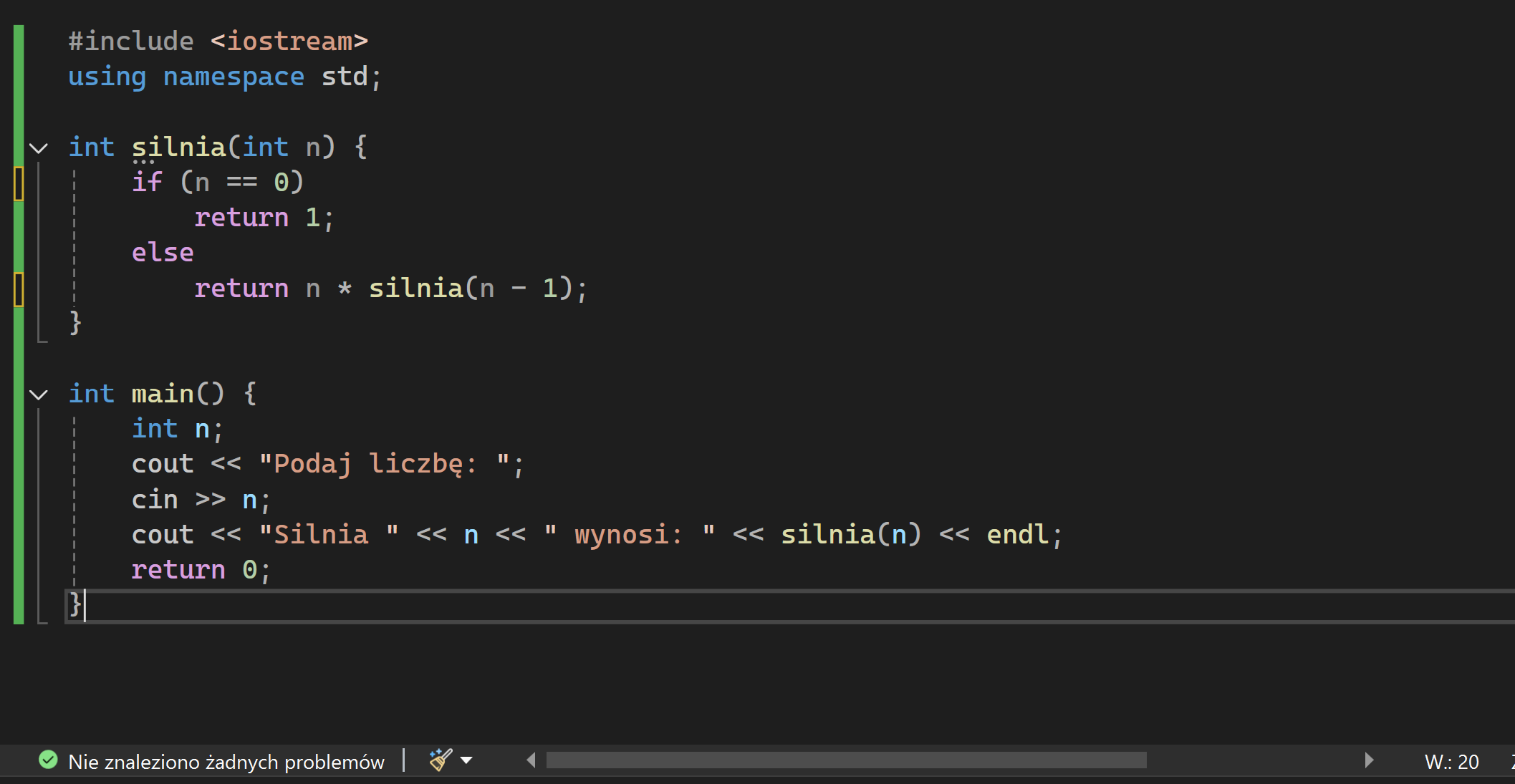
**Mariia Kobrina L02 Informatyka**

**Sprawozdanie z ćwiczenia 01 – Wykorzystanie rekurencji do rozwiązywania problemów algorytmicznych w C++**

**1. Wprowadzenie** Celem ćwiczenia było zapoznanie się z techniką rekurencji i jej zastosowaniem w języku C++. Rekurencja to technika, w której funkcja wywołuje samą siebie, co pozwala na rozwiązanie problemów algorytmicznych.

