Zadania1

Zadanie 1: Tradycji musi stać się zadość. Zapoznaj się ze środowiskiem pracy, przygotuj sobie wybrany edytor tekstu oraz kompilator, a następnie napisz, skompiluj i uruchom program `Hello world!`. Z drugiej strony – kurczowe trzymanie się tradycji bywa zgubne, więc niech raczej wypisuje na ekran coś wybranego przez Ciebie.

a) Jeśli wszystko działa, dopisz do programu prostą funkcję, która przyjmuje liczbę całkowitą, zapisuje jej kwadrat do nowej zmiennej i zwraca tę zmienną jako wynik. Użyj jej w programie.

b) Wejdź na stronę `https://godbolt.org/`, wklej swój program i podejrzyj jak wygląda w kodzie maszynowym.

c) Dla przejrzystości na stronie pozostaw jedynie kod napisanej poprzednio funkcji i zobacz jak zmienia się przy manipulowaniu optymalizacją kodu (opcja `-On`, gdzie `n`$\in \{0, 1, 2, 3\}$). Co zaobserwujesz w sytuacji gdy każesz funkcji zwracać jakąś stałą wartość (czyli mnożenie jest nadal wykonywane, ale jego wynik nie ma wpływu na zwracaną wartość)?

Zadanie 2: Napisz program, który wypisze wszystkie liczby pierwsze mniejsze bądź równe podanej przez użytkownika liczbie naturalnej.

Zadanie 3: Problem Collatza (`https://pl.wikipedia.org/wiki/Problem_Collatza`) to problem o bardzo prostym sformułowaniu, lecz w praktyce nadal nierozwiązany. Polega on na wzięciu dowolnej liczby naturalnej c_0, a następnie postępowaniu wedle następującej rekurencji:

$$c_{n+1} = \begin{cases} \frac{1}{2}c_n & \text{dla parzystej } c_n \\ 3c_n + 1 & \text{dla nieparzystej } c_n \end{cases}$$

Hipotezę problemu koroborują jak dotąd obserwacje, ale nadal nie jest potwierdzona na drodze dowodu matematycznego. Stanowi ona, że niezależnie od jakiej liczby c_0 rozpoczniemy, zawsze istnieje skończone n, dla którego otrzymamy $c_n = 1$ (a w praktyce ciąg zacznie wtedy cyklicznie powtarzać wartości 4, 2, 1).

a) Napisz program, który pobiera od użytkownika początkowy wyraz, a następnie przebiega ciąg Collatza aż do momentu osiągnięcia wartości 1. Zawrzyj algorytm w postaci funkcji, która przyjmuje c_0 i zwraca liczbę kroków potrzebnych do osiągnięcia pierwszej jedynki. Niech funkcja ta posiada także drugi argument – typu logicznego, który pozwala decydować czy chcemy, by wypisywała wszystkie kolejne wyrazy czy nie.

b) Poeksperymentuj z różnymi parametrami wejściowymi, sprawdź czy otrzymujesz wyniki zgodne z tym co można znaleźć online. Np. dla $c_0 = 8400511$ powinno wyjść 685 kroków.

c) Przeskanuj zakres wszystkich liczb naturalnych (które bierzemy jako c_0) pomiędzy 1 a 10^6 i znajdź wartość, dla której ciąg jest najdłuższy (powinny wyjść 524 kroki przy $c_0 = 837799$).

*) Udoskonal program tak, by podawał również największą wartość c_n osiąganą podczas wywołania funkcji. Przykładowo dla $c_0 = 8400511$ powinno wyjść $c_{max} = 159424614880$.

**) Jeśli chcesz, spróbuj zapisać ten sam algorytm w Pythonie i porównaj czas wykonywania. Najprościej chyba do pomiaru wersji w `C++` użyć linuxowego polecenia `time` (np. `time ./collatz`).

Zadanie 4: Napisz program, który prosi użytkownika o liczbę naturalną, a następnie oblicza i zwraca jej silnię.

a) W pierwszym wariancie wykorzystaj pętlę `for`.

b) Następnie przerób swój kod tak, aby wykorzystywał rekurencję.

c) Przetestuj program dla różnych wartości wejściowych. Co dzieje się dla dużych liczb? Dlaczego? Zmodyfikuj program (w dowolnej z powyższych wersji), by wyświetlał w takiej sytuacji komunikat o błędzie.

*) Pokombinuj jakie są możliwe strategie wzbogacenia programu, by zwiększyć jego zakres działania. Ulepsz go tak, by był w stanie podać 1000! z dokładnością do pierwszych 18 cyfr.

Zadanie 5: Napisz program, który prosi użytkownika o liczbę całkowitą, a następnie zwraca jej reprezentację binarną. Napisz warianty wykorzystujące pętlę oraz rekurencję. Porównaj wynik z postacią zwracaną przez funkcję `std::bitset<liczba_bitow>(zmienna)`, która jest zawarta w nagłówku `<bitset>`. Jak zachowują się liczby ujemne? Dlaczego tak się dzieje – i co zrobić, żeby uzyskać taki sam wynik?