# Objektinis Programavimas

## 1-oji praktinė užduotis "*Įrėmintas pasisveikinimas*"


© dr. Remigijus Paulavičius, remigijus.paulavicius@mii.vu.lt

# Įrėmintas pasisveikinimas

- Užduoties formuluotė
- Reikalavimai skirtingoms programos versijoms
- Versijos (v0.1) analizė
- Versijos (v0.2) analizė
- Versijos (v1.0) analizė

# Užduoties formuluotė (1)

Programa turi nuskaityti vartotojo **vardą** (pvz. Remigijus) ir **atspausdinti** "įrėmintą" pasisveikinimą:

```
**********************
*                    *
* Sveikas, Remigijus! *
*                    *
**********************
```

- Pirmoje eilutėje prasideda rėmelis, kuris yra simbolių ∗ seka. Rėmelio ilgis priklauso nuo:
  - įvesto vartotojo vardo ilgio;
  - pasisveikinimo "Sveikas, ";
  - tarpo ir ∗ simbolio rėmelio pradžioje ir pabaigoje.

# Užduoties formuluotė (2)

- Antroji eilutė prasideda ir baigiasi * simboliu, o vidus užpildytas reikiamu skaičiumi tarpo simbolių.
- Trečioji eilutė susideda iš *, tarpo, pasisveikinimo, tarpo ir *.
- Kervirta ir penkta eilutės bus analogiškos antrąjai ir pirmąjai.

# Reikalavimai versijai (`v0.1`)

## (Terminas: 2019-02-14 ❤️)

- Realizuokite programą, pagal Užduoties formuluotė aprašymą ir taip, kad kiekviena iš 5-ų "rėmelio" eilučių būtų saugoma atskirame kintamajame.
- Realizuotą programą turite patalpinti github (ar alternatyvioje, pvz. gitlab, bitbucket) repozicijoje, kurioje būtų tik Jūsų kurti (*source*) failai, t.y. jokių naudojamo IDE "šiukšlių".
- Kiekvienai iš versijų, sukurkite atskirą releas'ą.

# Reikalavimai versijai `(v0.2)`

## (Terminas: 2019-02-14 ❤️)

- Modifikuokite versiją `(v0.1)` taip, kad jeigu vartotojas yra moteris, tuomet vietoj `"Sveikas, "` rašytų `"Sveika, "`.

# Reikalavimai versijai (`v1.0`)

## (Terminas: 2019-02-14 ❤️)

- Modifikuokite versiją (`v0.2`) taip, kad nereiktų kiekvienos eilutės saugoti atskirame kintamajame.
- Realizuokite galimybę vartotojui nurodyti rėmelio plotį (eilučių skaičių), taip, kad pasisveikinimas išliktų rėmelio viduje.
- Galutinėje versijoje repozicijos `README` faile turi būti aprašyta:
  - ką programa atlieka;
  - visi programos releasai;
  - įdiegimo instrukcija ir kaip programa naudotis. Pavyzdinė repozicija pateikta: https://github.com/objprog/pasisveikinimas

# Versijos (v0.1) analizė (1)

- Daugelis C++ priemonių, pvz., įvesties ir išvesties valdymas (**input-output**), yra standartinės bibliotekos (**standard library**), o ne pagrindinės kalbos (**core language**) dalis
- Šis skirtumas yra svarbus, nes pagrindinė kalba yra visada prieinama, tačiau turime paprašyti (naudojant `#include`) standartinės bibliotekos dalių, kurias norite naudoti:

```
#include <iostream>
#include <string>
```

# Versijos (v0.1) analizė (2)

- `#include <iostream>` direktyva kreipiasi į standartinės bibliotekos antraštę (**standard header**).
- Pavyzdžiui, reiškinys:

  ```
  std::cout << "Koks Jūsų vardas: ";
  ```

  naudoja standartinės bibliotekos išvesties operatorių `<<`, tam kad parašyti **"Koks Jūsų vardas: "** į standartinę išvestį.

# Versijos (v0.1) analizė (3)

- `std::` kintamojo vardo pradžioje pažymi, kad naudojamas vardas (`cout`) yra deklaruotas `iostream` header'yje, vardų srityje (**namespace**), pavadintoje `std`.
- Galiausiai `std::cout` reiškia standartinį išvesties srautą. Tai C++ realizacija, kurios pagalba išvedame informaciją į išorinį šaltinį (pvz. monitorių).
- **Kaip manote, koks yra kintamojo `std::cout` tipas?**

# Versijos (v0.1) realizacija

```
#include <iostream>
#include <string>

int main() {
  std::cout << "Koks Jūsų vardas: ";  // paprašome prisistatyti
  std::string vardas;
  std::cin >> vardas;  // nuskaitome vardą
  const std::string sveikinimas = "Sveikas, " + vardas + "!";
  const std::string tarpai(sveikinimas.size(), ' ');
  const std::string antra = "* " + tarpai + " *";
  const std::string pirma(antra.size(), '*');
  std::cout << std::endl;
  std::cout << pirma << std::endl;
  std::cout << antra << std::endl;
  std::cout << "* " << sveikinimas << " *" << std::endl;
  std::cout << antra << std::endl;  // Ketvirta sutampa su antra
  std::cout << pirma << std::endl;  // Penkta sutampa su pirma
  return 0;
}
```

