Babę zesłał Bóg

Wskazówka

Wczytujemy ilość wyrazów do sprawdzenia n.

Dla każdego wyrazu wywołujemy funkcję `baba()`, której argumentem jest wyraz (ciąg znaków), a rezultat funkcji jest wartością logiczną (prawda/fałsz).

W najprostszej swojej wersji funkcja `baba()` poszukuje kolejnych liter tworzących wyraz (wzorzec) **baba**. Jeśli nie napotka którejkolwiek z nich (we właściwym momencie), wówczas przerywa szukanie i zwraca wartość fałszu. Jeśli szukanie dojdzie spokojnie do końca, wtedy zwracana jest wartość prawdy.

Kod programu w języku C++

W funkcji `main()` występuje pętla, gdzie warunkiem logicznym jest wartość zmiennej n, zmniejszająca się o jeden za każdym obiegiem pętli. Wartość równa zero oznacza logiczny fałsz, czyli zakończenie pętli.

Objawem nieznalezienia danej litery jest dojście w pętli (po zmiennej i) do końca przeglądanego ciągu znaków:

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

bool baba(string s)
{
  unsigned int i=0;
  while(s[i] != 'b') {i++; if(i == s.size()) return false; }
  while(s[i] != 'a') {i++; if(i == s.size()) return false; }
  while(s[i] != 'b') {i++; if(i == s.size()) return false; }
  while(s[i] != 'a') {i++; if(i == s.size()) return false; }
  return true;
}

int main()
{
  int n; cin >> n;
  while(n--)
  {
    string s; cin >> s;
    if(baba(s))
      cout << "Tak" << endl;
```

```
    else
      cout << "Nie" << endl;
  }
}
```

Wyjątkowo użyliśmy typu `unsigned int` (liczba całkowita bez znaku), gdyż jest to właśnie typ, jakim numerowane są kolejne znaki w ciągu (ale można też użyć zwykłego typu `int`).

Do wyszukiwania kolejnych liter wzorca można też użyć bibliotecznej funkcji `find()`.

Kod programu w języku Python

Funkcja `find()` wyszukuje położenie jednego ciągu w drugim. Drugi (opcjonalny) argument tej funkcji to indeks znaku, od którego należy rozpocząć poszukiwanie. Zwrócenie przez tę funkcję wartości -1 oznacza nieznalezienie szukanego ciągu.

```
def baba(s):
  i = s.find('b')
  if i == -1:
    return False
  i = s.find('a', i+1)
  if i == -1:
    return False
  i = s.find('b', i+1)
  if i == -1:
    return False
  i = s.find('a', i+1)
  if i == -1:
    return False
  return True

for _ in range(0, int(input())):
  if baba(input()):
    print('Tak')
  else:
    print('Nie')
```

Uwagi

Bardziej zaawansowani programiści skorzystają z wygodnego (i rozbudowanego) narzędzia, jakim są *wyrażenia regularne* (ang. **regular expressions**, wymawiaj: *regiular ekspreszyns*, z akcentem w pierwszym słowie na początkową sylabę). Jest to narzędzie stosowane powszechnie przez profesjonalistów przy zaawansowanym wyszukiwaniu wzorca i zastępowaniu ciągów znaków. W naszym przypadku wzorzec do wyszukania ma postać (bez względu na język programowania):

```
.*b.*a.*b.*a.*
```

Fragment wzorca `.*` oznacza po prostu dowolny ciąg znaków (być może pusty). Pozwala to na znaczące uproszczenie programu:

— w języku C++11:

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

int main()
{
  int n; cin >> n;
  regex e(".*b.*a.*b.*a.*");
  while(n--)
  {
    string s; cin >> s;
    if(regex_match(s, e))
      cout << "Tak" << endl;
    else
      cout << "Nie" << endl;
  }

}
```

— w języku Python:

```
import re

baba = re.compile('.*b.*a.*b.*a.*')
for _ in range(0, int(input())):
  if baba.match(input()):
    print('Tak')
  else:
    print('Nie')
```