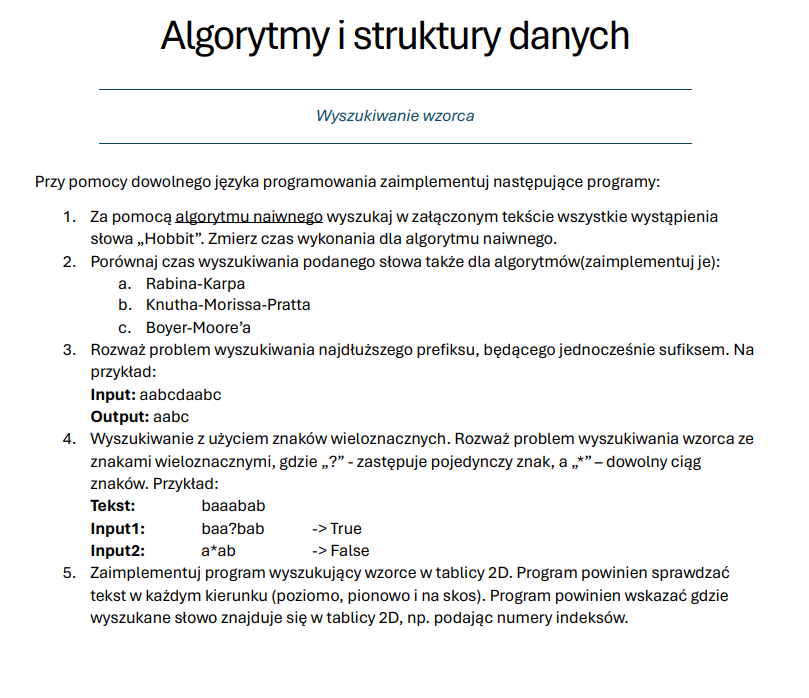
# Sprawozdanie, Zajęcia nr 4 – Drzewa BST.

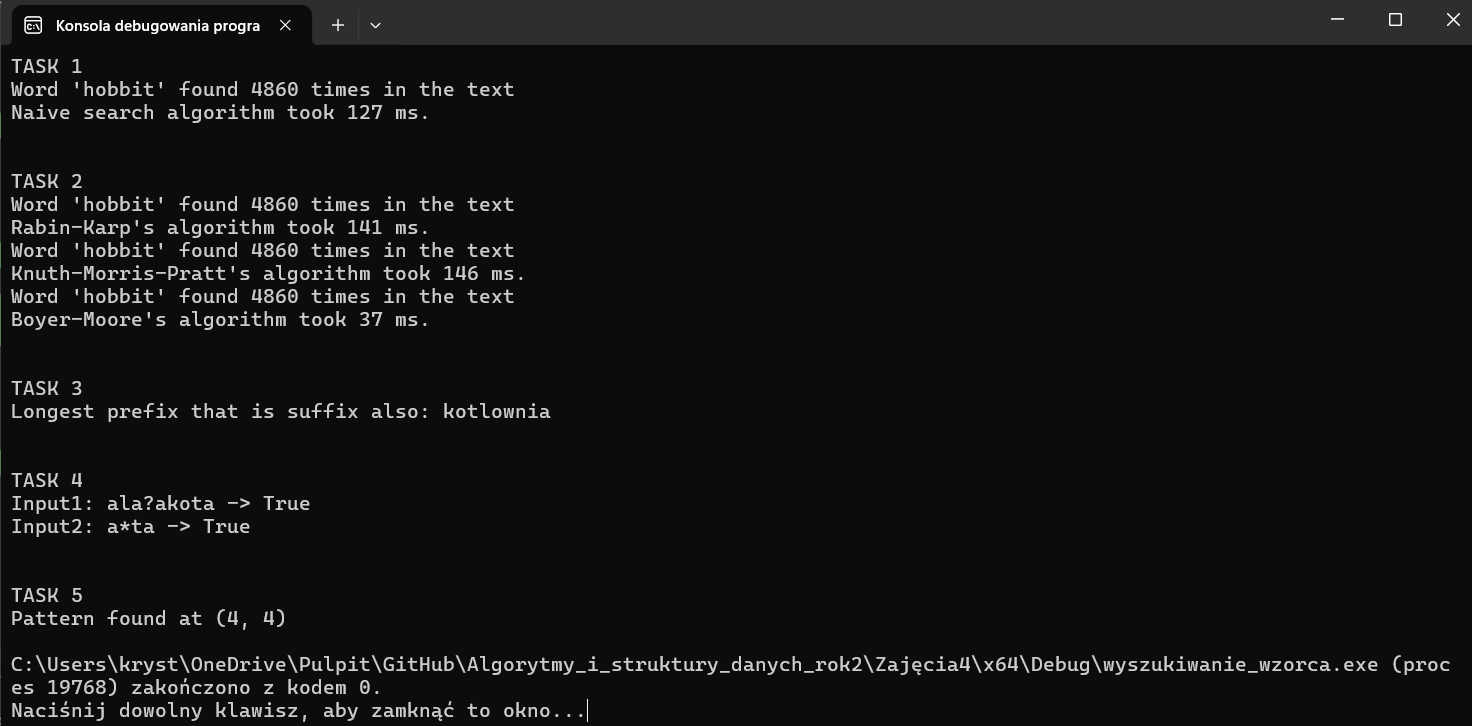
## Krystian Kostrzewa, 418845. WIMiIP, Informatyka Techniczna, sn. Data zajęć: 11.05.2024

