Wydawanie reszty

#testerka #PIWO
#zaokrąglanie_w_górę
#sufit #ceiling
#ceil #floor

Nauka umiejętności programowania wymaga od ucznia dużej pracy samodzielnej. Dlatego też ważne jest, aby rozwiązywać zadania programistyczne i algorytmiczne przy każdej okazji, gdy jest trochę wolnego czasu. Podczas ćwiczeń niezbędne jest korzystanie z jakiegoś narzędzia, które zweryfikuje poprawność rozwiązania, czyli napisanego przez nas programu. Jak już pisaliśmy w podrozdziale *Coś poszło nie tak*, kompilator języka C++ jest dalece niewystarczający na tym polu, choć stanowi pierwsze sito, wyłapując przynajmniej błędy składniowe.

Dlatego też uczący się programiści korzystają z automatycznych *testerek*, które sprawdzają poprawność rozwiązania problemu na zestawie specjalnie dobranych testów, ułożonych przez autora zadania. Istnieje w sieci kilka takich bezpłatnych serwisów (są i komercyjne) – do najpopularniejszych należą międzynarodowy Codeforces (`codeforces.com`) oraz krajowy Szkopuł (`szkopul.edu.pl`). W szczególności na Szkopule zamieszczony jest zbiór zadań **PIWO 2018/19**[*] zredagowany przez Kocurra, zawierający sporą ilość zadań, w większości łatwych, których pomysły często zaczerpnięte były z Codeforces. Każdy z Was może uzyskać dostęp do tego zbioru, zakładając sobie konto na Szkopule, a link aktywacyjny będzie można znaleźć na dole głównej strony *Kodowania z Kocurrem* (już niedługo).

W tym podrozdziale zaprezentujemy przykładowe zadanie *Wydawanie reszty*, którego treść możecie znaleźć tutaj. Oryginalne zadanie to problem 1061A z serwisu Codeforces, natomiast jego polska treść pochodzi od Kocurra.

Jest to zadanie z kategorii „obelżywie łatwe", ale pokazuje ono, jak wyglądają typowe zadania ze zbiorów problemów towarzyszących testerkom. Główną treść zadania stanowi tak zwana „bajka", tutaj: historia Janusza pracującego na kasie w Biedronce. W zadaniu opisane są szczegółowo *dane wejściowe*, czyli co program musi przeczytać, oraz *wynik programu*, czyli co program powinien wypisać.

W tym przykładzie chodzi o wydanie reszty w sposób optymalny, a więc przy użyciu minimalnej ilości monet. Janusz ma do dyspozycji dowolną ilość monet o nominałach 1, 2, ..., n, gdzie n jest dodatnią liczbą całkowitą z zakresu od 1 do 10^5. Kwota reszty, oznaczana tu przez R, jest dodatnią liczbą całkowitą z zakresu od 1 do 10^9. Dane wejściowe to dwie wymienione liczby, oddzielone pojedynczym odstępem.

[*]PIWO, czyli Piwniczne Informatyczne Warsztaty Olimpijskie, to edukacyjne przedsięwzięcie Kocurra, obchodzące w tym roku (2019) okrągły jubileusz 20-lecia.

Zatem nasz program powinien zaczynać się od zadeklarowania odpowiednich zmiennych i wczytania ich wartości:

```
int n, R;
cin >> n >> R;
```

Ile monet potrzebuje Janusz, by wydać resztę? Na oko widać, że opłaca mu się użyć przede wszystkim monet o najwyższym nominale, czyli n. Jeśli R dzieli się bez reszty przez n, to sprawa jest prosta: potrzeba R/n monet. Na przykład dla $n = 4$ oraz $R = 20$ wynik programu to $5 = 20/4$. Jeśli jednak R nie dzieli się bez reszty przez n, wtedy oprócz tego największego nominału trzeba użyć jeszcze jednej, mniejszej monety, aby dopełnić całą kwotę. Wartość tej monety to reszta z dzielenia R przez n, a zatem jej wartość będzie mniejsza od n (dlaczego?).

Możemy zatem naszkicować cały program z użyciem instrukcji warunkowej, która sprawdzi, czy ta dodatkowa moneta jest potrzebna (i w takim razie doliczy ją do wyniku programu):

```
#include <iostream>
using namespace std;

int main()
{
  int n, R;
  cin >> n >> R;
  if(R % n == 0)
    cout << R / n << endl;
  else
    cout << R / n + 1 << endl;
}
```

Pamiętamy oczywiście, że w języku C++ dzielenie R/n jest dzieleniem całkowitym (z odrzuceniem części ułamkowej).

