

U2: OOP Pagrindai. Klasės ir Objektai

Savaitės: 3-4

Svoris: 1 balas

Terminas: Savaitės 4 pabaiga

Prieš pradedant

Priminimas: Taikomi tie patys reikalavimai kaip U1.

 Žr. [Užduočių Gidas](#)

Užduoties tikslas

Išmokti kurti C++ klases, suprasti **enkapsuliacijos** principą, dirbtį su **konstruktoriais** ir **destruktoriais**.

Praktikuoti **statinius narius** ir masyvų evoliuciją objekto kontekste.

Mokymosi tikslai

Atlikę šią užduotį, mokėsite:

- Transformuoti C struktūrą į C++ klasę
 - Naudoti **private** ir **public** prieigos modifikatorius
 - Kurti konstruktorius (default ir parametrinius)
 - Kurti ir naudoti getterius/setterius
 - Naudoti **statinius narius** (klasės lygio duomenys)
 - Rašyti destruktorius su logging
 - Dirbtis su C-style masyvais klasėje
 - (Bonus) Evoliucija nuo masyvo prie **vector**
-

Užduoties formatas

Žingsnių struktūra:

- **5 pagrindiniai žingsniai** + 1 bonus žingsnis
- **Žingsniai 1-2:** Pilni pavyzdžiai (apšilimas)
- **Žingsniai 3-5:** Jūsų kodas pagal užduotis

Kodo žymėjimai:

- **PAVYZDYS** — pilnas veikiantis kodas (galite kopijuoti)
- **JŪSŲ KODAS** — turite patys implementuoti
- **// UŽDUOTIS:** — komentarai nurodo, ką reikia padaryti

Pavyzdys:

```
void pridetiPazymi(double p) {  
    // UŽDUOTIS: patikrinti ar p tinkamas (0.0 - 10.0)  
    // UŽDUOTIS: pridėti į masyvą  
}
```

Tai reiškia — **jūs rašote kodą**, kuris atlieka tai, kas aprašyta komentaruose.

📋 Užduoties žingsniai

1 žingsnis: Struct → Class transformacija

Direktorija: [U2/01/](#)

Reikalavimai:

Sukurkite **2 failus** tame pačiame žingsnyje, kad būtų matoma transformacija:

Failas student_struct.cpp - C struktūros versija:

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

struct Student {
    char vardas[50];
    int amzius;
    double pazymys;
};

int main() {
    Student s1;
    strcpy(s1.vardas, "Jonas");
    s1.amzius = 20;
    s1.pazymys = 8.5;

    cout << "Studentas: " << s1.vardas
        << ", Amžius: " << s1.amzius
        << ", Pažymys: " << s1.pazymys << endl;

    return 0;
}
```

Kompiliavimas:

```
g++ student_struct.cpp -o struct_versija
./struct_versija
```

Failas student_class.cpp - C++ klasės versija (su public nariais):

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

```
class Student {  
public: // Visi nariai public (kaip struct)  
    char vardas[50];  
    int amzius;  
    double pazymys;  
};  
  
int main() {  
    Student s1;  
    strcpy(s1.vardas, "Petras");  
    s1.amzius = 21;  
    s1.pazymys = 9.0;  
  
    cout << "Studentas: " << s1.vardas  
        << ", Amžius: " << s1.amzius  
        << ", Pažymys: " << s1.pazymys << endl;  
  
    return 0;  
}
```

Kompiliavimas:

```
g++ student_class.cpp -o class_versija  
./class_versija
```

Technikos:

- **struct** - visi nariai **public** pagal nutylėjimą
- **class** - visi nariai **private** pagal nutylėjimą
- Parodyti skirtumą tarp **struct** (viską galima pasiekti) ir **class** (reikia metodų)

Testas:

```
Studentas: Jonas, Amžius: 20, Pažymys: 8.5
```

Failų struktūra:

```
U2/01/  
└── student_struct.cpp  
└── student_class.cpp  
└── README.md (neprivalomas)
```

2 žingsnis: Konstruktoriai

Direktorija: U2/02/

Reikalavimai:

Sukurkite **Student** klasę su **konstruktoariais**:

Student klasė:

Private nariai:

- `char vardas[50]` - studento vardas
- `int amzius` - studento amžius
- `double pazymys` - studento pažymys (vienas)

Public konstruktoriai:

1. Default konstruktorius:

