Na drobne kawałki

#liczba_pierwsza
#rozkład_na_czynniki

Każdą dodatnią liczbę całkowitą większą od 1 można rozłożyć na czynniki pierwsze, czyli na iloczyn takich liczb naturalnych, których już bardziej rozłożyć się nie da. Przypominamy, że *liczba pierwsza* to taka liczba naturalna, która ma tylko dwa podzielniki: liczbę 1 oraz nią samą. Liczby 0 oraz 1 nie są liczbami pierwszymi (dlaczego?).

Weźmy dla przykładu taką budzącą respekt liczbę 113400 i spróbujmy rozłożyć ją na czynniki pierwsze. Na końcu liczby widzimy zero, więc na pewno dzieli się ona przez 2 i przez 5. Zarówno 2 jak i 5 są liczbami pierwszymi, więc nam pasują. Końcowych zer mamy dwa, więc na pewno możemy tak zapisać:

$$113400 = 1134 \cdot 2^2 \cdot 5^2.$$

Do rozłożenia została liczba 1134: jest ona parzysta, więc na pewno dzieli się przez 2:

$$1134 = 567 \cdot 2.$$

Liczba 567 nie wyglada zbyt przyjaźnie, ale jej suma cyfr jest podzielna przez 3, więc możemy spróbować (nawet kilka razy):

$$567 = 189 \cdot 3 = 63 \cdot 3^2 = 21 \cdot 3^3 = 7 \cdot 3^4.$$

Dalej nie pójdzie, ale i nie ma takiej potrzeby, bo 7 jest liczbą pierwszą. Jeśli pozbieramy *zusammen do kupy*[*] wszystkie otrzymane wyżej czynniki, otrzymamy następujący rozkład:

$$113400 = 2^3 \cdot 3^4 \cdot 5^2 \cdot 7.$$

Oczywiście, kto był cwany, ten wpisał liczbę 113400 w okienku serwisu WolframAlpha[†] (`www.wolframalpha.com`) i dostał od razu wynik:

[*]Wymawiaj: *cuzamen*, to po germańsku.
[†]Znakomite narzędzie, gorąco polecam, wszystko tam da się rozwiązać!

Angielskie słowo **prime** (wymawiaj: *prajm*) oznacza *liczbę pierwszą*, zaś słowo **factorization** (wymawiaj: *faktorizejszyn*) to *rozkład na czynniki*.

My pokusimy się o samodzielne napisanie programu, który dokonuje takiej sztuki. Trzeba to jakoś sprytnie zrobić, tak aby nie trzeba było się trudzić sprawdzaniem, czy kolejny czynnik jest liczbą pierwszą, tylko żeby jakoś to tak wyszło „siłami natury". Aha, i jeszcze nie będziemy się silić, aby dostać taki ładny rozkład z wykładnikami potęgowymi (zostawimy to jako ćwiczenie dla starszaków). Zadowolimy się programem, który po wprowadzeniu przykładowej liczby:

```
113400
```

wypisze na ekranie:

```
2 2 2 3 3 3 3 5 5 7
```

Na początek wystarczy. Nie od razu Nową Hutę zbudowano!

Od jakiego czynnika warto zacząć? Od najmniejszej liczby pierwszej, czyli od 2.[‡] Zaczynamy więc (podzielnik oznaczamy przez p, rozkładaną liczbę przez n):

```
int n;
cin >> n;
int p{ 2 };
```

Teraz będzie pętla wyglądająca z grubsza tak: jeśli dane p jest istotnie podzielnikiem, wtedy n dzielimy przez niego, a jeśli nie – to przechodzimy do następnego p (czyżby `p++` wystarczyło?). I tak w kółko, aż do końca (zdefiniujemy go za chwilę).

Oczywiście pasujący podzielnik p wyświetlamy na ekranie (aby się nim nacieszyć), a podzielenie n przez p skutkuje usunięciem tego podzielnika z liczby n, więc możemy zająć się szukaniem następnych podzielników. Pętla mogłaby więc tak się prezentować (trzy kropki to warunek kontynuacji, do uzupełnienia):

```
while(...)
  if(n % p == 0)
  {
    cout << p << ' ';
    n /= p;
  }
  else
    p++;
```

„Być albo nie być podzielnikiem" rozstrzygamy przez badanie reszty z dzielenia n przez p. Po każdym znalezionym i wyświetlonym podzielniku wyświetlamy odstęp (`' '`). Pamiętamy oczywiście, że instrukcja:

```
n /= p;
```

[‡]Warto przy okazji zauważyć, że jest to jedyna liczba pierwsza, która jest parzysta.

znaczy dokładnie to samo, co:

```
n = n / p;
```

Jeśli p nam odpowiada jako podzielnik, to pozostajemy przy nim, dzieląc n do spodu, póki się da.

Jednak czy takie proste „`p++`” wystarczy do przejścia do następnej sensownej wartości p? OK, od wartości 2 przejdziemy do wartości 3 (także liczby pierwszej). Ale po trójce mamy wartość $p = 4$, a to nie jest liczba pierwsza! Czy to jest problem? Odpowiedź brzmi: nie, ponieważ potencjalny podzielnik równy 4 nie ma szans załapać się w tym momencie, bo przecież wcześniej usunęliśmy wszystkie podzielniki o wartości 2, a więc – potencjalnie ich wielokrotności i potęgi.

Tak więc nasza pętla sprytnie przeskoczy nad wartością 4, nie zaprzątając sobie nią uwagi i szybko przejdzie do $p = 5$, które ma jakie takie szanse na sukces. I tak dalej, i tak dalej. Ten prosty schemat działa, gdyż na przykład przy $p = 9$ będziemy mieć podobny problem, ale już wcześniej usunęliśmy podzielnik 3 (w dowolnej potędze), więc liczba 9 nie ma szans. Tylko liczby pierwsze mogą zostać zaliczone jako podzielniki liczby n, choć oczywiście muszą wylegitymować się resztą z dzielenia równą 0.[§]

No to co nam jeszcze zostało, aby móc dumnie wstać z kolan i zaprezentować nasz program? Musimy sprecyzować warunek kontynuacji wspomnianej pętli `while`. Tutaj sporo zależy od tego, o jakim zakresie wartości n mówimy. Jeśli ograniczamy się do n nieprzekraczającego miliona (lub kilku), wtedy możemy spokojnie liczyć *na pałę*, czyli bez zbytniej finezji. Każdy znaleziony podzielnik zmniejsza nam wartość zmiennej n, zatem po znalezieniu wszystkich podzielników otrzymamy $n = 1$. Zatem zaprzeczenie tego stwierdzenia powinno być warunkiem kontynuacji pętli. Oto i cały program (dodaliśmy finalne wypisywanie końca wiersza `'\n'`):

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int n;
  cin >> n;
  int p{ 2 };
  while(n > 1)
    if(n % p == 0)
    {
      cout << p << ' ';
      n /= p;
    }
    else
      p++;
  cout << '\n';
```

[§]Ten chytry sposób wybierania tylko liczb pierwszych przez eliminację wielokrotności podzielników nosi nazwę *sita Eratostenesa*. Eratostenes żył w III wieku p.n.e., urodził się w Cyrenie (jak Szymon, z Drogi Krzyżowej), ale pracował głównie w Aleksandrii. Wyznaczył przybliżony obwód Ziemi oraz odległość Ziemi od Księżyca i Słońca. Dobry był, skubany! (Zwolennicy płaskiej Ziemi serdecznie go nienawidzą.)

```
  return 0;
}
```

Większe n wymaga głębszego pomyślunku, ale to może za chwilę.