1. **Wyszukiwanie wzorca**

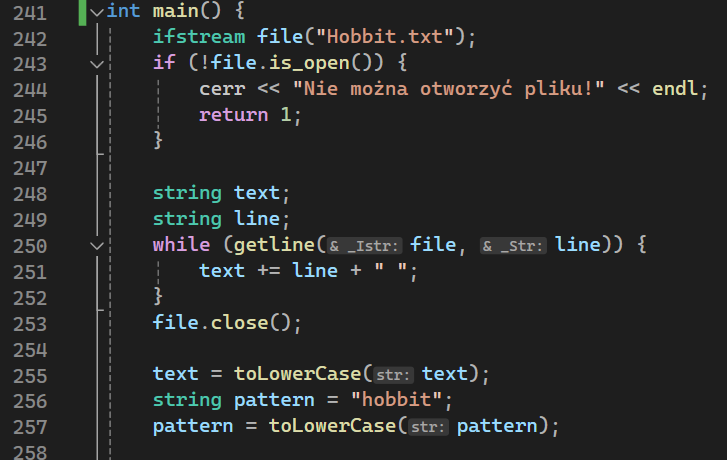
**Polecenia:**



**Wynik w konsoli:**

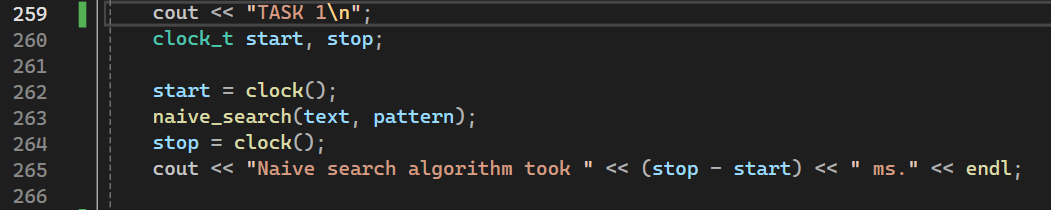


**Kod i opis rozwiązania:**

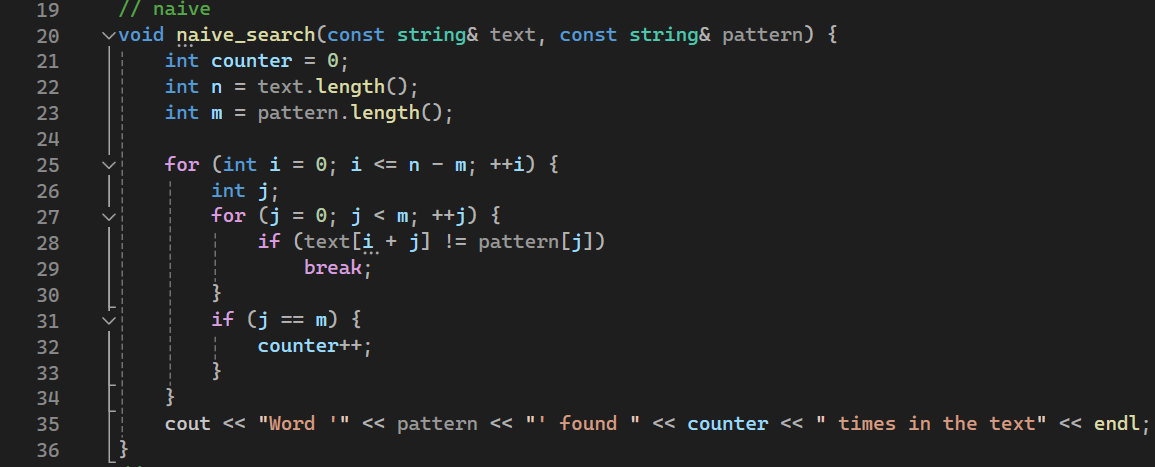
****

Fragment kodu na początku funkcji *main* otwiera plik "Hobbit.txt", wczytuje jego zawartość linia po linii, dodając każdą linię do zmiennej *text*, a następnie zamyka plik. Cały tekst