Objektinis programavimas

Įvadinė(s) paskaita(-os)

---

dr. Remigijus Paulavičius

@RemigPau

2018 m. vasario 8 d.

Vilniaus Universitetas

Turinys

Dalyko (modulio) apžvalga

Dalyko (modulio) tikslas

Dalyko (modulio) tikslas:

Supažindinti su objektiškai orientuoto programavimo (OOP) koncepcija bei **išmokinti kurti objektiškai orientuotas efektyvias programas**.

Dalyko (modulio) studijų siekiniai:

- Gebės suprasti objektiškai orientuoto programavimo (OOP) koncepciją.
- Gebės taikyti pagrindinius OOP koncepcijos principus.
- Gebės analizuoti dalykinės veiklos sritį ir nustatyti programinės įrangos kūrimo, atnaujinimo tikslus, rengti Doxygen tipo dokumentacijas.
- Gebės parinkti efektyvius algoritmus ir į spartą orientuotas (priklausomai nuo uždavinio specifikos) duomenų struktūras, pritaikyti sistemų projektavimo žinias sprendžiant užduotis.
- Gebės projektuoti ir įgyvendinti algoritmus daugiaprocesorinėse sistemose, išmanys našiųjų skaičiavimų taikymo sritis.
- Gebės planuoti ir atlikti tiriamojo pobūdžio eksperimentus, vertinti rezultatus, jais remiantis daryti išvadas.

Dalyko (modulio) tikslas

Dalyko (modulio) tikslas:

Supažindinti su objektiškai orientuoto programavimo (OOP) koncepcija bei **išmokinti kurti objektiškai orientuotas efektyvias programas**.

Dalyko (modulio) studijų siekiniai:

- Gebės suprasti objektiškai orientuoto programavimo (OOP) koncepciją.
- Gebės taikyti pagrindinius OOP koncepcijos principus.
- Gebės analizuoti dalykinės veiklos sritį ir nustatyti programinės įrangos kūrimo, atnaujinimo tikslus, rengti Doxygen tipo dokumentacijas.
- Gebės parinkti efektyvius algoritmus ir į spartą orientuotas (priklausomai nuo uždavinio specifikos) duomenų struktūras, pritaikyti sistemų projektavimo žinias sprendžiant užduotis.
- Gebės projektuoti ir įgyvendinti algoritmus daugiaprocesorinėse sistemose, išmanys našiųjų skaičiavimų taikymo sritis.
- Gebės planuoti ir atlikti tiriamojo pobūdžio eksperimentus, vertinti rezultatus, jais remiantis daryti išvadas.

Dalyko (modulio) tikslas

Dalyko (modulio) tikslas:

Supažindinti su objektiškai orientuoto programavimo (OOP) koncepcija bei **išmokinti kurti objektiškai orientuotas efektyvias programas**.

Dalyko (modulio) studijų siekiniai:

- Gebės suprasti objektiškai orientuoto programavimo (OOP) koncepciją.
- Gebės taikyti pagrindinius OOP koncepcijos principus.
- Gebės analizuoti dalykinės veiklos sritį ir nustatyti programinės įrangos kūrimo, atnaujinimo tikslus, rengti Doxygen tipo dokumentacijas.
- Gebės parinkti efektyvius algoritmus ir į spartą orientuotas (priklausomai nuo uždavinio specifikos) duomenų struktūras, pritaikyti sistemų projektavimo žinias sprendžiant užduotis.
- Gebės projektuoti ir įgyvendinti algoritmus daugiaprocesorinėse sistemose, išmanys našiųjų skaičiavimų taikymo sritis.
- Gebės planuoti ir atlikti tiriamojo pobūdžio eksperimentus, vertinti rezultatus, jais remiantis daryti išvadas.

Dalyko (modulio) tikslas

Dalyko (modulio) tikslas:

Supažindinti su objektiškai orientuoto programavimo (OOP) koncepcija bei **išmokinti kurti objektiškai orientuotas efektyvias programas**.

