# Scenariusz zajęć nr 12

## Temat: Login dla każdego

## Cele:

W trakcie zajęć uczeń zapozna się z następującymi pojęciami:

- Ciągi znaków (analiza zawartości),
- Łączenie ciągów znaków,
- Kolejność leksykograficzna ciągów znaków,
- „Krótka” ewaluacja złożonych wyrażeń logicznych,
- Użycie operatora inkrementacji `++` wewnątrz innego wyrażenia.

## Wstęp:

Uruchomienie środowiska programistycznego, zapoznanie się z edytorem kodu źródłowego i sposobem kompilacji programu (kontynuacja).

### *Dla nauczyciela:*

W razie braku zainstalowanego środowiska można skorzystać z dowolnego środowiska dostępnego w sieci Internet, na przykład `cpp.sh`.

## Przebieg zajęć:

### *Zapoznanie się z treścią zadania:*

Treść zadania jest dołączona do scenariusza zajęć w formie pliku `cpp_12-login.pdf`.

### *Dla nauczyciela:*

Tematem zajęć jest zadanie *Login dla każdego* dostępne na serwisie `szkopul.edu.pl`, wzorowane na zadaniu *Generate Login* (909A) z serwisu `codeforces.com`. Zadanie można rozwiązać w dowolnym języku programowania, a jego stopień trudności należy ocenić jako właściwy dla uczestników podstawowego kursu programowania.



Projekt Centrum Mistrzostwa Informatycznego współfinansowany jest ze środków Unii Europejskiej z Europejskiego Funduszu Rozwoju Regionalnego w ramach Programu Operacyjnego Polska Cyfrowa na lata 2014 - 2020.

## Wskazówki do rozwiązania zadania:

Zarówno z wczytanego imienia jak i z nazwiska musimy wziąć przynajmniej po jednym (początkowym) znaku. Zauważmy, że z nazwiska nie opłaca się brać ani jednego znaku więcej, bo to tylko „pogorszy sprawę". Sprawa przedstawia się jednak zgoła inaczej z kolejnymi (począwszy od indeksu 1) znakami imienia: jeśli są one wcześniejsze (mniejsze) od początkowego znaku nazwiska, wtedy opłaca się je użyć, gdyż poprawią sytuację leksykograficzną loginu. Należy jedynie uważać, aby najpierw sprawdzić, czy imię nam się już nie skończyło, aby nie sięgać poza długość ciągu znaków.

## Kod programu w języku C++:

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

int main()
{
  string a, b;
  size_t i = 1;
  cin >> a >> b;
  cout << a[0];
  while(i < a.size() && a[i] < b[0])
    cout << a[i++];
  cout << b[0];
  cout << endl;
  return 0;
}
```

Jeśli wyrażenie $i < a.size()$ ma wartość fałszu, wtedy drugi czynnik koniunkcji w ogóle nie jest obliczany. Jest to tak zwana *krótka* ewaluacja wyrażenia logicznego: nie oblicza się pozostałej jego części, jeśli znana jest wartość całości. Tutaj mamy koniunkcję: jeśli pierwszy czynnik jest fałszywy, nie oblicza się drugiego, bo całość i tak będzie fałszywa.

Proszę pamiętać o różnicy pomiędzy wyrażeniem `a++` oraz `++a`, jeśli jest ono użyte jako element innego wyrażenia (w tym przykładzie – jako indeks `string`-u).



Projekt Centrum Mistrzostwa Informatycznego współfinansowany jest ze środków Unii Europejskiej z Europejskiego Funduszu Rozwoju Regionalnego w ramach Programu Operacyjnego Polska Cyfrowa na lata 2014 - 2020.

## Podsumowanie i uwagi końcowe:

Czytanie ciągów znaków ze strumienia `cin` przy pomocy operatora `>>` oznacza czytanie pojedynczych wyrazów oddzielonych spacjami (ogólniej: białymi znakami). Jeśli musimy czytać wiersze tekstu jako całości, wtedy należy zastosować funkcję `getline()`. Jednak w typowych zadaniach algorytmicznych unika się formułowania problemu, który wymagałby użycia tej funkcji, a już na pewno nie miesza się czytania operatorem `>>` oraz funkcją `getline()`, gdyż kryje to spore niespodzianki. Podobnie nie używa się funkcji sprawdzającej, czy jesteśmy już na końcu danych wejściowych – ich rozmiar jest podawany z reguły jako pierwsza dana. Autorzy zadań skupiają się raczej na czystej algorytmice bez technicznych (implementacyjnych) detali.



Projekt Centrum Mistrzostwa Informatycznego współfinansowany jest ze środków Unii Europejskiej z Europejskiego Funduszu Rozwoju Regionalnego w ramach Programu Operacyjnego Polska Cyfrowa na lata 2014 - 2020.