Zauważmy, że zestaw monet użytych do wydania reszty nie jest określony jednoznacznie (w tym drugim przypadku, gdy potrzebna jest dodatkowa moneta). Na przykład dla $n = 5$ i $R = 18$ możliwe zestawy monet to $\{5, 5, 5, 3\}$ lub $\{5, 5, 4, 4\}$. Jednak minimalna ilość potrzebnych monet jest określona jednoznacznie (4), a o taką wartość chodzi właśnie w zadaniu.

Każde zadanie należy przede wszystkim przetestować na przykładowych danych z treści zadania, uruchamiamy więc nasz program i wpisujemy:

```
5 11
```

Otrzymujemy wynik 3, więc wygląda OK. Jeszcze sprawdzimy drugi wariant:

```
4 20
```

Otrzymujemy 5, więc też OK. Możemy zatem spróbować wysłać nasz program na serwis Szkopuł, na wspomniany konkurs PIWO 2018/19. Jeśli otrzymamy wynik 100 punktów (wyświetlony na zielono), to jest dobrze.

Mamy jednak propozycję innego sposobu obliczenia wyniku. Pamiętasz, jak w podrozdziale *Ilość cyfr* pisaliśmy o funkcji *podłoga* (ang. **floor**)? Ta funkcja powodowała obcięcie części ułamkowej (jeśli takowa występowała), czyli zaokrąglenie w dół. Dla przykładu mamy:

$$\left\lfloor \frac{9}{4} \right\rfloor = 2, \quad \left\lfloor \frac{9}{3} \right\rfloor = 3, \quad \left\lfloor \frac{4}{9} \right\rfloor = 0.$$

Otóż w matematyce mamy „do pary" również funkcję *sufit* (ang. **ceiling**, wymawiaj: *siling*, z twardym **s**), oznaczaną symbolem $\left\lceil \frac{a}{b} \right\rceil$. Ta funkcja z kolei powoduje dopełnienie liczby z częścią ułamkową do najbliższej większej liczby całkowitej, czyli zaokrąglenie w górę. (Sufit z liczby całkowitej jest równy jej samej.) Mamy zatem:

$$\left\lceil \frac{9}{4} \right\rceil = 3, \quad \left\lceil \frac{9}{3} \right\rceil = 3, \quad \left\lceil \frac{4}{9} \right\rceil = 1.$$

W języku C++ istnieją gotowe funkcje `floor()` oraz `ceil()`, ale dziedziną i zbiorem wartości tych funkcji jest zmiennoprzecinkowy typ danych `double` – podobnie jak wspomnianej w podrozdziale *Kocur do kwadratu* funkcji `pow()`, obliczającej potęgę liczby.

Jeżeli zamierzamy obliczyć wartość wyrażenia $\left\lfloor \frac{a}{b} \right\rfloor$ dla dodatnich liczb całkowitych a oraz b, wtedy wystarczy (w języku C++) napisać wyrażenie a/b i otrzymamy właśnie zaokrąglenie w dół. Natomiast, aby obliczyć $\left\lceil \frac{a}{b} \right\rceil$ trzeba uciec się do pewnej sztuczki arytmetycznej. Trzeba mianowicie do dzielnej a dodać dzielnik b pomniejszony o 1. Mamy bowiem tożsamość:

$$\left\lfloor \frac{a+b-1}{b} \right\rfloor = \left\lceil \frac{a}{b} \right\rceil.$$

Powyższa równość nie powinna nas specjalnie dziwić. Jeśli bowiem a dzieli się bez reszty przez b, wtedy zarówno sufit, jak i podłoga dają ten sam wynik – jest to po prostu iloraz a/b i dodanie $b-1$ do dzielnej nie zmieni tego wyniku. Jeśli natomiast reszta z dzielenia a przez b jest dodatnia, to dodanie $b-1$ do dzielnej spowoduje zwiększenie wyniku o 1, czyli o tyle, ile trzeba (sprawdź!).

Dla przykładowych danych z treści zadania ($n = 5$ oraz $R = 11$) mamy:

$$\left\lceil \frac{R}{n} \right\rceil = \left\lceil \frac{11}{5} \right\rceil = \left\lfloor \frac{11+5-1}{5} \right\rfloor = \left\lfloor \frac{15}{5} \right\rfloor = 3.$$

Tak więc możemy nasz program zapisać w elegancki, bardziej zwarty sposób:

```
#include <iostream>
using namespace std;

int main()
{
  int n, R;
  cin >> n >> R;
  cout << (R + n - 1) / n << endl;
}
```

Wynik programu będzie oczywiście taki sam, jak w jego poprzedniej wersji.