Dalyko (modulio) studijų siekiniai:

- Gebės suprasti objektiškai orientuoto programavimo (OOP) koncepciją.
- Gebės taikyti pagrindinius OOP koncepcijos principus.
- Gebės analizuoti dalykinės veiklos sritį ir nustatyti programinės įrangos kūrimo, atnaujinimo tikslus, rengti Doxygen tipo dokumentacijas.
- Gebės parinkti efektyvius algoritmus ir į spartą orientuotas (priklausomai nuo uždavinio specifikos) duomenų struktūras, pritaikyti sistemų projektavimo žinias sprendžiant užduotis.
- Gebės projektuoti ir įgyvendinti algoritmus daugiaprocesorinėse sistemose, išmanys našiųjų skaičiavimų taikymo sritis.
- Gebės planuoti ir atlikti tiriamojo pobūdžio eksperimentus, vertinti rezultatus, jais remiantis daryti išvadas.

Dalyko (modulio) tikslas

Dalyko (modulio) tikslas:

Supažindinti su objektiškai orientuoto programavimo (OOP) koncepcija bei **išmokinti kurti objektiškai orientuotas efektyvias programas**.

Dalyko (modulio) studijų siekiniai:

- Gebės suprasti objektiškai orientuoto programavimo (OOP) koncepciją.
- Gebės taikyti pagrindinius OOP koncepcijos principus.
- Gebės analizuoti dalykinės veiklos sritį ir nustatyti programinės įrangos kūrimo, atnaujinimo tikslus, rengti Doxygen tipo dokumentacijas.
- Gebės parinkti efektyvius algoritmus ir į spartą orientuotas (priklausomai nuo uždavinio specifikos) duomenų struktūras, pritaikyti sistemų projektavimo žinias sprendžiant užduotis.
- Gebės projektuoti ir įgyvendinti algoritmus daugiaprocesorinėse sistemose, išmanys našiųjų skaičiavimų taikymo sritis.
- Gebės planuoti ir atlikti tiriamojo pobūdžio eksperimentus, vertinti rezultatus, jais remiantis daryti išvadas.

Dalyko (modulio) tikslas

Dalyko (modulio) tikslas:

Supažindinti su objektiškai orientuoto programavimo (OOP) koncepcija bei **išmokinti kurti objektiškai orientuotas efektyvias programas**.

Dalyko (modulio) studijų siekiniai:

- Gebės suprasti objektiškai orientuoto programavimo (OOP) koncepciją.
- Gebės taikyti pagrindinius OOP koncepcijos principus.
- Gebės analizuoti dalykinės veiklos sritį ir nustatyti programinės įrangos kūrimo, atnaujinimo tikslus, rengti Doxygen tipo dokumentacijas.
- Gebės parinkti efektyvius algoritmus ir į spartą orientuotas (priklausomai nuo uždavinio specifikos) duomenų struktūras, pritaikyti sistemų projektavimo žinias sprendžiant užduotis.
- Gebės projektuoti ir įgyvendinti algoritmus daugiaprocesorinėse sistemose, išmanys našiųjų skaičiavimų taikymo sritis.
- Gebės planuoti ir atlikti tiriamojo pobūdžio eksperimentus, vertinti rezultatus, jais remiantis daryti išvadas.

Dalyko (modulio) tikslas

Dalyko (modulio) tikslas:

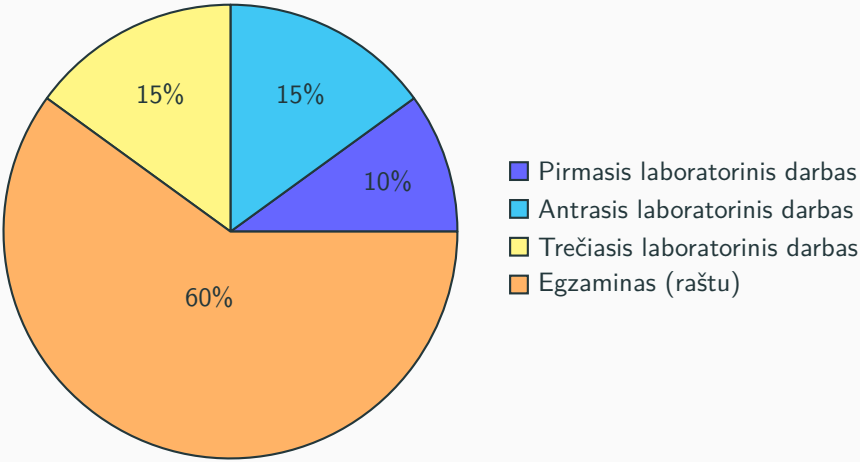
Supažindinti su objektiškai orientuoto programavimo (OOP) koncepcija bei **išmokinti kurti objektiškai orientuotas efektyvias programas**.

Dalyko (modulio) studijų siekiniai:

- Gebės suprasti objektiškai orientuoto programavimo (OOP) koncepciją.
- Gebės taikyti pagrindinius OOP koncepcijos principus.
- Gebės analizuoti dalykinės veiklos sritį ir nustatyti programinės įrangos kūrimo, atnaujinimo tikslus, rengti Doxygen tipo dokumentacijas.
- Gebės parinkti efektyvius algoritmus ir į spartą orientuotas (priklausomai nuo uždavinio specifikos) duomenų struktūras, pritaikyti sistemų projektavimo žinias sprendžiant užduotis.
- Gebės projektuoti ir įgyvendinti algoritmus daugiaprocesorinėse sistemose, išmanys našiųjų skaičiavimų taikymo sritis.
- Gebės planuoti ir atlikti tiriamojo pobūdžio eksperimentus, vertinti rezultatus, jais remiantis daryti išvadas.

Dalyko apžvalga

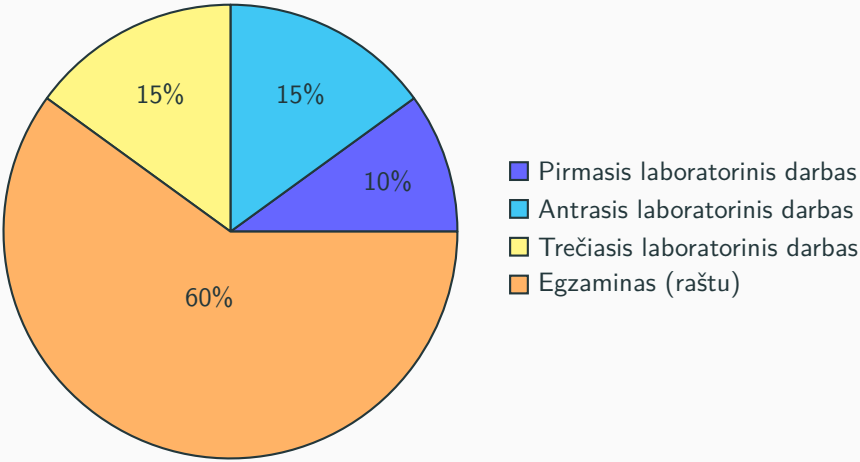
Vertinimo strategija



Pirmasis laboratorinis darbas

Studentams skiriamos individualios užduotys, apimančios 1-3 temas. Maksimalus įvertis už puikiai atliktas užduotis yra 2 balai (atitinkantys 10% bendrojo svorio). Skiriami papildomi balai (iki 20% maksimalaus įverčio svorio) kai užduotys atsiskaitomos anksčiau nurodytų terminų. Analogiškai, vėluojant atsiskaityti galutinis įvertinimas yra mažinamas (iki 20% maksimalaus įverčio svorio).