# Versijos (v0.2) analizė (1)

- Vienintelis skirtumas palyginus su `v0.1` yra vietoj reiškinio:

```
const std::string sveikinimas = "Sveikas, " + vardas + "!";
```

- Turime sąlyginį **if**, kuris atpažįsta lytį pagal paskutinę raidę:

```
std::string sveikinimas;
if (vardas.back() == 's') {  // .back() nuo C++11
  sveikinimas = "Sveikas, " + vardas + "!";
} else {
  sveikinimas = "Sveika, " + vardas + "!";
}
```

# Versijos (v0.2) analizė (2)

- Pirmiausiai pastebime, kad kintamasis `sveikinimas` turi būti ne **const** ir deklaruotas prieš **if** ciklą.
- Nuo `C++11` standarto galima naudoti `back()` funkciją norint gauti paskutinį `string` tipo kintamojo (šiuo atveju kintamojo `vardas`) elementą:

```
char ch = vardas.back();
```

- `C++03` standarte `std::string::back()` funkcijos nėra,tačiau tą patį rezultatą galima gauti atliekant atvirštinio pradžios įteratoriaus (*reverse iterator*) gaunamo iš `rbegin()` dereferencing'ą:

```
char ch = *vardas.rbegin();
```

# Versijos (v0.2) realizacija

```
#include <iostream>
#include <string>

int main() {
  std::cout << "Koks Jūsų vardas: ";  // paprašome prisistatyti
  std::string vardas;
  std::cin >> vardas;  // nuskaitome vardą
  std::string sveikinimas;
  if (vardas.back() == 's') {  // .back() nuo C++11
    sveikinimas = "Sveikas, " + vardas + "!";
  } else {
    sveikinimas = "Sveika, " + vardas + "!";
  }
  /* Toliau viskas analogiškai kaip v0.1 */
}
```

# Versijos (v1.0) analizė (1)

- Pradėkime nuo to, ko mums nereikia modifikuoti iš (`v0.2`):

```
#include <iostream>
#include <string>

int main()
{
    std::cout << "Koks Jūsų vardas: ";  // paprašome prisistatyti
    std::string vardas;
    std::cin >> vardas;  // nuskaitome vardą
    std::string sveikinimas;
    if (vardas.back() == 's') {  // .back() nuo C++11
        sveikinimas = "Sveikas, " + vardas + "!";
    } else {
        sveikinimas = "Sveika, " + vardas + "!";
    }
    /* Reikia modifikuoti šią dalį... */
}
```

# Versijos (v1.0) analizė (2)

**Eilučių skaičiaus nustatymas**

- **Sveikinimas** kartu su viršumi ir apačia užima **tris eilutes**.
- Jei žinotume tuščių eilučių/stulpelių tarp sveikinimo ir rėmelio kraštų skaičių (`tarpai`), jį padauginus iš 2 ir pridėjus 3 ankstesnes eilutes, gautume **bendrą eilučių skaičių**:

```
// Tuščių eilučių/stulpelių (tarpų) skaičius iki rėmelio
const int tarpai = 1;
// Bendras eilučių skaičius
const int eilutes = tarpai * 2 + 3;
```

# Versijos (v1.0) analizė (3)

**Stulpelių skaičiaus nustatymas**

- Bendrą stulpelių skaičių (ilgį) gauname iš **sveikinimo ilgio** (`sveikinimas.size()`), kintamojo `tarpai` padauginto iš 2 ir dar dviejų * simbolių pradžioje ir pabaigoje:

```
// Bendras stulpelių skaičius
const std::string::size_type stulpeliai = sveikinimas.size() + tarpai*2 + 2;
```

- Pirmasis `::` yra srities (**scope**) operatorius, todėl `std::string` reiškia tipą iš `std` vardų srities (**namespace**).
- Antrasis `::` reiškia, kad mes norime naudoti `size_type` tipą, kuris yra apibrėžtas **string** klasėje.

# Versijos (v1.0) analizė (4)

- `std::string` tipas apibrėžia `size_type` kaip tinkamą (**unsigned**) tipą, simbolių skaičiui eilutėje (**string**'e) sekti.
- Kai mums reikia lokalaus kintamojo eilutės dydžiui sekti, turėtume naudoti `std::string::size_type` tipą. Tačiau:

```
// Naudojantis C++ 11, bendras stulpelių skaičius
auto stulpeliai = sveikinimas.size()+tarpai*2+2;
```

# Versijos (v1.0) analizė (5)

**Išvengimas pakartotinio `std::` naudojimo**

- C++ galima pasakyti, kad konkretus vardas visada turėtų būti suprantamas kaip iš konkrečios vardų srities (**namespace**) naudojant **using-deklaracijas**:

```
using std::cout;
```

- Tokiu būdu pasakome, kad mes ketiname naudoti `cout` vietoj `std::cout`.