**Zadanie 1: Obliczanie silni (n!) za pomocą rekurencji**

**Kod:**



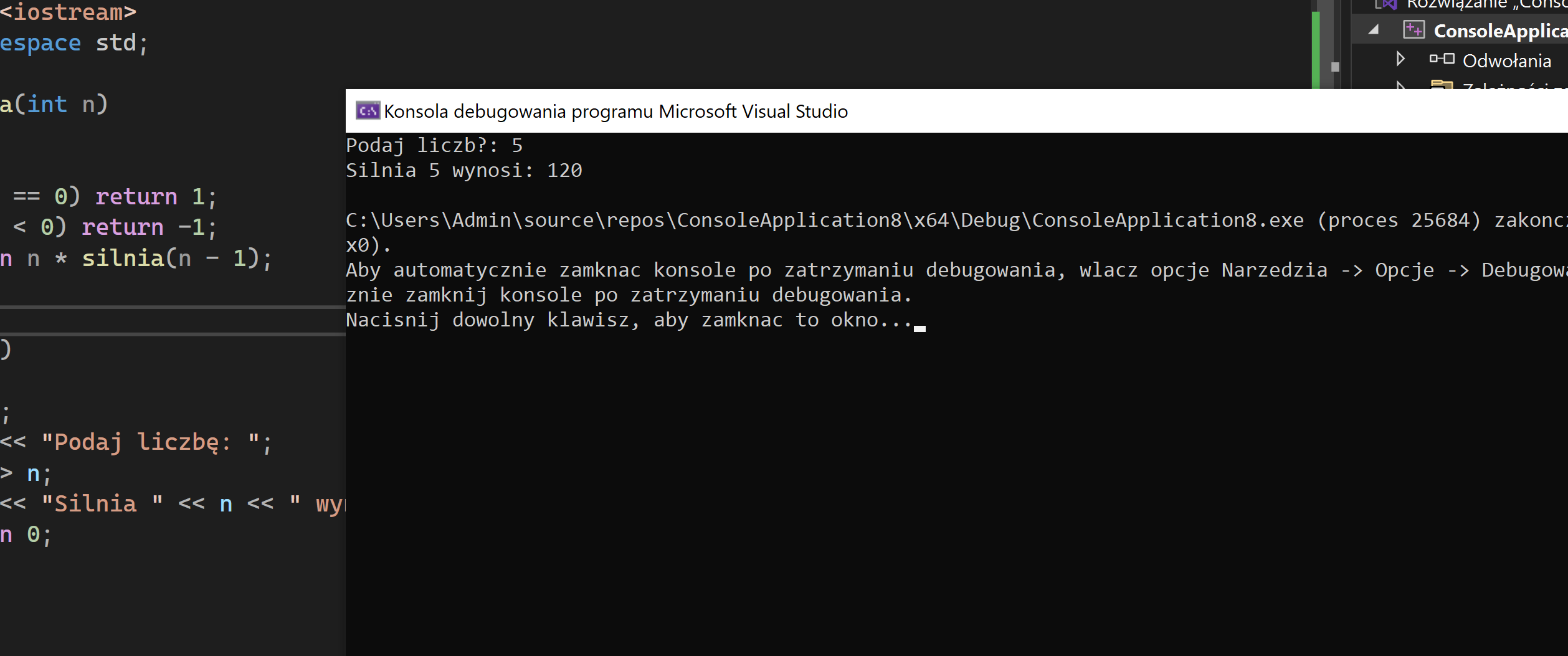
**Wyjasnienie kodu:**

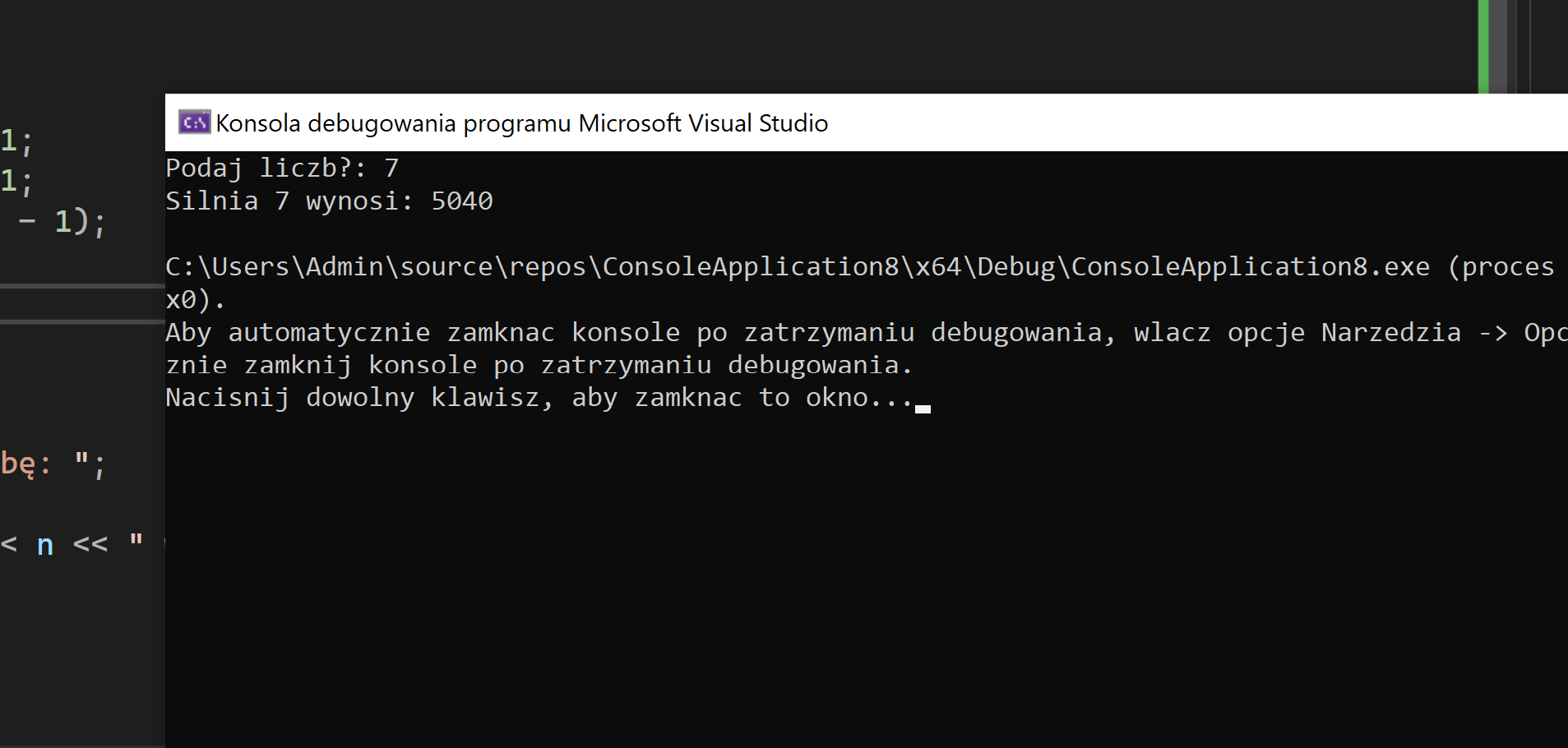
int silnia(int n) - Deklaracja funkcji, która oblicza silnię liczby całkowitej n (czyli n!).

Warunek zakończenia rekurencji: if (n == 0) - Funkcja kończy rekurencję, kiedy n wynosi 0, ponieważ silnia 0 jest zdefiniowana jako 1 (0! = 1).