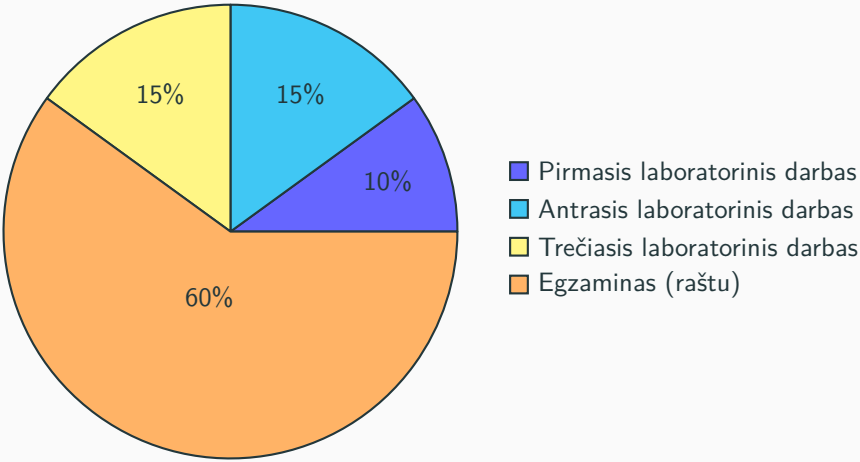
Vertinimo strategija



Antrasis laboratorinis darbas

Studentams skiriamos individualios užduotys, apimančios 4-6 temas. Maksimalus įvertis už puikiai atliktas užduotis yra 3 balai (atitinkantys 15% bendrojo svorio). Skiriami papildomi balai (iki 20% maksimalaus įverčio svorio) kai užduotys atsiskaitomos anksčiau nurodytų terminų. Analogiškai, vėluojant atsiskaityti galutinis įvertinimas yra mažinamas (iki 20% maksimalaus įverčio svorio).

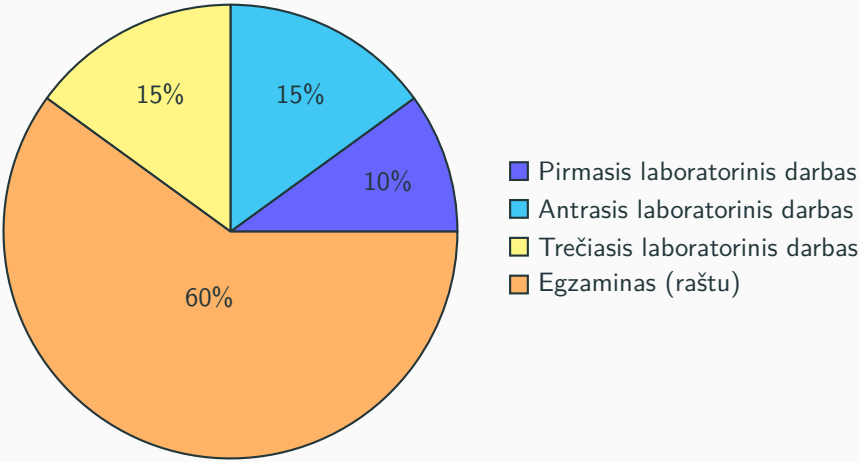
Vertinimo strategija



Trečiasis laboratorinis darbas

Studentams skiriamos individualios užduotys, apimančios 7-9 temas. Maksimalus įvertis už puikiai atliktas užduotis yra 3 balai (atitinkantys 15% bendrojo svorio). Skiriami papildomi balai (iki 20% maksimalaus įverčio svorio) kai užduotys atsiskaitomos anksčiau nurodytų terminų. Analogiškai, vėluojant atsiskaityti galutinis įvertinimas yra mažinamas (iki 20% maksimalaus įverčio svorio).

