Rok przestępny

#alternatywa
#inkrementacja
#dekrementacja

Kalendarz gregoriański* jest obecnie powszechnie używanym kalendarzem, choć w użyciu są również inne kalendarze, jak na przykład prawosławny kalendarz liturgiczny. Wszyscy mniej więcej wiedzą, na czym polega specyfika naszego kalendarza: niektóre lata są przestępne (ang. **leap year**, wymawiaj: *lip jer*, z długim *i*) i wtedy w lutym mamy jeden dodatkowy dzień. Tylko który rok jest przestępny? Co czwarty? No, w zasadzie tak – ale sytuacja jest nieco bardziej skomplikowana.

Przed kalendarzem gregoriańskim używano kalendarza juliańskiego,† w którym istotnie co czwarty rok był o dzień dłuższy. Dawało to średnią długość roku równą 365,25 dnia, co jednak niezbyt zgadzało się z dokładnymi obserwacjami astronomicznymi, wedle których średnia długość roku powinna wynosi około 365,2422 dnia. Kalendarz juliański generował zatem odstępstwo od rachuby astronomicznej: 1 dzień na około 128 lat. W XVI wieku ta różnica wynosiła już 10 dni i trzeba było coś z tym zrobić.

Wprowadzono więc bardziej złożoną regułę „przestępności"roku: rok jest przestępny, jeśli dzieli się przez 4, natomiast jeśli dzieli się przez 100, wtedy nie jest przestępny – no, chyba że dzieli się przez 400. Tak więc na przykład rok 2000 był przestępny, a rok 1900 – nie (mimo że dzieli się przez 4). Uff! Ogarnęliście to?

Taki sposób wyboru roku przestępnego – mimo swej zawiłości – ma jednak niezaprzeczalną zaletę: daje on średnią długość roku równą 365,2425 dnia, co różni się od roku astronomicznego zaledwie o 0,0003 dnia. To znaczy, że przez jakieś 3000 lat mamy spokój z korektą kalendarza.

Napiszemy program, który wczyta pewną ilość lat (dodatnich liczb całkowitych) i dla każdego roku wypisze komunikat TAK/NIE w zależności od tego, czy jest on przestępny. Liczba lat (zmienna N) będzie pierwszą daną wczytaną przez nasz program. Wczytaniem kolejnego roku, sprawdzeniem jego przestępności i wypisaniem komunikatu zajmie się funkcja `leap_year()`.

Zaczniemy od funkcji `main()`. Pierwsza przymiarka może wyglądać tak:

```
int main()
{
  int N;
  cin >> N;
```

*Wprowadzony przez papieża Grzegorza XIII w 1582 r.
†Wprowadzonego przez Juliusza Cezara w 45 r. p. n. e.

```
  while(N > 0)
  {
    leap_year();
    N--;
  }
}
```

Jest OK, ale można to zgrabniej zapisać. Otóż jeśli posługujemy się operatorem zwiększania (inkrementacji) `++` lub zmniejszania (dekrementacji) `--`, wtedy możemy go zapisać *przed* zmienną lub *po* niej. Jeśli jest to samodzielna instrukcja, wówczas jest to wszystko jedno. Rezultatem poniższego kodu jest nadanie wartości 6 zmiennej a:

```
int a = 5;
a++;
```

W wyniku poniższego kodu zmienna a również otrzyma wartość 6:

```
int a = 5;
++a;
```

Warto jednak wiedzieć, że instrukcje `a++` oraz `++a` mają wartość liczbową: w pierwszym przypadku jest to wartość zmiennej a przed zwiększeniem, a w drugim – po zwiększeniu. To samo obowiązuje dla operatora dekrementacji `--`, co pozwala nam wstawić operację zmniejszania zmiennej N bezpośrednio do warunku kontynuacji pętli `while` i zredukowanie treści pętli do jednej instrukcji:

```
while(N-- > 0)
  leap_year();
```

Mało tego, możemy nawet usunąć porównanie z zerem – rzecz w tym, iż wartość liczbowa 0 jest traktowana jako fałsz, zaś jakakolwiek inna – jako prawda. Zatem naszą pętlę zapiszemy tak:

```
while(N--)
  leap_year();
```

Teraz czas na funkcję `leap_year()`: najpierw oczywiście wczytujemy zmienną *year*. Od czego rozpocząć sprawdzanie? Od tego warunku, którego spełnienie gwarantuje przestępność roku, czyli:

```
year % 400 == 0
```

Jeśli to nie wypali, to jeszcze musimy sprawdzić, czy rok dzieli się przez 4, ale nie dzieli się przez 100:

```
year % 4 == 0 && year % 100 > 0
```

A jaki spójnik logiczny postawić pomiędzy tymi warunkami? Wystarczy, że spełniony jest jeden z nich,[‡] zatem powinien to być spójnik „lub" (ang. **or**), oznaczający *alternatywę*. Symbol reprezentujący to działanie ma postać `||`.

Kompletny program realizujący rozwiązanie postawionego problemu ma postać:

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

void leap_year()
{
  int year;
  cin >> year;
  if(year % 400 == 0 || (year % 4 == 0 && year % 100 > 0))
    cout << "TAK" << endl;
  else
    cout << "NIE" << endl;
}

int main()
{
  int N;
  cin >> N;
  while(N--)
    leap_year();
}
```

Ponieważ mamy do czynienia ze zdaniem logicznym złożonym, z dwoma różnymi spójnikami logicznymi, więc drugi warunek został umieszczony w nawiasach.[§]

Po uruchomieniu programu i wpisaniu przykładowych danych:

```
4
1900
2000
2019
2020
```

otrzymamy rezultat:

```
NIE
TAK
NIE
TAK
```

[‡]W tym przypadku trudno byłoby spełnić oba (dlaczego?).

[§]Tak naprawdę te nawiasy nie są konieczne, gdyż koniunkcja (`&&`) ma wyższy priorytet niż alternatywa (`||`), ale tak będzie bardziej czytelnie.