i wzorzec "hobbit" są konwertowane na małe litery, aby umożliwić wyszukiwanie niewrażliwe na wielkość liter.

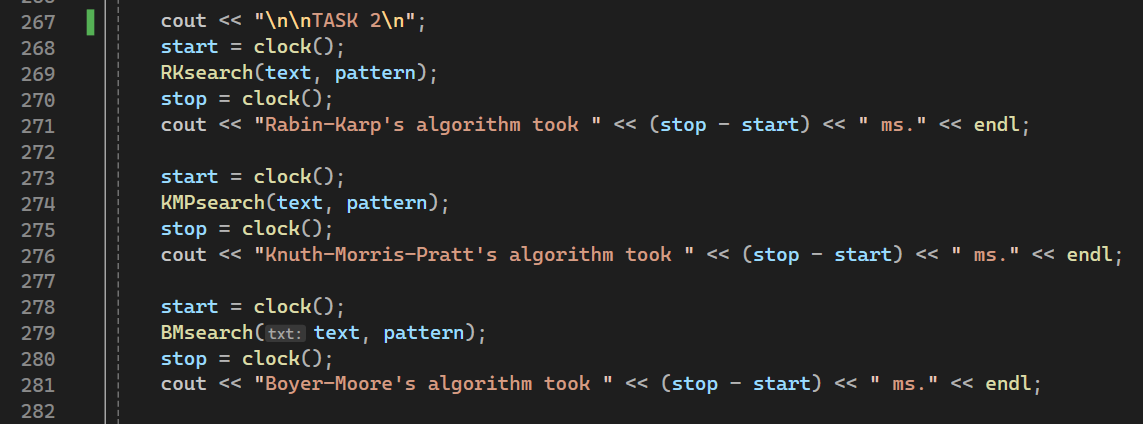
Rozwiązanie zadania 1:



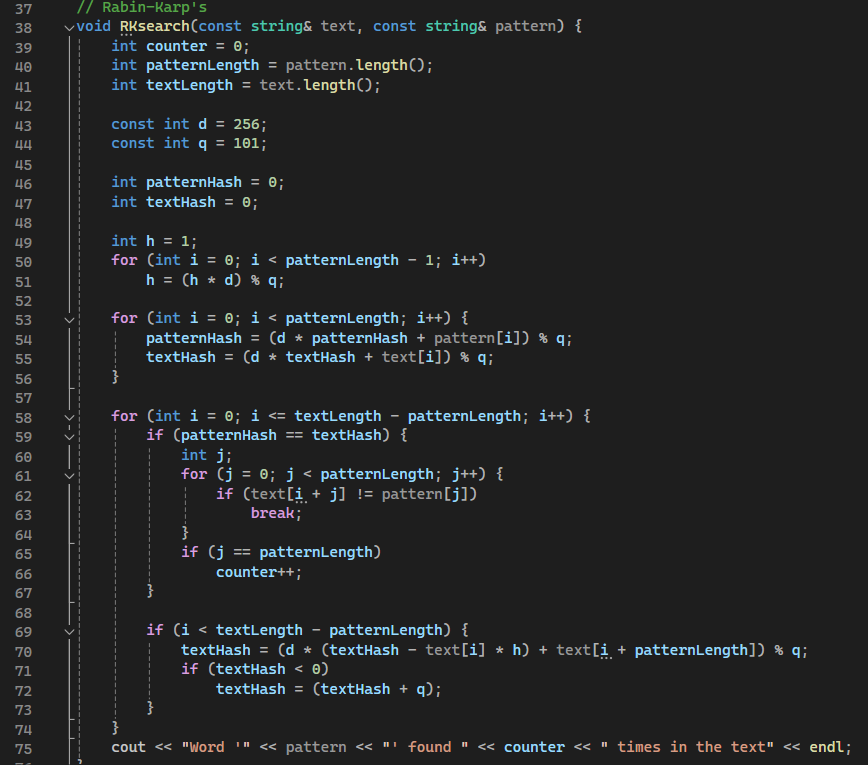
W funkcji main odbywa się proste wywołanie funkcji *naive\_*search, pobranie czasu przed i po wykonaniu operacji, a na końcu i opis jaka metoda znajdowania wzorca zajęła daną ilość czasu.