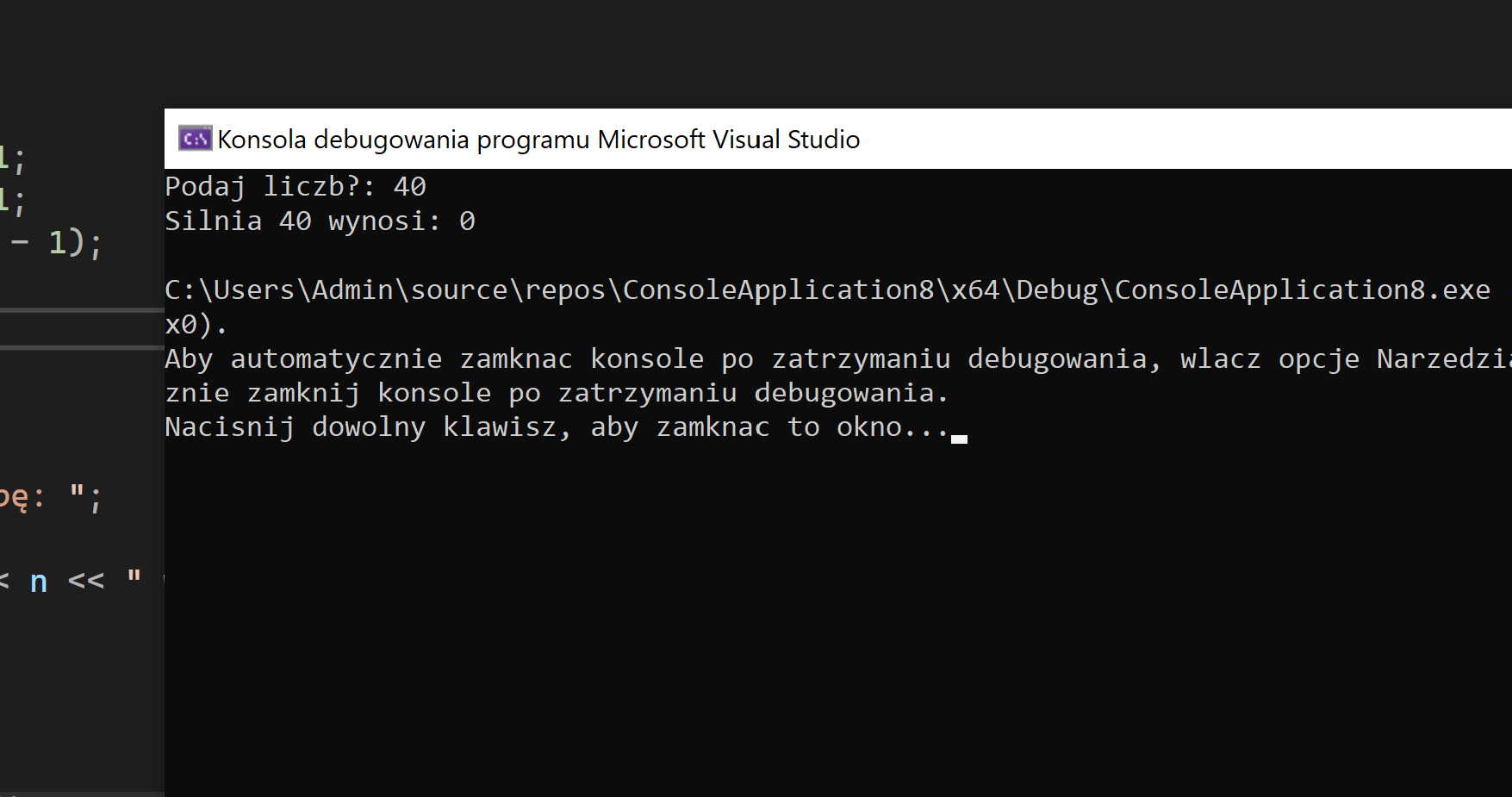
Wywołanie rekurencyjne: return n \* silnia(n - 1); - W przeciwnym przypadku funkcja wywołuje samą siebie z argumentem n - 1 i zwraca iloczyn n oraz silni n-1. To wywołanie rekurencyjne powtarza się, aż n osiągnie 0.