```
Student() {
    strcpy(vardas, "Nezinomas");
    amzius = 0;
    pazymys = 0.0;
    cout << "[TEST] Student sukurtas (default): " << varda << endl;
}
```

2. Parametrinis konstruktorius:

```
Student(const char* v, int a, double p) {
    strcpy(vardas, v);
    amzius = a;
    pazymys = p;
    cout << "[TEST] Student sukurtas: " << varda << endl;
}
```

Technikos:

- Naudokite `strcpy()` (iš `<cstring>`) C-style string'ams
- Konstruktoriai **inicializuoj** objektą sukūrimo metu
- Logging padeda matyti, kada objektais kuriami

Testas:

```
int main() {
    Student s1; // Default konstruktorius
    Student s2("Jonas", 20, 8.5); // Parametrinis
```

```
Student s3("Petras", 21, 9.0);

return 0;
}
```

Švvestis:

```
[TEST] Student sukurtas (default): Nezinomas
[TEST] Student sukurtas: Jonas
[TEST] Student sukurtas: Petras
```

Failų struktūra:

```
U2/02/
├── Student.h
├── Student.cpp
└── main.cpp
└── Makefile
```

3 žingsnis: Metodai (getters, setters, utility)

Direktorija: U2/03/

Reikalavimai:

Pridékite **metodus** prie Student klasés:

Public metodai:

Getters (const metodai) - PAVYZDYS:

```
const char* gautiVarda() const {
    return vardas;
}

int gautiAmziu() const {
    // UŽDUOTIS: grąžinti amžių
}

double gautiPazymi() const {
    // UŽDUOTIS: grąžinti pažymį
}
```

Setters - JŪSŲ KODAS:

```
void nustatytiVarda(const char* v) {
    // UŽDUOTIS: nukopijuoti vardą su strcpy()
}

void nustatytiAmziu(int a) {
    // UŽDUOTIS: patikrinti ar amžius tinkamas (> 0 ir < 120)
    // UŽDUOTIS: jei taip - priskirti
}

void nustatytiPazymi(double p) {
    // UŽDUOTIS: patikrinti ar pažymys tinkamas (0.0 - 10.0)
    // UŽDUOTIS: jei taip - priskirti
}
```

Pagalbiniai metodai - JŪSŲ KODAS:

```
bool arPilnametis() const {
    // UŽDUOTIS: grąžinti true jei amzius >= 18, kitaip false
}

void spausdinti() const {
```

```
// UŽDUOTIS: išspausdinti "Studentas: <vardas>, Amžius: <amzius>, Pažymys:  
<pazymys>"  
// UŽDUOTIS: jei pilnametis, pridėti " (pilnametis)"  
}
```

Technikos:

- `const` metodai - **nekeičia** objekto būsenos
- Getters - **tik skaito** duomenis
- Setters - **modifikuoja** duomenis (su patvirtinimu!)
- Pagalbiniai metodai - **skaičiuoja** ar **tikrina** logiką

Testas:

```
int main() {  
    Student s1("Jonas", 20, 8.5);  
    s1.spausdinti();  
  
    s1.nustatytiPazymi(9.0);  
    cout << "Naujas pažymys: " << s1.gautiPazymi() << endl;  
  
    Student s2("Petras", 16, 7.5);  
    s2.spausdinti();  
  
    return 0;  
}
```

Įšvestis:

```
Studentas: Jonas, Amžius: 20, Pažymys: 8.5 (pilnametis)  
Naujas pažymys: 9  
Studentas: Petras, Amžius: 16, Pažymys: 7.5
```

4 žingsnis: Static counter + destruktorius

Direktorija: U2/04/

Reikalavimai:

Pridékite **statinius narius** ir **destruktorių**:

Student.h:

```
class Student {
private:
    char vardas[50];
    int amzius;
    double pazymys;

    static int sukurtaStudentu; // Statinis skaitiklis

public:
    Student(); // Default konstruktorius
    Student(const char* v, int a, double p); // Parametrinis
    ~Student(); // Destruktorius