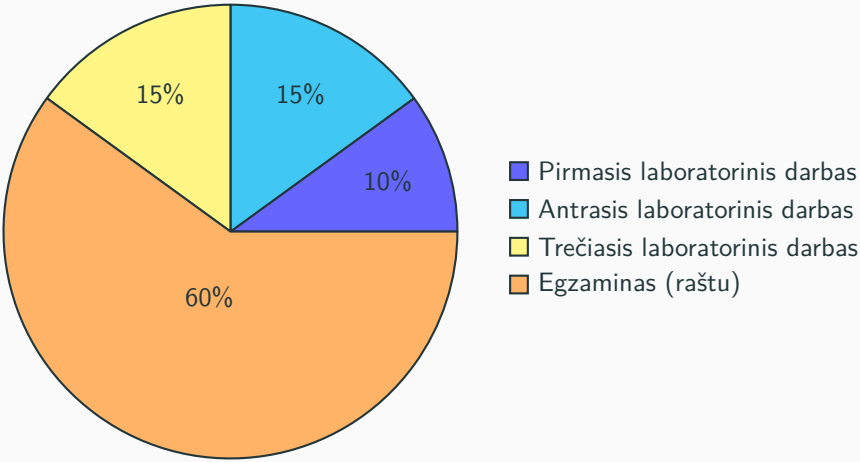
Vertinimo strategija



Egzaminas (raštu)

Egzaminą laikyti leidžiama semestro metu surinkus ne mažiau, nei minimalų 2.0 balų skaičių, atitinkantį 25% laboratoriniams darbams skirtojo svorio. Egzamino metu galima surinkti iki 12 taškų, kurie atitinka 60% galutinio įvertinimo. Egzamino susideda iš dviejų etapų. Pirmiausia, studentas turi atsakyti į įvairaus sudėtingumo klausimus iš paskaitose pateiktų temų (iki 4 taškų). Antroje egzamino dalyje studentas turi pateikti praktinį pateiktos problemos sprendimą (iki 8 taškų), motyvuojant naudojamų priemonių efektyvumą, bei analizuojant alternatyvius užduoties sprendimo būdus.

Vertinimo strategija



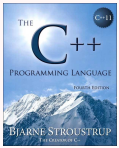
Išreiškiame savo nuomonę (anonimiškai):

- www.poll.si
- poll's ID - **Icka**

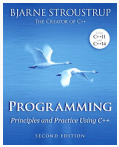
Privaloma literatūra

Paremta: **The Definitive C++ Book Guide and List (Stack Overflow)**:

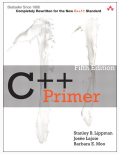
Bjarne Stroustrup
The C++ Programming Language, 4th Ed.
Addison-Wesley, 2013.



Bjarne Stroustrup
Programming: Principles and Practice Using C++, 2nd Ed.
Addison-Wesley, 2014.



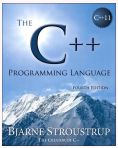
Stanley Lippman, Josée Lajoie, and Barbara E. Moo
C++ Primer, 5th Ed.
Addison-Wesley, 2012.



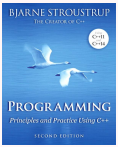
Privaloma literatūra

Paremta: **The Definitive C++ Book Guide and List (Stack Overflow)**:

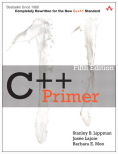
Bjarne Stroustrup
The C++ Programming Language, 4th Ed.
Addison-Wesley, 2013.



Bjarne Stroustrup
Programming: Principles and Practice Using C++, 2nd Ed.
Addison-Wesley, 2014.



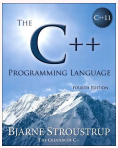
Stanley Lippman, Josée Lajoie, and Barbara E. Moo
C++ Primer, 5th Ed.
Addison-Wesley, 2012.



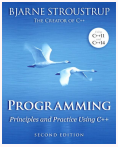
Privaloma literatūra

Paremta: **The Definitive C++ Book Guide and List (Stack Overflow)**:

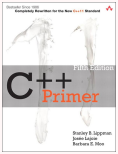
Bjarne Stroustrup
The C++ Programming Language, 4th Ed.
Addison-Wesley, 2013.



Bjarne Stroustrup
Programming: Principles and Practice Using C++, 2nd Ed.
Addison-Wesley, 2014.



Stanley Lippman, Josée Lajoie, and Barbara E. Moo
C++ Primer, 5th Ed.
Addison-Wesley, 2012.

