Czysta doskonałość

#liczba_doskonała
#reszta_z_dzielenia
#porównanie

Ten przykład poświęcimy sprawdzaniu, czy dana liczba naturalna jest *liczbą doskonałą*. Aby zasłużyć na takie zaszczytne miano, liczba powinna być równa sumie swoich podzielników właściwych, czyli mniejszych od niej samej.*

Na przykład liczba 6 spełnia ten warunek, gdyż jej podzielnikami właściwymi są liczby 1, 2 oraz 3, i mamy $6 = 1 + 2 + 3$. Jest więcej takich liczb (być może nawet nieskończenie wiele), ale w zakresie wartości typu całkowitego `int` – czyli do dwóch miliardów – jest ich zaledwie pięć. (Jakie to liczby?)

Napiszemy program, który będzie wczytywał liczbę naturalną n i wypisywał komunikat TAK/NIE w zależności od tego, czy n jest liczbą doskonałą.

Musimy zatem znaleźć wszystkie podzielniki właściwe liczby n i sprawdzić, czy ich suma jest równa samej liczbie n. Zrobimy to metodą „brutalnej siły” (ang. **brute force**, wymawiaj: *brut fors*), sprawdzając wszystkie potencjalne podzielniki p od 1 do $n - 1$ włącznie. Po czym rozpoznamy, że dana liczba p jest podzielnikiem liczby n? Ano po tym, że reszta z dzielenia n przez p jest równa zero. W C++ zapisujemy takie zdanie logiczne w następujący sposób:

```
n % p == 0
```

Symbol `%` oznacza *resztę z dzielenia*, na przykład `20%6` jest równe 2, a `30%5` jest równe 0.

Symbol `==` to *operator porównania*, czyli na przykład „`A==B`” jest zdaniem logicznym prawdziwym, jeśli A jest równe B.

Brakuje nam jeszcze pomocniczej zmiennej, w której będziemy zbierać sumę dotychczas znalezionych podzielników – nazwijmy ją więc *sum* (z angielska, wymawiaj: *sam*) i dajmy na początku wartość zero.

Program oparty będzie na pętli† po zmiennej p: musi ona kolejno przybrać wartości od 1 do $n - 1$. Jeśli dana wartość będzie nam pasować (to znaczy jeśli będzie podzielnikiem liczby n), wtedy powiększymy o nią wartość zmiennej *sum*. Na końcu programu sprawdzimy, czy *sum* jest równa n i wypiszemy odpowiedni komunikat.

*O liczbach doskonałych wspominał już Euklides w swoim monumentalnym dziele *Elementy* z IV w. p.n.e.

†Błagam, by nie mówić „o pętlę”, jak to się nieraz w mediach słyszy. Oprzeć „o” to można (się) o drzewo, gdy wiatr wieje.

Oto i cały program:

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

int main()
{
  int n;
  cin >> n;
  int p = 1, sum = 0;
  while(p < n)
  {
      if(n % p == 0)
        sum = sum + p;
      p++;
  }
  if(sum == n)
    cout << "TAK" << endl;
  else
    cout << "NIE" << endl;
}
```

Zdanie logiczne „`p < n`” jest równoważne zdaniu „`p <= n - 1`”. Póki ten warunek jest spełniony, póty pętla `while` jest wykonywana.

Zwróćmy uwagę na instrukcję „`sum = sum + p;`”, która oznacza „powiększ wartość zmiennej *sum* o wartość zmiennej *p*”. Pojedynczy znak równości `=` oznacza nadanie wartości, a nie równość w sensie algebraicznym.

Oczywiście w pętli mamy także znaną nam już instrukcję powiększenia o jeden, czyli „`p++;`”.

Po uruchomieniu programu wpisujemy jakąś liczbę naturalną, a program wypisuje TAK lub NIE. Na przykład liczba 28 jest liczbą doskonałą:

```
28
TAK
```

natomiast liczba 100 nie ma tej własności:

```
100
NIE
```