Liczba czy ciąg znaków

#zamiana_int_na_string
#zamiana_string_na_int
#for-each

Kiedy jakiś program wyświetla na ekranie liczbę naturalną, na przykład:

```
456
```

nie jesteśmy w stanie odróżnić, czy ta wartość wewnątrz programu egzystowała jako liczba całkowita, czy jako ciąg znaków. Poniższy kod:

```
int n{456};
cout << n << '\n';
```

daje taki sam rezultat – na ekranie – jak następujący fragment programu:

```
string s{"456"};
cout << s << '\n';
```

Jednak w kodzie programu to rozróżnienie jest zwykle bardzo istotne. Jeśli na danej zmiennej zamierzamy wykonywać operacje arytmetyczne (choćby powiększenie wartości o 1), wtedy należy wybrać typ liczbowy – na przykład `int`. Jeśli z kolei chcemy wykonywać operacje typowo ciągowoznakowe (na przykład wyszukiwanie fragmentu lub łączenie/sklejanie dwóch liczb w jedną), wtedy użycie typu `string` może być korzystniejsze.*

Tak czy owak, warto zaimplementować funkcje `int2str()` oraz `str2int()` umożliwiające zamianę typu liczbowego na `string` i na odwrót.†

Funkcja `int2str()`

Zaczniemy od funkcji zamieniającej liczbę naturalną na ciąg znaków. Nazwa funkcji – `int2str` – to skrót od angielskiego zwrotu **integer to string** (wymawiaj: *intidżer tu string*), czyli *liczba całkowita na string*). Wykorzystuje się tutaj jednakową (prawie) wymowę angielskiego słowa **to** oraz angielskiego liczebnika 2 (**two**).

*Wykonywanie operacji typowo arytmetycznych na liczbach reprezentowanych przez zmienne typu `string` jest obszernie omówione w *Siódmych miaukotach*.

†W języku C++ istnieją gotowe mechanizmy takiej zamiany, ale chyba warto w ramach ćwiczeń zrobić to samodzielnie – może się to przydać na przykład na maturze z informatyki.

Metoda będzie podobna do tej, którą zastosowaliśmy w podrozdziale *Kot dwójkowy*, czyli obliczanie reszty z dzielenia przez 10 (to będzie ostatnia cyfra), a następnie dzielenie przez 10. I tak w kółko, aż nam się skończą cyfry konwertowanej liczby. Przypadek zera należy wyifować i obsłużyć na początku funkcji.

Nie ma co dużo gadać, napiszmy naszą funkcję:

```
string int2str(int n)
{
  if(n == 0)
    return "0";
  string w;
  while(n > 0)
  {
    w = char(n % 10 + '0') + w;
    n = n / 10;
  }
  return w;
}
```

Program testujący tę funkcję może wyglądać na przykład tak:

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

string int2str(int n)
{
  . . .
}

int main()
{
  int n;
  cin >> n;
  cout << int2str(n) << '\n';
  return 0;
}
```

Po uruchomieniu programu i wpisaniu dowolnej liczby naturalnej (mieszczącej się w zakresie typu `int`) jako rezultat programu dostaniemy tę samą liczbę, jednak będzie ona wypisana jako `string`.

Funkcja `str2int()`

Czas na implementację funkcji działającej odwrotnie – posłużymy się schematem Hornera opisanym w podrozdziale *Kot dziesiętny*. Właściwie nowość polega tylko na tym, że podstawą układu pozycyjnego będzie w tym przypadku liczba 10 (zresztą tak samo, jak w podrozdziale *Liczbowe palindromy*).

Kompletny program testujący rozważaną funkcję ma taką oto przykładową postać:

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

int str2int(string s)
{
  int w{ };
  for(char c : s)
    w = w * 10 + c - '0';
  return w;
}

int main()
{
  string s;
  cin >> s;
  cout << str2int(s) << '\n';
  return 0;
}
```

I znów, po uruchomieniu tego programu i wpisaniu ciągu znaków reprezentującego liczbę naturalną (nie za długą), na ekranie ponownie pojawi się ta sama liczba wypisana jako wartość typu `int`.

W funkcji `str2int()` zastosowaliśmy petlę **for-each** (*dla każdego*) – zmienna c typu znakowego (`char`) reprezentuje kolejno wszystkie elementy ciągu znaków s.