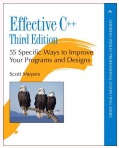


Papildoma literatūra

Scott Meyers

Effective C++: 55 Specific Ways to Improve Your Programs and Designs

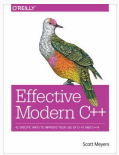
Addison-Wesley, 2005.



Scott Meyers

Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14

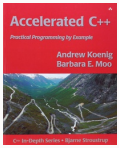
Addison-Wesley, 2014.



Andrew Koenig and Barbara E. Moo

Accelerated C++. Practical Programming by Example

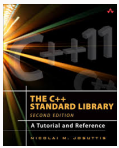
Addison-Wesley, 2000.



Nicolai M. Josuttis

The C++ Standard Library: A Tutorial and Reference (2nd Ed.)

Addison-Wesley, 2012.

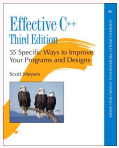


Papildoma literatūra

Scott Meyers

Effective C++: 55 Specific Ways to Improve Your Programs and Designs

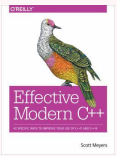
Addison-Wesley, 2005.



Scott Meyers

Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14

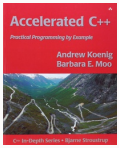
Addison-Wesley, 2014.



Andrew Koenig and Barbara E. Moo

Accelerated C++. Practical Programming by Example

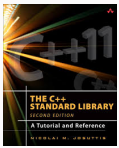
Addison-Wesley, 2000.



Nicolai M. Josuttis

The C++ Standard Library: A Tutorial and Reference (2nd Ed.)

Addison-Wesley, 2012.

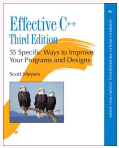


Papildoma literatūra

Scott Meyers

Effective C++: 55 Specific Ways to Improve Your Programs and Designs

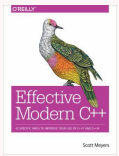
Addison-Wesley, 2005.



Scott Meyers

Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14

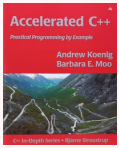
Addison-Wesley, 2014.



Andrew Koenig and Barbara E. Moo

Accelerated C++. Practical Programming by Example

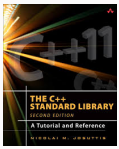
Addison-Wesley, 2000.



Nicolai M. Josuttis

The C++ Standard Library: A Tutorial and Reference (2nd Ed.)

Addison-Wesley, 2012.

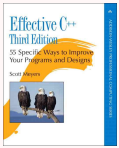


Papildoma literatūra

Scott Meyers

Effective C++: 55 Specific Ways to Improve Your Programs and Designs

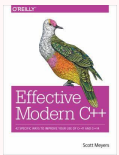
Addison-Wesley, 2005.



Scott Meyers

Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14

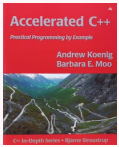
Addison-Wesley, 2014.



Andrew Koenig and Barbara E. Moo

Accelerated C++. Practical Programming by Example

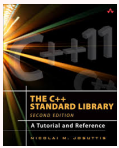
Addison-Wesley, 2000.



Nicolai M. Josuttis

The C++ Standard Library: A Tutorial and Reference (2nd Ed.)

Addison-Wesley, 2012.



Kodėl C++?

Kodėl C++?

Kaip bebūtų netikėta ir keista, tačiau "senukė" C buvo greičiausiai 2017 auganti programavimo kalba pagal **TIOBE** indeksą ir nominuota **metų programavimo kalba**:

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215%	-3.06%
2	2		C	11.037%	+1.69%
3	3		C++	5.603%	-0.70%
4	5	^	Python	4.678%	+1.21%
5	4	v	C#	3.754%	-0.29%
6	7	^	JavaScript	3.465%	+0.62%
7	6	v	Visual Basic .NET	3.261%	+0.30%
8	16	^^	R	2.549%	+0.76%
9	10	^	PHP	2.532%	-0.03%
10	8	v	Perl	2.419%	-0.33%