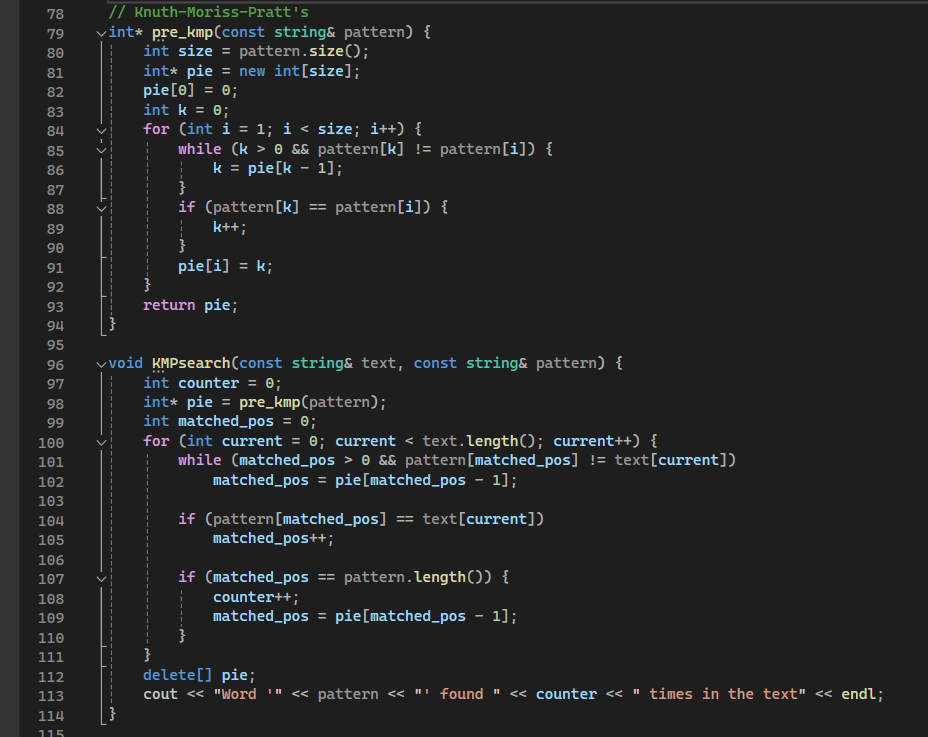
Funkcja *naive\_search* szuka wzorca (ciągu znaków) w tekście używając algorytmu wyszukiwania naiwnego. Dla każdego możliwego miejsca rozpoczęcia wzorca w tekście, sprawdza ona, czy wzorzec pasuje do fragmentu tekstu. Jeśli tak, zwiększa licznik znalezionych wystąpień. Na końcu wypisuje, ile razy wzorzec został znaleziony w tekście.

Rozwiązanie zadania 2: 