**Wyniki:**



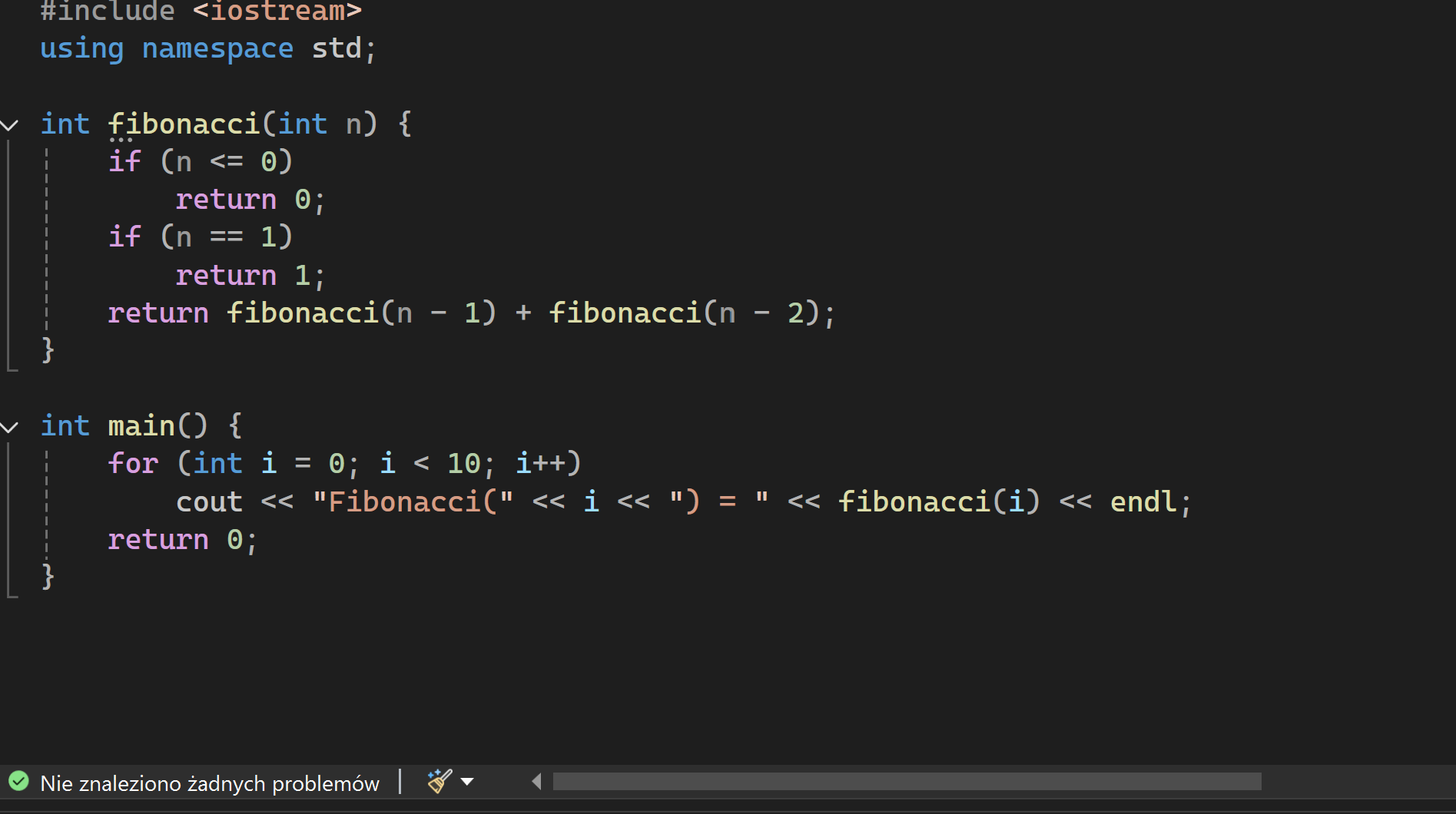


**Wnioski:** Funkcja poprawnie oblicza silnię, ale dla dużych wartości n wydajność spada.

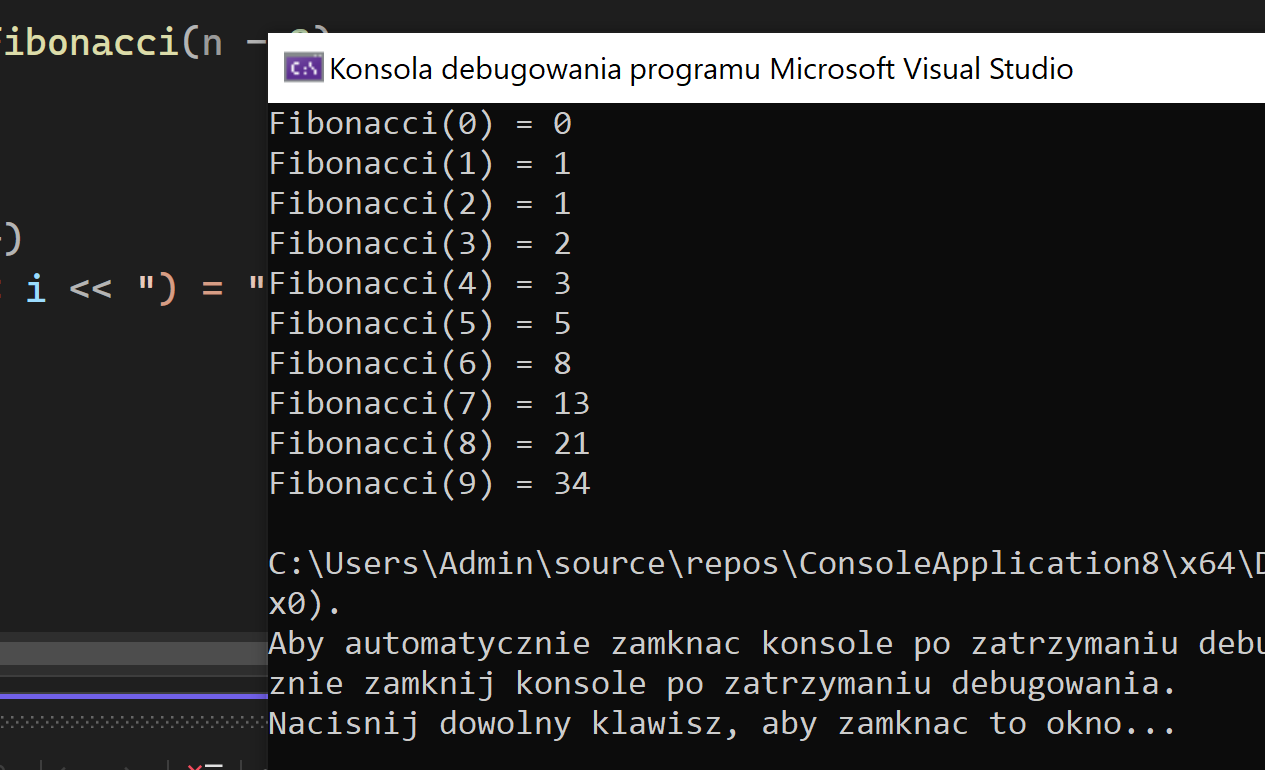


**Zadanie 2: Obliczanie n-tego elementu ciągu Fibonacciego**

**Kod:**

****

**Wyniki:**



**Wyjasnienie kodu:**

int fibonacci(int n) - Funkcja, która oblicza n-ty element ciągu Fibonacciego.

Definicja ciągu Fibonacciego: Ciąg Fibonacciego to ciąg liczb, w którym każda liczba jest sumą dwóch poprzednich, z dwoma pierwszymi elementami wynoszącymi 0 i 1. Czyli:  
F(0) = 0, F(1) = 1, F(n) = F(n - 1) + F(n - 2) dla n > 1

cout << "Fibonacci(" << i << ") = " << fibonacci(i) << endl; - Dla każdej wartości i od 0 do 9, funkcja fibonacci(i) oblicza i zwraca n-ty element ciągu Fibonacciego. Wynik jest wypisywany w formie:  
"Fibonacci(i) = wynik"

**Wnioski:** Rekurencyjna implementacja działa.

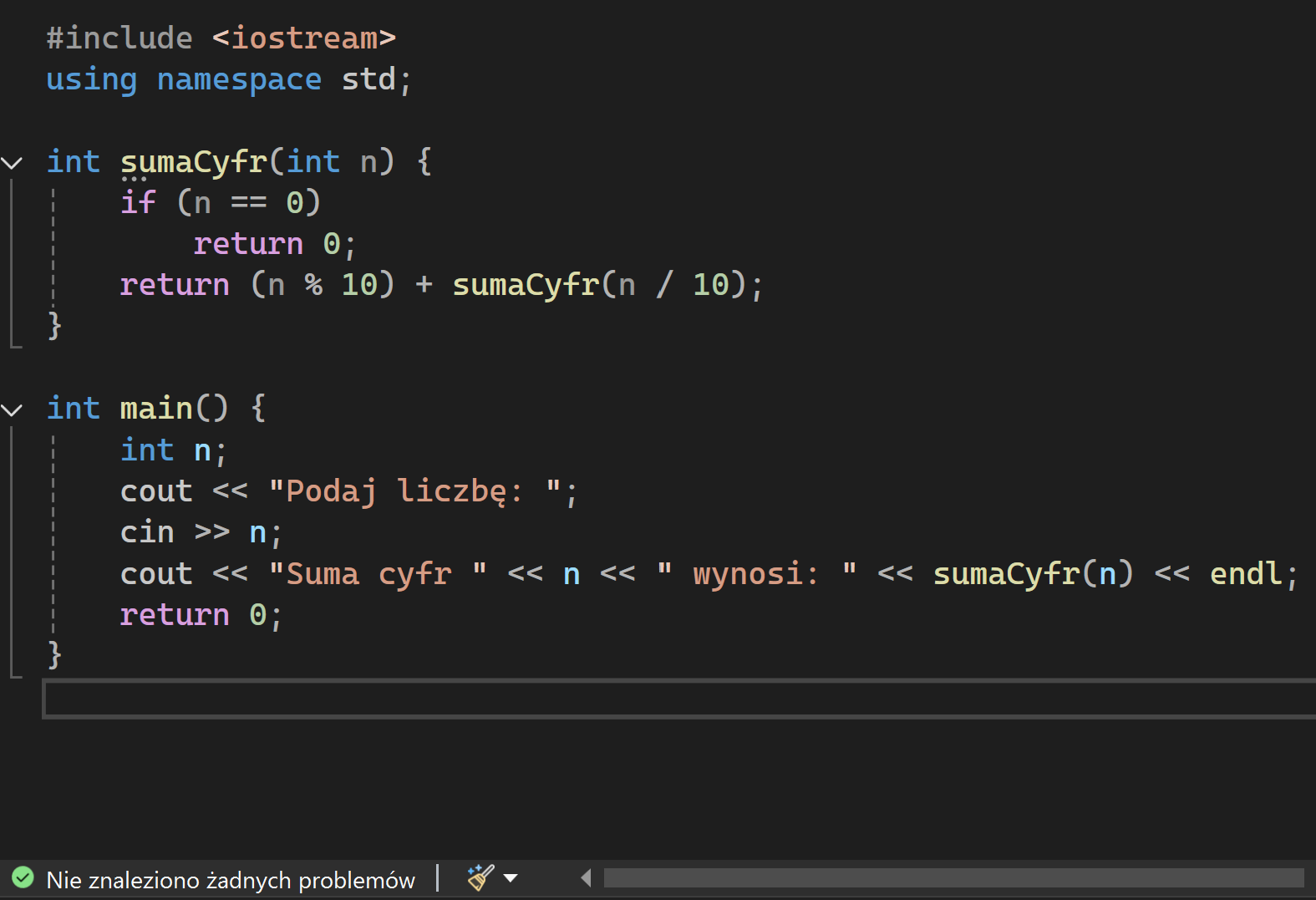
Problemy związane z rekurencją:

Duża liczba wywołań: Ponieważ dla każdego wywołania funkcji obliczane są dwie kolejne wartości (fibonacci(n-1) oraz fibonacci(n-2)), dla dużych wartości n może dojść do bardzo dużej liczby wywołań, co prowadzi do nadmiernego zużycia pamięci i procesora.

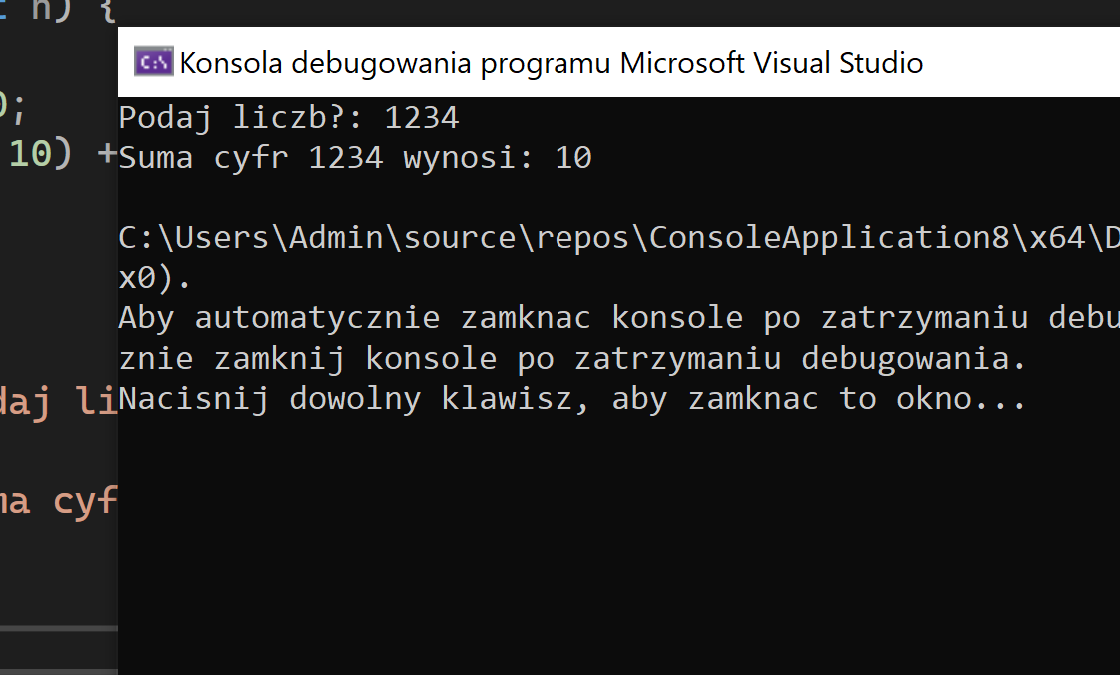
Brak optymalizacji: Program nie wykorzystuje żadnych technik optymalizacyjnych, takich jak zapamiętywanie wyników, co prowadzi do obliczania tych samych wyników wielokrotnie. Na przykład dla fibonacci(5) funkcja będzie obliczać fibonacci(3) i fibonacci(2) wielokrotnie.

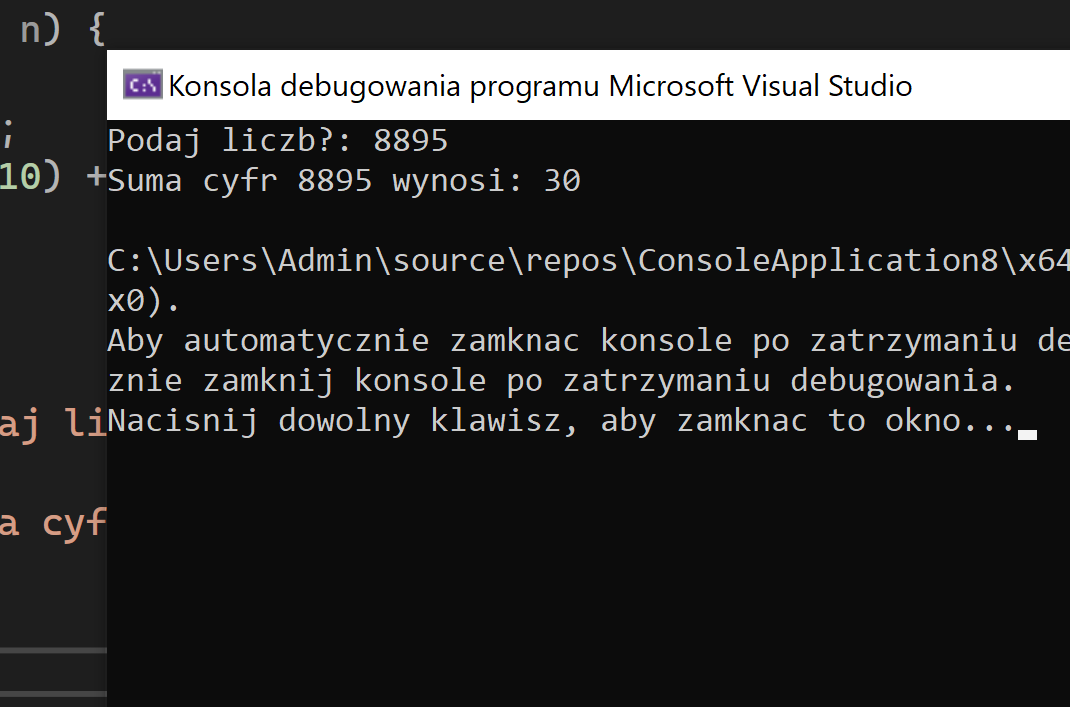
**Zadanie 3: Obliczanie sumy cyfr liczby**

