Wstęp do Informatyki
zadania przygotowawcze z rozwiązaniami

Opisz efekt wykonania podanego kawałka kodu. Odpowiedź uzasadnij.

Uwaga: odpowiedzi nie są pełnym uzasadnieniem poprawności algorytmu. Gdyby chcieć sformułować ścisłe uzasadnienia, należałoby pomyśleć nad niezmiennikami pętli. Z drugiej strony, celem poniższych zadań było raczej zrozumienie niż uzasadnienie.

Przykład 1.

```
// 1 <= k <= n
int tab[n];
int x = 0;
for(int i=0; i<k; i++)
    for(int j=i; j<n; j+=k)
x += tab[j];
// x = ?
```

Po wykonaniu zmienna x jest równa sumie elementów $tab[0], tab[1], \cdots, tab[n-1]$. Wewnętrzna pętla dodaje do x wartości $tab[i], tab[i+k], tab[i+2k], \cdots, tab[i+\lfloor\frac{n-1-i}{k}\rfloor k]$, czyli wszystkie liczby z przedziału 0 do $n-1$, których reszta z dzielenia przez k wynosi i. Jest ona wywoływana dla wartości i od 0 do $k-1$, zatem ostatecznie x jest sumą wszystkich liczb w tablicy.

Przykład 2.

```
void f(int tab[], int n)
{
    for(int i=0; i<n/2; i++)
    {
        int tmp = tab[i];
        tab[i] = tab[n-1-i];
        tab[n-1-i] = tmp;
    }
}

int tab[k];
// tab wypełniamy danymi
//0 <= r <= k

f(tab, k);
f(tab, r);
```

```
f(tab+r, k-r);
```

Funkcja f odwraca tablicę. Kolejno zamienia elementy o indeksach i oraz $n-i-1$ dla i od 0 do $\lfloor \frac{n}{2} \rfloor$.

Trzy wywołania tej funkcji wykonują przesunięcie cykliczne o r miejsc w prawo.

Jeśli na początku w tablicy były liczby

$$1, 2, 3, \cdots, k,$$

to po pierwszym wywołaniu dostaniemy

$$k, k-1, k-2, \cdots, 1,$$

po drugim

$$k-r+1, k-r+2, k-r+3, \cdots, k, k-r, k-r-1, \cdots, 1,$$

a po trzecim:

$$k-r+1, k-r+2, k-r+3, \cdots, k, 1, 2, 3, \cdots, k-r.$$

Przykład 3.

```
int akt = 0;
int skok = 256*128;
//n >= 0
while(skok)
{
    if((akt+skok)*(akt+skok) <= n)
        akt += skok;
    skok /= 2;
}
// n = ?
```

Program oblicza podłogę z pierwiastka z n. Wynik znajduje się w zmiennej *akt*. Końcowa wartość zmiennej *skok* to 0.

Na początku do zmiennej *skok* przypisujemy 2^{15}. Potem w każdym kroku pętli *skok* jest dzielony przez 2, zatem zmienna ta przechodzi przez kolejne potęgi dwójki.

Jeśli $\lfloor \sqrt{n} \rfloor = 2^{p_1} + 2^{p_2} + \cdots + 2^{p_k}$, gdzie $p_1 > p_2 > \cdots > p_k$, to warunek w `if` jest prawdziwy dokładnie wtedy, gdy $skok = 2^{p_i}$ dla pewnego i. Zatem na końcu $akt = \lfloor \sqrt{n} \rfloor$.

Trzeba dodać jeszcze, że maksymalny możliwy do obliczenia wynik to $2^{16}-1$ i właśnie ta liczba trafi do zmiennej *akt* dla $n > (2^{16}-1)^2$.

Przykład 4.

```
for(int i=0; i<n; i++)
    for(int j=0; j<n-i-1; j++)
        if(tab[j] > tab[j+1])
        {
            int tmp = tab[j];
            tab[j] = tab[j+1];
            tab[j+1] = tmp;
        }
```

Ten kod sortuje tablicę w porządku niemalejącym.

Wewnętrzna pętla operuje na elementach od 0 do $n-i-1$ i sprawia, że maksimum spośród nich trafia na pozycję $n-i-1$ (poza tym jeszcze trochę zmienia kolejność elementów, ale to nieistotne). Zatem, po wykonaniu jej dla i od 0 do $n-1$ dostaniemy posortowaną tablicę.

Przykład 5.*

```
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        if(tab[i] < tab[j])
        {
            int tmp = tab[i];
            tab[i] = tab[j];
            tab[j] = tmp;
        }
```

To też jest sortowanie, uzasadnienie jest nieco trudniejsze (omówiliśmy na zajęciach).

Przykład 6.*

```
//int tab[n][n] - tablica z liczbami 0 i 1
for(int x=0; x<n; x++)
    for(int y=0; y<n; y++)
        for(int z=0; z<n; z++)
            if(tab[y][x] && tab[x][z])
                tab[y][z] = 1;
```

Najłatwiej wyjaśnić na przykładzie.

Rysujemy na mapie n miast o numerach od 0 do $n-1$, a tablicę wypełniamy zerami. Teraz połączmy niektóre miasta **jednokierunkowymi** strzałkami. W momencie rysowania strzałki od i do j do $tab[i][j]$ wpisujemy 1.

Po wykonaniu kodu, $tab[k][l]$ będzie równe jeden, jeśli da się dojść z k do l, poruszając się jedynie po strzałkach.