Programowanie Strukturalne - arkusz zadań nr 2

1. Użytkownik podaje trzy liczby całkowite x, y, z - napisać, czy obie z liczb dzielą się przez liczbę z.

2. Rozwinąć poprzednie zadanie - jeśli jedna z liczb, np. x, nie dzieli się równo, napisać na ekranie, o ile trzeba by ją zwiększyć, aby tak było. Tak samo dla y. **Wskazówka:** *wypisać sobie po kolei ile wynosi reszta z dzielenia przykładowego x przez przykładowe z i ile musimy dodać do x, aby uzyskać to co trzeba. Spróbować zapisać to jako równanie i zaimplementować w kodzie.*

3. (**) Inna wersja zadania nr 1&2 - sprawdzić, czy x, y mają taką samą resztę z dzielenia przez z. Jeśli nie, to napisać, o ile trzeba by zwiększyć x, aby tak było. *Dopuszcza się wersję, gdzie "dodajemy" ujemną liczbę do x*.

4. Użytkownik podaje trzy wartości kątów (w stopniach, liczby od 0 do 180 - można zrobić na zmiennych `int`). Sprawdzić, czy możliwy jest trójkąt o podanych kątach (wartości muszą się sumować do `180`) - wyświetlić komunikat na ekranie.

5. Użytkownik podaje wartości kątów dwóch trójkątów. Orzec, czy te dwa trójkąty są podobne. **Rozwinięcie:** wykorzystać wynik poprzedniego zadania i sprawdzić poprawność danych przed testem, zarówno dla trójkąta nr 1 i 2.

    1. Załóżmy dla uproszczenia, że użytkownik podaje wartości kątów α, β, γ od najmniejszego, do największego w sensie $\alpha \leq \beta \leq \gamma$.

    2. Mówimy, że trójkąty są podobne, jeśli po ewentualnym skalowaniu i symetrycznym odbiciu, moglibyśmy idealnie "położyć" jeden trójkąt na drugim, i na płaszczyźnie wszystko by się pokryło. W tym wypadku to po prostu oznacza, że odpowiednie pary kątów 2 trójkątów (od kątów najmniejszych, do największych) muszą być po prostu równe sobie.

6. Użytkownik podaje długości boków trójkąta - najdłuższy x oraz pozostałe y, z. Używając twierdzenia Pitagorasa, określić, czy dany trójkąt jest prostokątny, z założoną dokładnością δ (`delta`) - czyli, czy równość postulowana w twierdzeniu jest spełniona do tej dokładności - tak, gdy $|x^2 - (y^2 + z^2)| < \delta$, gdzie x, y, z to odpowiednie długości boków trojkąta (`float` lub `double`). δ definiujemy jako małą stałą w kodzie np. `1e-30` (1×10^{-30}) . Czyli użytkownik podaje x, y, z, a my wyświetlamy komunikat

    1. **Wskazówka:** przy wyświetlaniu wartości błędu, użyć specyfikatora formatu `%.<SomeNumber>f` (dla float) bądź `%.<SomeNumber>lf`, gdzie `<someNumber>` zastępujemy przyjętą precyzją - przyjąć dość dużą wartość, np. 35 miejsc po przecinku.

    2. Przetestuj poprawność programu na początku dla małego δ np. `1e-30`. Następnie, zwiększ δ do np. wartości `0.001` i spróbuj progresywnie modyfikować jedną/kilka z wartości x, y, z, zaakceptowanych na dokładności `1e-30,` aż do momentu odrzucenia przy mniejszej dokładności.

