podane dwa wyrazy nie sa anagramami”.

Twoje zadania

1. Napisz od zera program, który sprawdza, czy podane dwa słowa są anagramami – jeżeli tak, to wypisuje komunikat tekstowy "Podane dwa wyrazy to anagramy". W przeciwnym razie, wypisuje "Podane dwa wyrazy nie sa anagramami".

Dla nauczyciela

Autor: zespół autorski Contentplus.pl sp. z o.o.
Przedmiot: Informatyka

Temat: Anagramy w języku C++

Grupa docelowa:

Liceum ogólnokształcące i technikum, 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.

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:

b) na tekstach: porównywania tekstów, wyszukiwania wzorca w tekście metodą naiwną, szyfrowania tekstu metodą Cezara i przestawieniową,

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

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):

- Przeanalizujesz implementację algorytmu weryfikującego, czy podana para wyrazów to anagramy.
- Napiszesz w języku C++ program sprawdzający, czy słowa są anagramami.
- Wykonasz kilka prostych zadań związanych z anagramami.

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: „Anagramy w języku C++”. Uczniowie mają zapoznać się z treściami w sekcji „Przeczytaj”.

Faza wstępna:

1. Prowadzący wyświetla na tablicy interaktywnej zawartość sekcji „Wprowadzenie” i omawia cele do osiągnięcia w trakcie lekcji.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie przystępują do cichego czytania tekstu e-materiału. Indywidualnie zapoznają się z treścią w sekcji „Film samouczek”.strong>Praca z tekstem. Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj”.
2. **Praca z multimedium.** Uczniowie w parach zapoznają się z treścią problemu 1 z sekcji „Film samouczek”. Opracowują swoje rozwiązania, a następnie porównują je z przedstawionymi w filmie. Następnie wspólnie analizują prezentację. Nauczyciel wyjaśnia ewentualne niezrozumiałe kwestie.
3. **Ćwiczenie umiejętności.** Uczniowie, pracując w parach, wykonują ćwiczenie nr 1 z sekcji „Sprawdź się”. Nauczyciel sprawdza poprawność pisanych kodów, porównuje je i omawia wraz z uczniami. Wskazuje najbardziej efektywne rozwiązanie.
4. Nauczyciel dzieli uczniów na grupy czteroosobowe. Uczniowie wykonują ćwiczenie nr 2 a następnie każda grupa wyznacza jedną osobę, którą wymienia się z inną grupą. W nowych zespołach uczniowie porównują swój kod i wybierają najbardziej efektywne rozwiązanie.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie” i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).
2. Wybrany uczeń podsumowuje zajęcia z programowania w C++, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się”.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- 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.

- 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:

- Treści w sekcji „Film samouczek" można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.