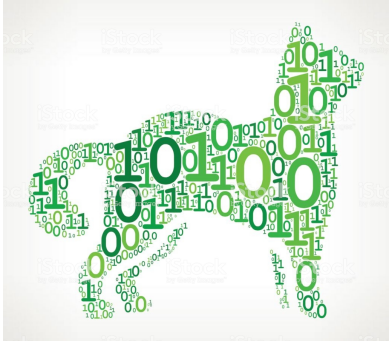
Kot dwójkowy



#układ_pozycyjny
#układ_dwójkowy
#char

W informatyce wielką karierę zrobił dwójkowy układ pozycyjny (ang. **binary system**, wymawiaj: *bajnary system*, z akcentem na pierwsze *a*) – ze względu na fakt, że w elektronice cyfrowej wygodnie jest operować dwoma stanami: *włączony/wyłączony*. W tym układzie mamy tylko dwie cyfry: 0 oraz 1. Dla przykładu liczba dwójkowa 11010 ma następującą wartość dziesiętną:

$$(11010)_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 16 + 8 + 2 = 26.$$

Napiszemy teraz funkcję `binary()`, która będzie obliczać dwójkową postać liczby naturalnej. Argument tej funkcji będzie typu całkowitego (`int`), ale jej rezultat powinien raczej być typu `string`, gdyż liczby w postaci dwójkowej są na ogół dłuższe od postaci dziesiętnej (dlaczego?). Zatem nagłówek funkcji powinien wyglądać tak:

```
string binary(int n)
```

Obliczanie kolejnych cyfr postaci dwójkowej powinniśmy zacząć od ostatniej cyfry, gdyż jest to po prostu reszta z dzielenia n przez 2. Teraz możemy podzielić n przez 2 (odrzucając ewentualną część ułamkową) i znów zapytać o resztę z dzielenia przez 2 – to będzie druga od końca cyfra. Potem znowu dzielimy przez 2, obliczamy resztę – i tak dalej, aż w końcu cyfry nam się skończą i n osiągnie wartość 0.

Musimy jednak pamiętać, że kolejne cyfry dwójkowe dostajemy w kolejności *od tylca*, zatem musimy doklejać je do wynikowej liczby (zmienna w) w odpowiedni sposób: na początku (przerabialiśmy to już, bodajże omawiając palindromy).

Oto jak mogłaby wyglądać nasza funkcja:

```
string binary(int n)
{
  string w;
  while(n > 0)
  {
    if(n % 2 > 0)
      w = '1' + w;
    else
      w = '0' + w;
```

```
    n = n / 2;
  }
  return w;
}
```

Pamiętamy, że znak '0' (typu `char`) to nie to samo, co liczba 0 (typu `int`). W pojedynczych apostrofach zapisujemy właśnie stałe typu znakowego.

Funkcja wygląda całkiem fajnie, ale ma jeden feler: dla $n = 0$ zwraca pusty ciąg znaków. Łatwo temu zaradzić: wystarczy zastąpić pętlę `while` przez pętlę `do-while`, która musi obejść przynajmniej jeden raz.

Poza tym możemy pozbyć się instrukcji warunkowej, jeśli wykorzystamy fakt, że wyrażenie `char(k)` oznacza znak o kodzie ASCII k. Nie warto uczyć się kodów na pamięć, ale łatwo sprawdzić na przykład na Wikipedii, że kod ASCII znaku `'0'` wynosi 48. Tak więc wyrażenie `char(n%2+48)` to odpowiednio znak '0' lub '1' w zależności od wartości reszty z dzielenia n przez 2.*

Ostateczna postać naszej funkcji wygląda tak:

```
string binary(int n)
{
  string w;
  do
  {
    w = char(n % 2 + 48) + w;
    n = n / 2;
  }
  while(n > 0);
  return w;
}
```

Możemy przetestować ją na przykład takim oto programem:†

```
int main()
{
  int n;
  cin >> n;
  cout << binary(n) << '\n';
  return 0;
}
```

Na przykład po wpisaniu liczby 26 otrzymujemy wynik 11010.

*Równie dobrze moglibyśmy napisać `char(n%2+'0')`, ponieważ jeśli znaki występują w kontekście arytmetycznym (czyli w wyrażeniu), wtedy zamieniane są na ich kod ASCII.

†Program należy oczywiście uzupełnić odpowiednim nagłówkiem i wstawić definicję funkcji `binary()` przed funkcję `main()`.