    // Getters, setters, utility metodai...

    static int gautiSukurtaStudentu(); // Static getter
};
```

Student.cpp:

Statinio nario inicializacija (už klasės ribų):

```
int Student::sukurtaStudentu = 0;
```

Konstruktoriai (atnaujinti):

```
Student::Student() {
    strcpy(vardas, "Nezinomas");
    amzius = 0;
    pazymys = 0.0;
    // UŽDUOTIS: padidinti sukurtaStudentu
    // UŽDUOTIS: išspausdinti [TEST] žinutę su vardas ir sukurtaStudentu
}

Student::Student(const char* v, int a, double p) {
    strcpy(vardas, v);
```

```

amzius = a;
pazymys = p;
// UŽDUOTIS: padidinti sukurtaStudentu
// UŽDUOTIS: išspausdinti [TEST] žinutę su vardas ir sukurtaStudentu
}

```

Destruktorius - JŪSŲ KODAS:

```

Student::~Student() {
    // UŽDUOTIS: sumažinti sukurtaStudentu
    // UŽDUOTIS: išspausdinti [TEST] žinutę su vardas ir sukurtaStudentu
}

```

Static getter:

```

int Student::gautiSukurtaStudentu() {
    return sukurtaStudentu;
}

```

Technikos:

- **Static** narys - **bendras visiems objektams** (ne kiekvienas objektas turi savo kopiją)
- Inicializuoti **už klasės ribų**: `int Student::sukurtaStudentu = 0;`
- Destruktorius - **automatiškai** kviečiamas, kai objektas "miršta" (išeina iš scope)
- **Alternatyva**: Inicializavimo sąrašas konstruktoriuje (modernesnė C++ praktika):

```

Student::Student(const char* v, int a, double p)
    : amzius(a), pazymys(p) // Initialization list
{
    strcpy(vardas, v); // char[] reikia body'je
    sukurtaStudentu++;
}

```

Testas:

```

int main() {
    cout << "Studentų: " << Student::gautiSukurtaStudentu() << endl; // 0

    {
        Student s1("Jonas", 20, 8.5);
        Student s2("Petras", 21, 9.0);
        cout << "Studentų: " << Student::gautiSukurtaStudentu() << endl; // 2
    } // s1 ir s2 sunaikinami čia

    cout << "Studentų: " << Student::gautiSukurtaStudentu() << endl; // 0
}

```

```
    return 0;  
}
```

Įšvestis:

```
Studentų: 0  
[TEST] Student sukurtas: Jonas. Viso studentų: 1  
[TEST] Student sukurtas: Petras. Viso studentų: 2  
Studentų: 2  
[TEST] Student sunaikintas: Petras. Liko studentų: 1  
[TEST] Student sunaikintas: Jonas. Liko studentų: 0  
Studentų: 0
```

5 žingsnis: Pažymių masyvas

Direktorija: U2/05/

Reikalavimai:

Pakeiskite klasę - vietoj **vieno pažymio** naudokite **pažymių masyvą**:

Private nariai:

```
private:
    char vardas[50];
    int amzius;

    // NAUJAS KODAS: pažymių masyvas
    static const int MAX_PAZYMIU = 20;
    double pazymiai[MAX_PAZYMIU];
    int paizmiuKiekis;

    static int sukurtaStudentu;
```

Konstruktoriai (atnaujinti):

```
Student::Student() {
    strcpy(vardas, "Nezinomas");
    amzius = 0;
    paizmiuKiekis = 0; // Pradžioje 0 pažymių
    sukurtaStudentu++;
    // ...
}

Student::Student(const char* v, int a) { // Be pažymio!
    strcpy(vardas, v);
    amzius = a;
    paizmiuKiekis = 0;
    sukurtaStudentu++;
    // ...
}
```

Nauji metodai - JŪSŲ KODAS:

```
void pridetiPazymi(double p) {
    // UŽDUOTIS: patikrinti ar paizmiuKiekis < MAX_PAZYMIU
    // UŽDUOTIS: patikrinti ar pažymys tinkamas (0.0 - 10.0)
    // UŽDUOTIS: jei viskas OK - pridėti į masyvą ir padidinti kiekį
    // UŽDUOTIS: jei ne - išspausdinti klaidos pranešimą
}
```