**Kod:**

****

**Wyniki:**

****

****

**Wyjasnienie kodu:**

Program używa rekurencji do obliczenia sumy cyfr liczby n. Funkcja sumaCyfr wywołuje samą siebie, przekazując do niej zredukowaną liczbę (po usunięciu ostatniej cyfry) i sumując ostatnią cyfrę z wynikiem rekurencyjnego wywołania.

Działanie funkcji sumaCyfr jest następujące:

Operacja n % 10 - Oblicza ostatnią cyfrę liczby n (np. dla n = 1234, 1234 % 10 = 4).

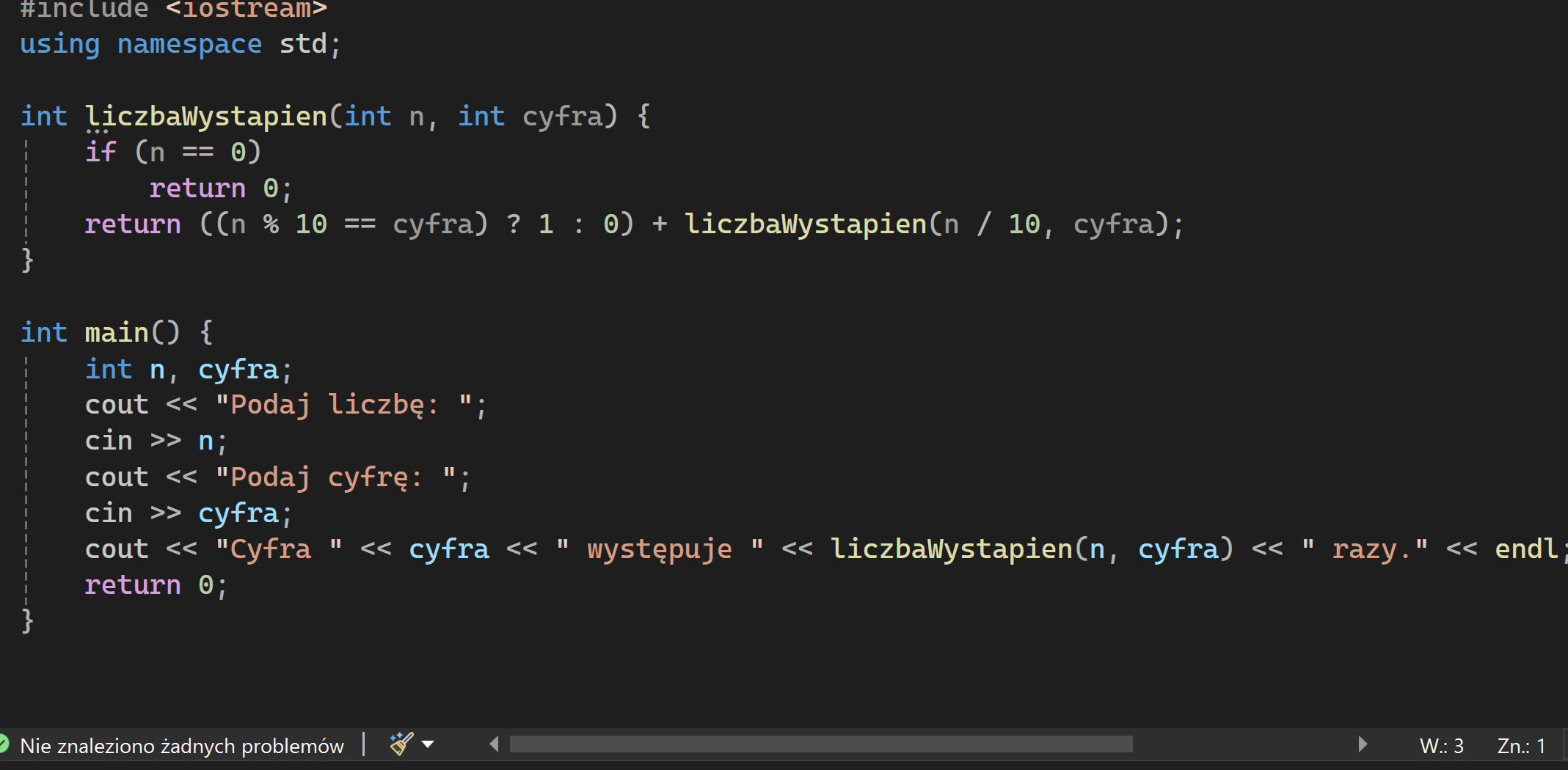
Operacja n / 10 - Usuwa ostatnią cyfrę z liczby (np. dla n = 1234, 1234 / 10 = 123).

Funkcja rekurencyjnie sumuje ostatnią cyfrę z wynikiem dalszego wywołania dla liczby n / 10.

