Od przodu i od tyłu

#ciąg_znaków #string
#znak #char
#palindrom

Co jest szczególnego w wyrazach takich, jak **kajak** lub **oko**? Czytane wspak wyglądają tak samo – takie wyrażenia nazywamy *palindromami* (z greckiego: **palindromeo** = *biec z powrotem*). Palindromy traktowane są jako zabawy słowne, choć niewykluczone, że ongiś przypisywano im znaczenie magiczne.

Napiszemy teraz program, który przeczyta wyraz (ciąg liter) i odpowie na pytanie, czy jest to palindrom.

Do przechowania wyrazu nadaje się typ danych `string`, w którym możemy przechowywać dowolny ciąg znaków – my ograniczymy się do małych liter alfabetu łacińskiego. Takie zmienne deklaruje się i wczytuje podobnie, jak używane przez nas do tej pory zmienne całkowite, na przykład:

```
string s;
cin >> s;
```

Ciąg znaków s to tabelka zawierająca poszczególne jego znaki. Elementy tej tabelki mają swoje numery: 0, 1, 2, ..., $d-1$, gdzie d oznacza *długość* ciągu znaków, czyli ilość znaków w ciągu. Na przykład ciąg znaków `"wombat"` ma długość 6, a ciąg pusty `""` ma długość 0, bo nie zawiera żadnych znaków. Długość ciągu znaków jest zwracana przez funkcję `length()`, na przykład:*

```
int d = s.length();
```

Zauważ, że funkcja `length()` działa na zmienną s przy użyciu operatora kropki `.` – dzieje się tak dlatego, że `string` nie jest zwykłym typem danych, takim jak `int`, ale jest *typem obiektowym*. Dokładniej, nazwa `string` oznacza *klasę*, czyli wzorzec dla tworzenia *obiektów*. W naszym przykładzie takim obiektem jest s.†

Poszczególne elementy ciągu znaków (w naszym przykładzie $s[0], s[1], s[2], \ldots$) reprezentują *typ znakowy* (ang. **character**, w skrócie **char** – wymawiaj: *karakter*, *kar*).

Jednym ze sposobów sprawdzenia, czy wprowadzony wyraz jest palindromem, jest utworzenie drugiego wyrazu o odwrotnej kolejności znaków. Jeśli obydwa wyrazy będą równe, wtedy wpisany wyraz jest palindromem.

*W tym samym celu można także użyć funkcji `size()` – te dwie funkcje to synonimy.

†Programowanie obiektowe to bardzo szeroki temat i w niniejszym kursie dla początkujących pojawią się tylko niewielkie wzmianki na ten temat.

Jak wygenerować odwrócony ciąg znaków? Wystarczy znak po znaku ustawić w odwrotnej kolejności. Poniższa pętla wykonuje właśnie to zadanie (odwrócony ciąg znaków umieszczony jest w zmiennej t):

```
string t = "";
int i{ };
while(i < s.length())
{
  t = s[i] + t;
  i++;
}
```

Na początku zmienna t to pusty ciąg znaków,[‡] a zmienna i ma wartość zero. Do ciągu t doklejamy kolejne znaki z ciągu s tak, aby ułożyły się w odwrotnej kolejności. Dlatego właśnie mamy tu instrukcję `t = s[i] + t`, która oznacza:

> *Weź* `i`*-ty znak z ciągu* s*, doklej go przed dotychczasowym ciągiem* t*, po czym podstaw wynik za* t*.*

Operator plus (`+`) w tym kontekście oznacza właśnie sklejanie (łączenie) ciągów znaków.[§]

Pozostaje tylko porównanie ciągów s oraz t – jeśli są takie same, to wprowadzony wyraz s jest palindromem.

Oto i cały program:

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

int main()
{
  string s;
  cin >> s;
  string t = "";
  int i{ };
  while(i < s.length())
  {
      t = s[i] + t;
      i++;
  }
  if(s == t)
    cout << "TAK\n";
  else
    cout << "NIE\n";
  return 0;
}
```

[‡]W praktyce wystarczy napisać `string t;`, gdyż zmienne typu `string` są automatyczne inicjalizowane – przy deklaracji otrzymują wartość pustego ciągu znaków.

[§]Szpanerzy i puryści używają terminu: *konkatenacja ciągów znaków.*

Po uruchomieniu programu wpisujemy wyraz, a program wypisuje TAK lub NIE. Na przykład po wpisaniu **kajak** otrzymujemy:

```
kajak
TAK
```

natomiast po wpisaniu **canoe** otrzymujemy:

```
canoe
NIE
```