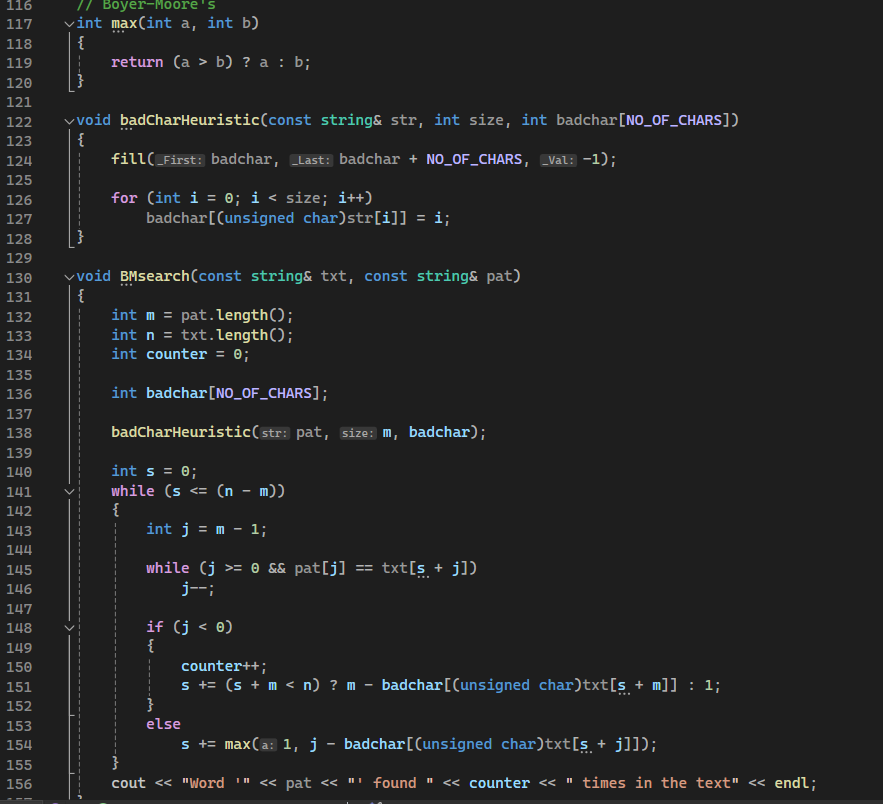
W funkcji main odbywa się proste wywołanie różnych funkcji wyszukiwania, pobranie czasu przed i po wykonaniu operacji, a na końcu i opis jaka metoda znajdowania wzorca zajęła daną ilość czasu.



Funkcja `RKsearch` używa algorytmu Rabina-Karpa do wyszukiwania słowa w tekście. Algorytm ten jest sprytnym sposobem na znajdowanie wzorców, który polega na przekształcaniu słów na liczby (hasze) i porównywaniu tych liczb zamiast bezpośredniego porównywania liter. Najpierw przekształca szukane słowo i pierwsze fragmenty tekstu na liczby. Potem przesuwa się przez cały tekst, za każdym razem obliczając nową liczbę dla aktualnego fragmentu tekstu. Gdy liczby się zgadzają, sprawdza, czy fragment tekstu rzeczywiście pasuje do szukanego słowa. W końcu, funkcja wypisuje, ile razy znaleziono szukane słowo w tekście.



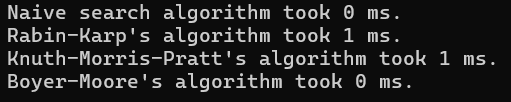
Funkcja `KMPsearch` używa algorytmu Knutha-Morrisa-Pratta (KMP) do wyszukiwania słowa w tekście. Algorytm KMP jest zaawansowaną metodą, która pozwala na efektywne przeszukiwanie tekstu bez zbędnego cofania się w tekście. Najpierw funkcja `pre\_kmp` tworzy pomocniczą tablicę (tablica prefiksów) dla szukanego słowa, która pomaga szybko znaleźć miejsce, gdzie można zacząć dalsze porównania po znalezieniu niezgodności. Potem funkcja `KMPsearch` używa tej tablicy, aby porównywać słowo z tekstem. Kiedy napotyka na niezgodność, nie zaczyna porównywania od nowa, tylko używa informacji z tablicy prefiksów, aby wiedzieć, gdzie kontynuować. Dzięki temu algorytm jest bardzo efektywny. Na koniec, funkcja wypisuje, ile razy znaleziono szukane słowo w tekście.



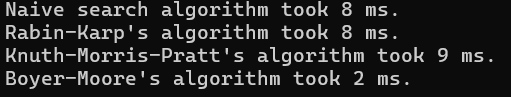
Funkcja `BMsearch` używa algorytmu Boyera-Moore'a do wyszukiwania słowa w tekście. Algorytm Boyera-Moore'a jest bardzo efektywną metodą, która przeszukuje tekst od prawej do lewej i używa dwóch heurystyk (reguł) do szybkiego przesuwania wzorca w przypadku niezgodności. Pierwsza heurystyka, `badCharHeuristic`, tworzy tablicę, która dla każdego znaku wzorca zapisuje ostatnią pozycję jego wystąpienia. W głównej funkcji `BMsearch`, wzorzec jest przesuwany nad tekstem, zaczynając od prawej strony. Gdy występuje niezgodność, algorytm sprawdza tablicę i przesuwa wzorzec o odpowiednią liczbę pozycji, zamiast porównywać wszystkie znaki od początku. Jeśli wzorzec pasuje do fragmentu tekstu, algorytm przesuwa wzorzec w prawo o pełną jego długość lub w zależności od wartości w tablicy heurystyki. Na koniec, funkcja wypisuje, ile razy znaleziono szukane słowo w tekście.