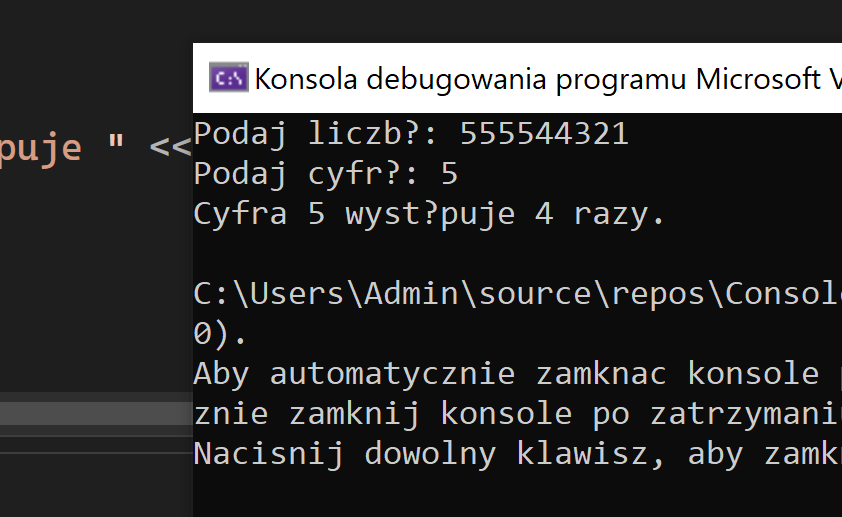
**Wnioski:** Program poprawnie oblicza sumę cyfr liczby przy użyciu rekurencji.

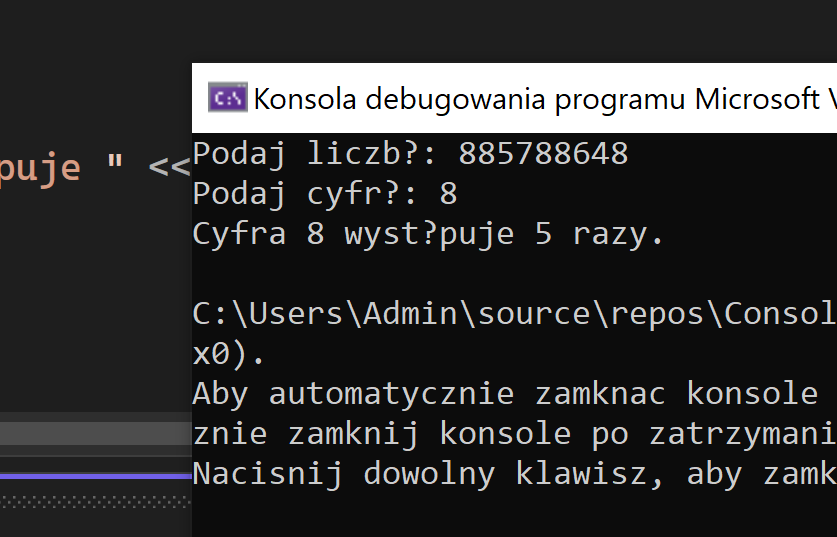
**Zadanie 4: Liczenie liczby wystąpień danej cyfry w liczbie**

**Kod:**



**Wyniki:**





**Wyjasnienie kodu:**

Funkcja liczbaWystapien(int n, int cyfra) jest używana do obliczenia liczby wystąpień cyfry w liczbie n. Funkcja ta korzysta z rekurencji, aby przejść przez każdą cyfrę liczby n i sprawdzić, czy dana cyfra jest równa podanej cyfrze cyfra.

Rekurencja działa, dzieląc liczbę przez 10 (czyli usuwając ostatnią cyfrę) i w każdym wywołaniu sprawdzając, czy ostatnia cyfra liczby jest równa cyfra.

n % 10 == cyfra - Ta część kodu sprawdza, czy ostatnia cyfra liczby n jest równa cyfrze cyfra. Jeśli tak, zwraca 1, w przeciwnym razie zwraca 0.

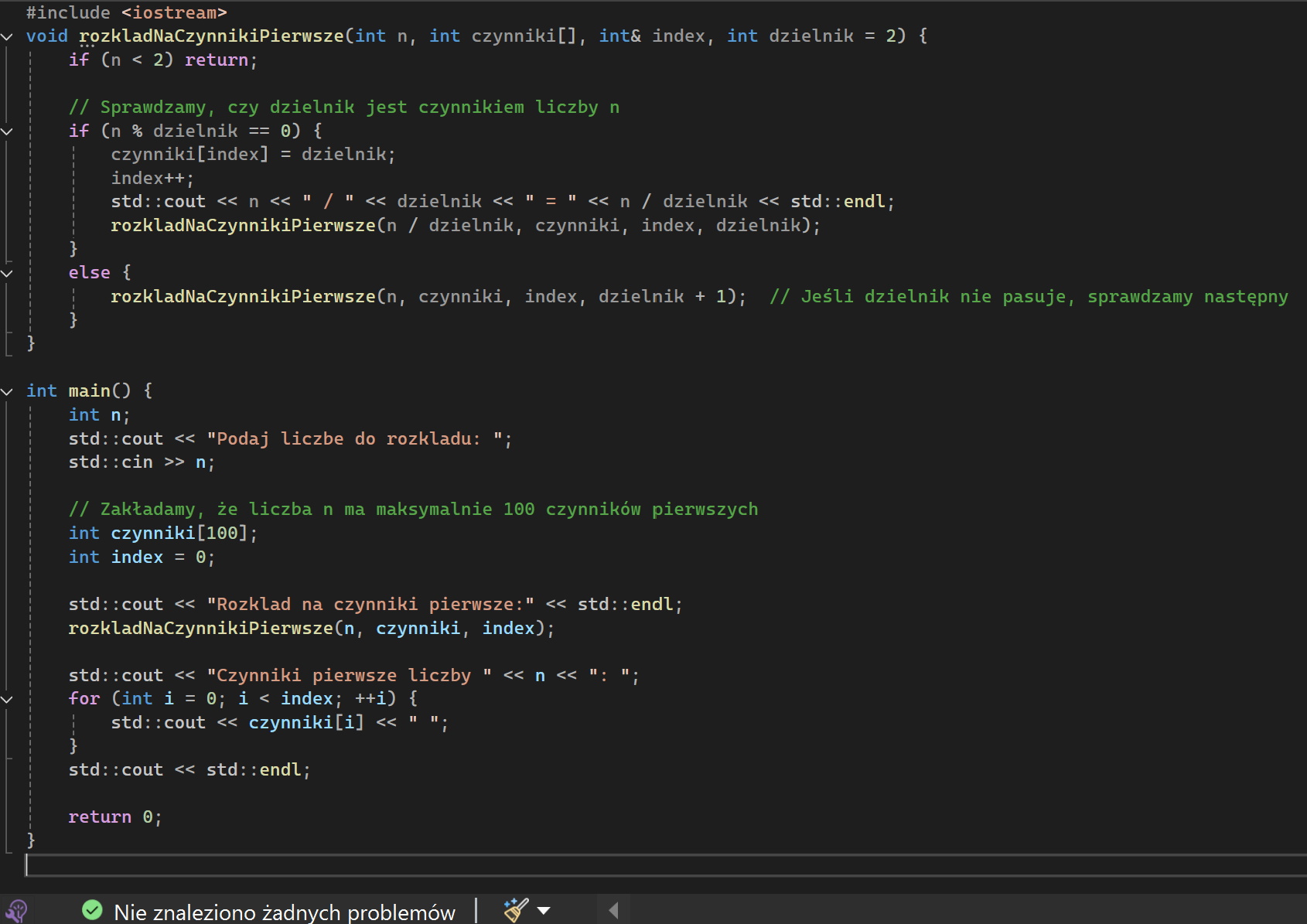
liczbaWystapien(n / 10, cyfra) - Rekurencyjne wywołanie funkcji, które skraca liczbę n przez usunięcie ostatniej cyfry (dzieląc ją przez 10). Ta część zapewnia, że funkcja przechodzi przez wszystkie cyfry liczby.