```

double skaiciuotiVidurki() const {
    // UŽDUOTIS: jei pazymiuKiekis == 0, grąžinti 0.0
    // UŽDUOTIS: suskaičiuoti sumą visų pažymių (naudoti for ciklą)
    // UŽDUOTIS: grąžinti suma / pazymiuKiekis
}

void spausdintiPazymius() const {
    // UŽDUOTIS: išspausdinti "Pažymiai (<kiekis>): "
    // UŽDUOTIS: išspausdinti visus pažymius atskiriant kableliais
    // Pavyzdys: "Pažymiai (4): 8.5, 9, 7.5, 8"
}

```

Atnaujinta spausdinti() - JŪSŲ KODAS:

```

void spausdinti() const {
    // UŽDUOTIS: išspausdinti "Studentas: <vardas>, Amžius: <amzius>"
    // UŽDUOTIS: iškvesti spausdintiPazymius()
    // UŽDUOTIS: išspausdinti "Vidurkis: <vidurkis>"
}

```

Testas:

```

int main() {
    Student s1("Jonas", 20);

    s1.pridetiPazymi(8.5);
    s1.pridetiPazymi(9.0);
    s1.pridetiPazymi(7.5);
    s1.pridetiPazymi(8.0);

    s1.spausdinti();

    return 0;
}

```

Išvestis:

```

[TEST] Student sukurtas: Jonas. Viso studentų: 1
Studentas: Jonas, Amžius: 20
Pažymiai (4): 8.5, 9, 7.5, 8
Vidurkis: 8.25
[TEST] Student sunaikintas: Jonas. Liko studentų: 0

```

6 žingsnis (BONUS +0.2 balo): Masyvas → vector

Direktorija: U2/06-bonus/

Reikalavimai:

Pakeiskite **masyvą** į **vector<double>** (kaip U1 žingsnis 4):

Student.h:

```
#include <vector> // SVARBU!

class Student {
private:
    char vardas[50];
    int amzius;

    // NAUJAS KODAS: vector
    vector<double> pazymiai;

    static int sukurtaStudentu;

public:
    // ...
};
```

Konstruktoriai (atnaujinti):

```
Student::Student() {
    strcpy(vardas, "Nezinomas");
    amzius = 0;
    // pazymiai jau tuščias vector (default konstruktorius)
    sukurtaStudentu++;
}
```

Atnaujinti metodai - JŪSŲ KODAS:

```
void pridetiPazymi(double p) {
    // UŽDUOTIS: patikrinti ar p tinkamas (0.0 - 10.0)
    // UŽDUOTIS: naudoti pazymiai.push_back(p)
}

double skaiciuotiVidurki() const {
    // UŽDUOTIS: patikrinti ar pazymiai.empty()
    // UŽDUOTIS: suskaičiuoti sumą (galite naudoti range-based for: for (double p
    : pazymiai))
    // UŽDUOTIS: grąžinti suma / pazymiai.size()
```

```
}

void spausdintiPazymius() const {
    // UŽDUOTIS: išspausdinti "Pažymiai (<pazymiai.size()%>): "
    // UŽDUOTIS: iteruoti per pazymiai ir spausdinti
}
```

PRIVALUMAI:

- Dinaminis dydis (ne limitai!)
- Paprastesnis kodas (`push_back()`, `size()`, `empty()`)
- Range-based for loop (moderniškas C++)

Testas: tas pats kaip žingsnis 5, bet be limitų.

Pateikimas

GitLab direktorių struktūra:

```

cpp-2026/
├── README.md
├── .gitignore
└── U1/
└── U2/
    ├── README.md      ← Užduoties santrauka (PRIVALOMA)
    ├── 01/             ← Struct → Class
    │   ├── student_struct.cpp
    │   └── student_class.cpp
    ├── 02/             ← Konstruktoriai
    │   ├── Student.h
    │   ├── Student.cpp
    │   ├── main.cpp
    │   └── Makefile
    ├── 03/             ← Metodai (getters/setters)
    │   ├── Student.h
    │   ├── Student.cpp
    │   ├── main.cpp
    │   └── Makefile
    ├── 04/             ← Static + destruktoriai
    │   ├── Student.h
    │   ├── Student.cpp
    │   ├── main.cpp
    │   └── Makefile
    ├── 05/             ← Pažymiu masyvas (FINAL)
    │   ├── Student.h
    │   ├── Student.cpp
    │   ├── main.cpp
    │   └── Makefile
    └── 06-bonus/       ← BONUS: vector (neprivalomas)
        ├── Student.h
        ├── Student.cpp
        ├── main.cpp
        └── Makefile

```

Git workflow:

```

git add U2/01/
git commit -m "U2: 1 žingsnis - Struct → Class transformacija"
git push

git add U2/02/
git commit -m "U2: 2 žingsnis - Konstruktoriai"
git push

```

```
# ... ir t.t.
```

U2/README.md šablonas:

```
# U2: OOP Pagrindai. Klasės ir Objektais
```

****Būsena**:** Atlikta

****Pateikta**:** 2026-03-01

📁 Žingsniai

Žingsnis	Direktorija	Aprašymas
1	`01/`	Struct → Class transformacija
2	`02/`	Konstruktoriai (default + parametrinis)
3	`03/`	Metodai (getters, setters, pagalbiniai)
4	`04/`	Static counter + destruktorius
5	`05/`	Pažymių masyvas
6	`06-bonus/`	BONUS: array → vector

✎ Testavimas

Testas 1 (konstruktoriai):

```
[TEST] Student sukurtas: Jonas. Viso studentų: 1  
 VEIKIA
```

Testas 2 (pažymių vidurkis):

```
Pažymiai (4): 8.5, 9, 7.5, 8  
Vidurkis: 8.25  
 VEIKIA
```

☁ Pagrindinės ižvalgos

1. Class vs Struct - private vs public
2. Konstruktoriai inicializuoją objektą
3. Static nariai - bendri visiems objektams
4. Destruktorius - automatiškas cleanup
5. Masyvas → vector (lankstesnis!)

Moodle pateikimas:

```
cd cpp-2026
git archive --format=zip --output=U2_VardasPavarde.zip HEAD U2/ README.md
.gitignore
```

💡 Patarimai

1. Struct vs Class:

- `struct` - public pagal nutylėjimą
- `class` - private pagal nutylėjimą

2. Konstruktoriai:

- Naudokite `strcpy()` C-style string'ams
- Inicializuokite **visus** narius

3. Static nariai:

- Deklaruoti klasėje: `static int sukurtaStudentu;`
- Inicializuoti už klasės ribų (`.cpp`): `int Student::sukurtaStudentu = 0;`

4. Const metodai:

- Getters **visada const**
- Pagalbiniai metodai, kurie **nekeičia** objekto - `const`

5. Patikrinimas:

- Amžius: `> 0 && < 120`
- Pažymys: `>= 0.0 && <= 10.0`
- Masyvo ribos: `pazymiuKiekis < MAX_PAZYMIU`

6. Seną kodą galite ištrinti - Git saugo visą istoriją commit'uose

🔗 Naudingos nuorodos

- [C++ classes](#)
 - [Constructors](#)
 - [Destructors](#)
 - [Static members](#)
 - [Vector container](#)
-

❓ Dažnai užduodami klausimai

K: Kodėl `strcpy()` vietoj `string`?

A: Demonstruoti C-style string'us (tradicinė sintaksė). Vėliau (U4+) naudosime `std::string`.

K: Ar privaloma naudoti `const` metodams?

A: Taip, getters ir utility metodai **turi būti** `const` (gera praktika).

K: Kodėl static narys inicializuojamas už klasės?

A: Nes static narys - **bendras visiems objektams**, ne kiekvieno objekto dalis. Reikia **vienos kopijos** visai klasei.

K: Ar destruktorius visada automatinis?

A: Taip, destruktorius **automatiškai** kviečiamas, kai objektas išeina iš scope (arba `delete` jei dinaminis).

K: Ar žingsnis 6 (bonus) privalomas?

A: Ne, tai **bonus** (+0.2 balo). Bet rekomenduojama stipresniems studentams.

Daugiau klausimų? → Žr. [Užduočių Gidas - DUK](#)

Sékmės! 🎉