Pomiary:

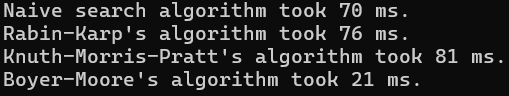
Pomiar dla tekstu przykładowego:



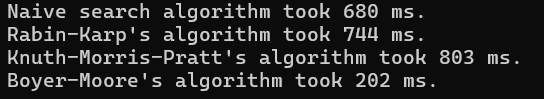
Pomiar dla tekstu przykładowego x10:



Pomiar dla tekstu przykładowego x100:



Pomiar dla tekstu przykładowego x1000:



Wnioski:

Na podstawie przedstawionych pomiarów dla różnych algorytmów wyszukiwania, można wyciągnąć następujące wnioski:

**Wydajność dla małych tekstów**

Dla pomiaru pojedynczego tekstu oraz dla tekstu powielonego 10 razy, wszystkie algorytmy działają bardzo szybko, z czasem wykonania na poziomie pojedynczych milisekund. Różnice są niewielkie, ale algorytm Boyer-Moore'a jest najszybszy.

**Wydajność dla średnich tekstów**

Dla tekstu powielonego 100 razy, czasy wykonania zaczynają się różnić bardziej znacząco. Algorytm Boyer-Moore'a pozostaje najszybszy (21 ms), podczas gdy reszta algorytmów potrzebuje więcej czasu (od 70 ms dla algorytmu wyszukiwania naiwnego do 81 ms dla algorytmu Knuth-Morris-Pratta).

**Wydajność dla dużych tekstów**

Dla tekstu powielonego 1000 razy, różnice w czasach wykonania są najbardziej widoczne. Algorytm Boyer-Moore'a jest nadal najszybszy (202 ms), natomiast pozostałe algorytmy wykazują znacząco większe czasy wykonania (od 680 ms dla algorytmu wyszukiwania naiwnego do 803 ms dla algorytmu Knuth-Morris-Pratta).

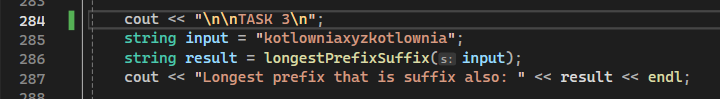
**Ogólne porównanie**

Algorytm Boyer-Moore'a jest zdecydowanie najbardziej wydajny dla dużych tekstów, co wynika z jego złożoności czasowej O(n/m) w najgorszym przypadku (gdzie n to długość tekstu, a m to długość wzorca).

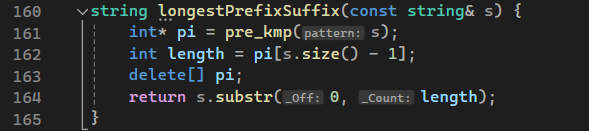
Algorytm wyszukiwania naiwnego, mimo swojej prostoty, radzi sobie zaskakująco dobrze dla małych tekstów, ale jego wydajność drastycznie spada wraz ze wzrostem długości tekstu.

Algorytmy Rabin-Karpa i Knuth-Morris-Pratta wykazują lepszą wydajność niż algorytm naiwny, ale nie są tak szybkie jak algorytm Boyer-Moore'a dla dużych tekstów.

Rozwiązanie zadania 3:

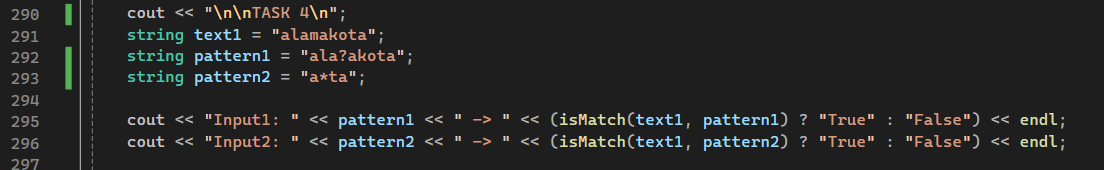


W funkcji main odbywa się proste wywołanie funkcji i opis co zostanie wyświetlone.