7. Jak zadanie nr 5, ale użytkownik podaje, dla każdego z trojkątów, długości boków $a \leq b \leq c$. Niech x_1, x_2, x_3 to długości boków trojkąta nr 1 a y_1, y_2, y_3 - trójkąta nr 2 (uporządkowane rosnąco). Wtedy, musimy po prostu sprawdzić, czy $\frac{x_1}{x_2} = \frac{y_1}{y_2}$, $\frac{x_2}{x_3} = \frac{y_2}{y_3}$ oraz $\frac{x_1}{x_3} = \frac{y_1}{y_3}$. *W tym zadaniu, tak jak w zadaniu poprzednim, dobrze by było testować nie **ścisłą** równość, tylko przyjąć jakąś dokładność δ, i testować czy np. $\left|\frac{x_1}{x_2} - \frac{y_1}{y_2}\right| < \delta$ etc.*

8. Użytkownik podaje współrzędne 3 punktów - 3 pary zmiennych `double` bądź `float`. Mamy orzec, czy rysując odcinki pomiędzy tymi trzema punktami, da się z nich zbudować prawilny trójkąt i poinformować o tym na ekranie.

    1. W zadaniu najpierw liczymy długości 3 boków trójkąta na podstawie 3 par współrzędnych punktów.

    2. Potem, sprawdzamy, czy długość najdłuższego boku jest mniejsza lub równa sumie pozostałych boków. Równie dobrze można też sprawdzić warunek $a < b + c$ dla wszystkich możliwych kombinacji 3 boków (wtedy trzeba sprawdzić 3 przypadki - mamy od użytkownika x, y, z, i testujemy każdy wariant podstawienia jako a wartości x, y lub z. *NIe trzeba sprawdzać wyczerpująco 6 kombinacji podstawień, no bo przecież* $b + c = c + b$).

    3. Pomijamy niedokładności numeryczne.

9. (**) Na światłach drogowych stoją 3 samochody. Każdy z nich ma włączony kierunkowskaz i mruga z określoną częstoliwością , np. 1 raz na 3 sekundy. Załóżmy, że obserwujemy sytuację od czasu t_0. Napisać program, który:

1. Pobierze od użytkownika dane na temat każdego samochodu:

    1. Co ile czasu samochód "mruga" (1 mrugnięcie na k sekund, użytkownik podaje k jako liczbę całkowitą).

    2. W której sekundzie, licząc od czasu t_0, samochód "mrugnął" drogowskazem. (liczba całkowita ≥ 0, ale $< k$.) Tą wielkość nazwiemy "fazą" i oznaczymy ϕ (`phi`). Jeśli $\phi = 0$, to zakładamy, że samochód mrugnął w czasie t_0, zatem mrugnie kolejny raz w k-tej sekundzie.

2. Pobierze od użytkownika liczbę sekund t_Δ (`"delta_t` ")- to będzie określenie, ile sekund po t_0 dokonujemu drugiego pomiaru.

3. Ostatecznie określi, czy w chwili $t_0 + t_\Delta$ wszystkie z 3 samochodów mrugną jednocześnie.

(w przypadku najprostszym, gdzie ϕ jest równe zero dla każdego z samochodów - czyli wszystkie samochody "mrugnęły" w czasie t_0, najprościej to zrobić, sprawdzając, czy reszta z dzielenia t_Δ przez okres k wychodzi równa zero, dla każdego z samochodó`w. W przypadku, gdy $\phi > 0$, trzeba się zastanowić, ile ta reszta musi być równa, zebyśmy mogli stwierdzić, że dany samochód mruga w sekundzie t_Δ).

Przykłady danych wejściowych $(k_1, k_2, k_3); (phi_1, phi_2, phi_3); t_{delta}$ vs oczekiwany wynik:

- (3,5,7);(0,0,0), `t_delta=0 lub 105` - mrugną jednocześnie; `t_delta=35` - nie
- (3,6,12);(1,1,1), `t_delta=1 lub t_delta=13 lub t_delta=25...` - mrugną jednocześnie; `t_delta=19 lub 2` - nie
- (4,4,4);(1,2,3) - nie mrugną jednocześnie nigdy, dowolna wartość `t_delta` $\rightarrow$ nie
- (7,7,7);(3,3,3)- dowolne `t_delta` dające resztę z dzielenia przez 7 równą 3, np. 3,10,17 - mrugną jednocześnie
- (4,6,10);(1,3,1)- mrugną jednocześnie pierwszy raz w czasie `t_delta=21`, potem w czasach `81,141,201...`