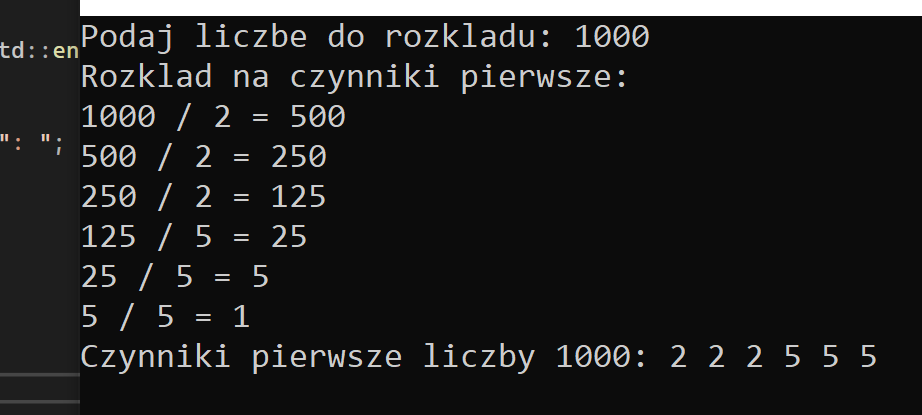
Wynik tego wywołania dodaje 1, jeśli ostatnia cyfra była równa poszukiwanej, lub 0, jeśli nie była.

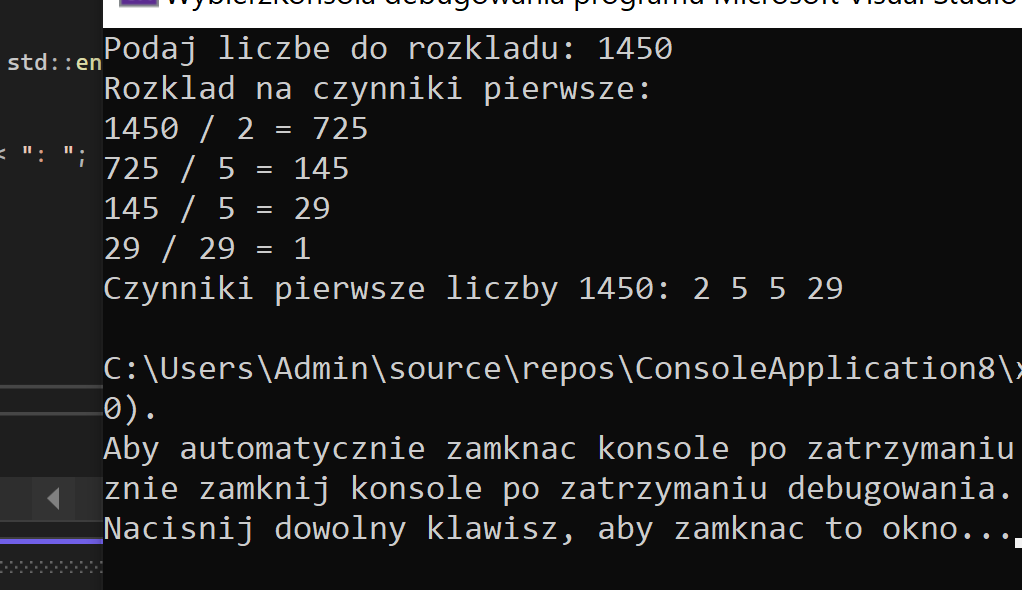
**Wnioski:** Program poprawnie oblicza liczbę wystąpień danej cyfry w liczbie przy użyciu rekurencji.

**Zadanie 5: Rozkładanie liczby na czynniki pierwsze**

**Kod:**

**Wyniki:**





### Wyjaśnienie kodu

**Funkcja** rozkladNaCzynnikiPierwsze przyjmuje cztery argumenty:

* + - n — liczba do rozkładu na czynniki,
    - czynniki[] — tablica, do której zapisujemy znalezione czynniki,
    - index — zmienna, która śledzi, na jakiej pozycji w tablicy zapisujemy nowy czynnik,
    - dzielnik — bieżący dzielnik, początkowo ustawiony na 2.

Wewnątrz funkcji:

Jeśli n jest mniejsze niż 2, zakończymy rekurencję.

Sprawdzamy, czy n jest podzielne przez dzielnik (początkowo 2). Jeśli tak, zapisujemy dzielnik do tablicy czynniki[] i wywołujemy funkcję rekurencyjnie z nową wartością n (czyli n / dzielnik).

Jeśli liczba nie jest podzielna przez dzielnik, zwiększamy dzielnik o 1 i sprawdzamy następny.

**Wymiki:** Po zakończeniu rekurencji, program wypisuje wszystkie czynniki pierwsze znalezione w tablicy. Program pozwolił na lepsze zrozumienie rekurencji jako narzędzia do rozwiązywania problemów, które wymagają wielokrotnego wywoływania tej samej funkcji z nowymi danymi. Tablica czynniki[] przechowuje czynniki pierwsze, a zmienna index pozwala na kontrolowanie, w której pozycji tablicy zapisujemy nowe czynniki. Rozkład liczby na czynniki pierwsze przy użyciu rekurencji jest efektywnym i prostym rozwiązaniem problemu. Ćwiczenie umożliwiło lepsze zrozumienie operacji na tablicach, rekurencji oraz manipulacji zmiennymi w języku C++.

**3. Podsumowanie**

Zapoznałam się z techniką rekurencji oraz jej zastosowaniem w rozwiązywaniu probłemów algorytmicznych. Rekurencyjne podejście jest efektywne i eleganckie dla małych liczb, ale może być niewydajne dla dużych liczb, prowadząc do problemów z głębokością rekurencji i zużyciem pamięci.