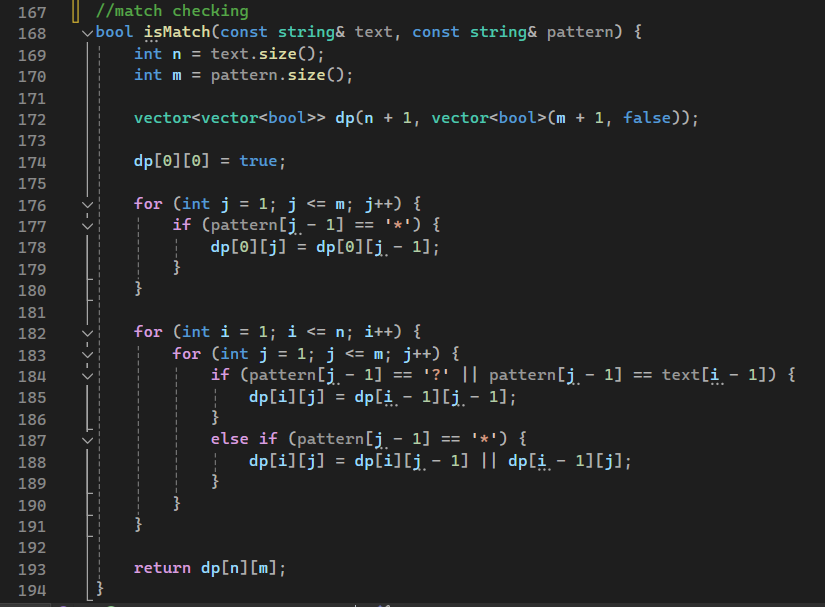


Funkcja *longestPrefixSuffix* służy do znajdowania najdłuższego prefiksu, który jest jednocześnie sufiksem dla danej ciągłej zmiennej *s*. Wykorzystuje ona algorytm obliczania tablicy przejścia KMP (funkcji *pre\_kmp*), która została użyta wcześniej w algorytmie wyszukiwania wzorca Knutha-Morrisa-Pratta (KMPsearch). Po obliczeniu tablicy przejścia, funkcja uzyskuje długość najdłuższego prefiksu-sufiksu poprzez odczytanie ostatniego elementu tablicy przejścia. Następnie zwraca ona odpowiednią część ciągu *s*, zaczynającą się od indeksu 0 i o długości równej długości obliczonego najdłuższego prefiksu-sufiksu.

Rozwiązanie zadania 4:

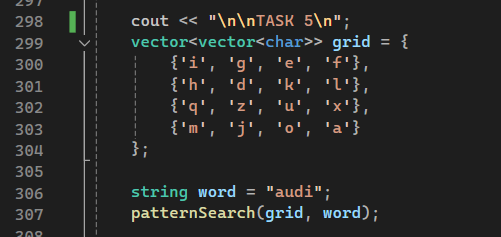


W funkcji main odbywa się przygotowanie zmiennych do sprawdzenia, proste wywołanie funkcji i odpowiedź, czy zgodnie z instrukcjami teksty się pokrywają.