# Versijos (v1.0) analizė (6)

**Ciklų invariantai**

- Invariantą formuluojame taip, kad galėtume įsitikintume, jog **programa elgiasi kaip ketinome**.
- Nors invariantas nėra programos dalis, tai yra vertingas įrankis (**komentaro pavidale**) projektuojant programas.
- Kiekvienas ciklas, turi su juo susijusį invariantą.
- Invariantai padeda geriau suprasti ciklų logiką.

# Versijos (v1.0) analizė (7)

```
int e = 0;  // atspausdintų eilučių skaičius
// invariantas: mes jau atspausdinome e eilučių
while (e != eilutes) {
    // atspausdiname naują eilutę
    ++e;  // padidiname e reikšmę
}
```

- Ekvivalenčiai, vietoj **while** galime naudoti **for** ciklą:

```
// invariantas: mes jau atspausdinome e eilučių
for (int e = 0; e != eilutes; ++e) {
    // atspausdiname naują eilutę
}
```

# Versijos (v1.0) analizė (8)

**Kraštinių simbolių * spausdinimas**

```
int e = 0;                  // atspausdintų eilučių skaičius
string::size_type s = 0;    // atspausdintų stulpelių skaičius
                            // C++11: auto s = 0;
// patikriname ar esame ant rėmelio krašto
if (e == 0 || e == eilutes - 1 || s == 0 || s == stulpeliai - 1) {
    cout << "*";
} else {
    cout << " ";
}
```

- Kodėl reiškiniuose: `e == eilutes - 1` ir `s == stulpeliai - 1` nereikia skliaustų[op]?

---

[op] https://en.cppreference.com/w/cpp/language/operator_precedence

# Versijos (v1.0) analizė (9)

**Sveikinimo, tarpų ir * simbolių spausdinimas**

```
if (e == tarpai + 1 && s == tarpai + 1) {  // ar laikas sveikinimui?
    cout << sveikinimas;
    s += sveikinimas.size();
} else {  // patikriname ar ant krašto?
    if (e == 0 || e == eilutes - 1 || s == 0 || s == stulpeliai - 1)
        cout << "*";
    else  // jei ne sveikinimas ir ne ant krašto tai tarpas
        cout << " ";
    ++s;
}
```

# Versijos (v1.0) realizacija[gh]

```cpp
/*!
 *  \brief     Įrėmintas pasisveikinimas
 *  \details   Padaro įrėmintą sveikinimą pagal vartotojo vardą
 *  \author    Remigijus
 *  \version   1.0
 *  \date      2018-02-20
 */
#include <iostream>
#include <string>
// Standartinės bibliotekos using-deklaracijos
using std::cin;
using std::endl;
using std::cout;
using std::string;

int main() {
  // paprašome prisistatyti
  cout << "Koks Jūsų vardas: ";
  // nuskaitome vardą
  string vardas;
  cin >> vardas;
  string sveikinimas;
  // pagal lytį sudarome sveikinimo tekstą - vidurinė eilutė
  if (vardas.back() == 's') {  // .back() veikia nuo C++11
    sveikinimas = "Sveikas, " + vardas + "!";
  } else {
    sveikinimas = "Sveika, " + vardas + "!";
  }
  // Tuščių eilučių/stulpelių (tarpų) skaičius iki rėmelio
  int tarpai = 1;
  cout << "Įveskite rėmelio dydį (nuo 1 iki 10): ";
  cin >> tarpai;
  // Bendras eilučių skaičius
  const int eilutes = tarpai * 2 + 3;
  // Bendras stulpelių skaičius
  const string::size_type stulpeliai = sveikinimas.size() + tarpai * 2 + 2;
  // tuščia eilutė atskirti įvedimą nuo išvedimo
  cout << endl;
  // invariantas: iki šiol atspausdinome e eilučių
  for (int e = 0; e != eilutes; ++e) {
    string::size_type s = 0;  // Nuo C++11 galima: auto s = 0;
    // invariantas: iki šiol šioje eilutėje atspausdinome s simbolių
    while (s != stulpeliai) {
      // ar laikas sveikinimui?
      if (e == tarpai + 1 && s == tarpai + 1) {
        cout << sveikinimas;
        s += sveikinimas.size();
      } else {
        // patikriname ar ant krašto?
        if (e == 0 || e == eilutes - 1 || s == 0 || s == stulpeliai - 1)
          cout << "*";
        else  // jei ne ant krašto ir ne sveikinimo pozicija
          cout << " ";
        ++s;
      }
    }
    cout << endl;
  }
  return 0;
}
```

---

[gh] https://github.com/objprog/paskaitos2019/wiki/1-osios-užduoties-sprendimo-analizė#versijos-v10-realizacija

# Klausimai !?