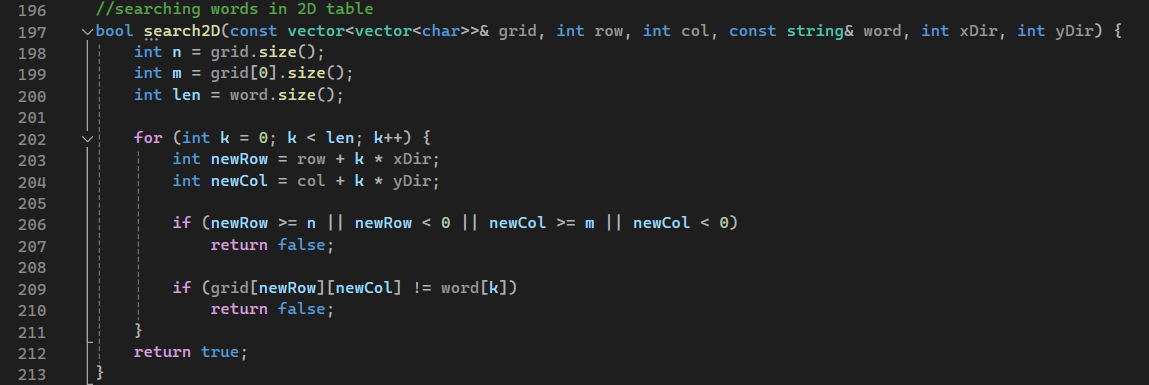


Funkcja *isMatch* służy do sprawdzania, czy dany wzorzec *pattern* pasuje do danego tekstu *text*, uwzględniając znaki wieloznaczne: znak zapytania *?* zastępuje pojedynczy znak, a gwiazdka *\** oznacza dowolny ciąg znaków (w tym także brak znaku). Algorytm wykorzystuje dynamiczne programowanie, tworząc tablicę *dp*, gdzie *dp[i][j]* oznacza, czy pierwsze *i*  znaków tekstu i pierwsze *j* znaków wzorca pasują do siebie. Iterując przez tekst i wzorzec, algorytm aktualizuje wartości tablicy *dp* na podstawie warunków: dopasowanie znaku, znaku wieloznacznego *?* lub gwiazdki *\**. Po przetworzeniu całego tekstu i wzorca, funkcja zwraca *true*, jeśli cały wzorzec pasuje do tekstu, lub *false* w przeciwnym przypadku.

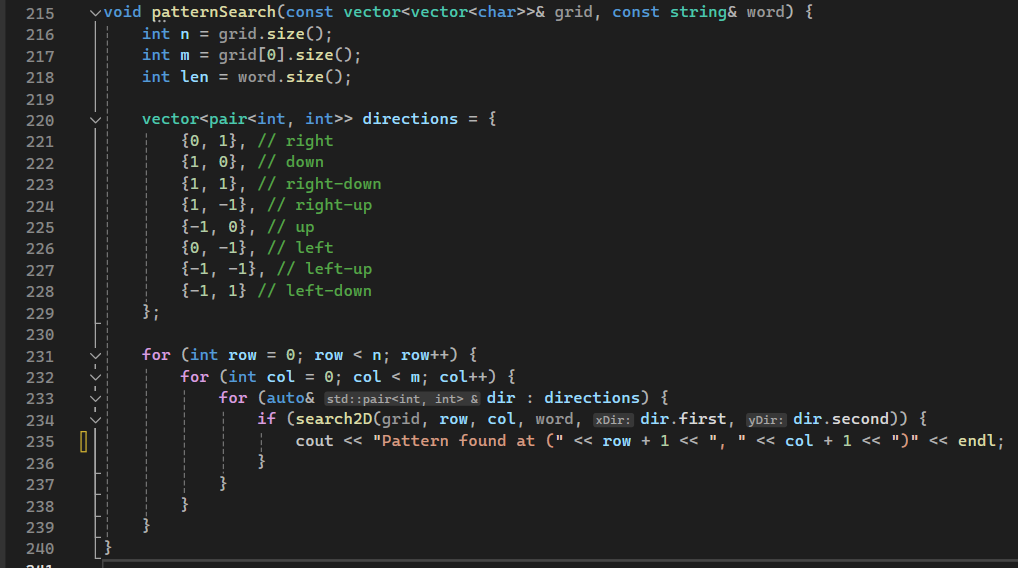
Rozwiązanie zadania 5:



W funkcji main odbywa się przygotowanie tabeli dwuwymiarowej, zmiennej przechowującej słowo, którego będziemy szukać oraz wywołanie funkcji.



Funkcja *search2D* służy do wyszukiwania słowa w dwuwymiarowej tabeli (*grid*). Przyjmuje ona pozycję początkową (*row, col*) w tabeli oraz kierunki (*xDir* i *yDir*), w których ma szukać słowa. Następnie iteruje po każdej literze słowa i sprawdza, czy kolejne litery słowa odpowiadają kolejnym pozycjom w tabeli przesuniętym o odpowiednie wartości zgodnie z kierunkiem. Jeśli którykolwiek z warunków brzegowych jest naruszony (np. wyjście poza granice tabeli), funkcja zwraca *false*, w przeciwnym razie zwraca *true*, co oznacza, że słowo zostało znalezione w tabeli w określonym kierunku. Jest ona używana w docelowej funkcji *patternSearch*, którą wywołujemy w *main*ie.



Funkcja *patternSearch* służy do wyszukiwania wzorca (*word*) w dwuwymiarowej tabeli (*grid*). Najpierw określa wymiary tabeli oraz długość szukanego słowa. Następnie definiuje kierunki, w których ma odbywać się wyszukiwanie, reprezentowane jako pary liczb określające przesunięcia wierszy (*first*) i kolumn (*second*). W kolejnych zagnieżdżonych pętlach iteruje po każdym polu tabeli i dla każdego pola iteruje po wszystkich kierunkach, wywołując funkcję *search2D*, aby sprawdzić, czy słowo znajduje się w tabeli zaczynając od danego pola i w określonym kierunku. Jeśli słowo zostanie znalezione, funkcja wypisuje informację o jego pozycji.