Figure 1: TIOBE indeksas, 2018 m. sausis

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

- **Naujos galimybės**: "classes, member functions, derived classes, separate compilation, public and private access control, friends, type checking of function arguments, default arguments, inline functions, overloaded assignment operator, constructors, destructors, f() same as f(void), call-function and return-function (synchronization features, not in C++)"

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

2007: Parengtas TR1. Oficialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace std:: dalimi. Oficialus pavadinimas *Information Technology — Programming*

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

2007: Parengtas TR1. Officialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace std:: dalimi. Oficialus pavadinimas *Information Technology — Programming Languages — C++* (ISO/IEC 14882:2011).

2014: C++14 standartas yra ankstesniojo C++11 standarto atnaujinimai ir klaidų pataisymai (ISO/IEC 14882:2014).

2017: C++17 - naujausias C++ standartas (ISO/IEC 14882:2017).

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

2007: Parengtas TR1. Officialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace std:: dalimi. Oficialus pavadinimas *Information Technology — Programming Languages — C++* (ISO/IEC 14882:2011).

2014: C++14 standartas yra ankstesniojo C++11 standarto atnaujinimai ir klaidų pataisymai (ISO/IEC 14882:2014).

2017: C++17 - naujausias C++ standartas (ISO/IEC 14882:2017).

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

- **Naujos galimybės:** "RTTI (dynamic_cast, typeid), covariant return types, cast operators, mutable, bool, declarations in conditions, template instantiations, member templates, export"
- **Naujos bibliotekos (Library) galimybės:** "containers, algorithms, iterators, function objects (based on STL), locales, bitset, valarray, auto_ptr, templatized string, iostream, and complex."

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

2007: Parengtas TR1. Officialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

2007: Parengtas TR1. Officialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace std:: dalimi. Oficialus pavadinimas *Information Technology — Programming Languages — C++* (ISO/IEC 14882:2011).

2014: C++14 standartas yra ankstesniojo C++11 standarto atnaujinimai ir klaidų pataisymai (ISO/IEC 14882:2014).

2017: C++17 - naujausias C++ standartas (ISO/IEC 14882:2017).

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

- **Naujos galimybės:** "value initialization".

2007: Parengtas TR1. Officialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace std:: dalimi. Oficialus pavadinimas *Information Technology — Programming Languages — C++* (ISO/IEC 14882:2011).

2014: C++14 standartas yra ankstesniojo C++11 standarto atnaujinimai ir klaidų pataisymai (ISO/IEC 14882:2014).

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

2007: Parengtas TR1. Officialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace std:: dalimi. Oficialus pavadinimas *Information Technology — Programming Languages — C++* (ISO/IEC 14882:2011).

2014: C++14 standartas yra ankstesniojo C++11 standarto atnaujinimai ir klaidų pataisymai (ISO/IEC 14882:2014).

2017: C++17 - naujausias C++ standartas (ISO/IEC 14882:2017).

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

2007: Parengtas TR1. Officialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace std:: dalimi. Oficialus pavadinimas *Information Technology — Programming Languages — C++* (ISO/IEC 14882:2011).

2014: C++14 standartas yra ankstesniojo C++11 standarto atnaujinimai ir klaidų pataisymai (ISO/IEC 14882:2014).

2017: C++17 - naujausias C++ standartas (ISO/IEC 14882:2017).

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

2007: Parengtas TR1. Officialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace std:: dalimi. Oficialus pavadinimas *Information Technology — Programming Languages — C++* (ISO/IEC 14882:2011).

2014: C++14 standartas yra ankstesniojo C++11 standarto atnaujinimai ir klaidų pataisymai (ISO/IEC 14882:2014).

2017: C++17 - naujausias C++ standartas (ISO/IEC 14882:2017).

C++ istorija (Standartai) [Group of C++ enthusiasts, 2000]

1979: Pirminė versija C su klasėmis:

1989: C++ standartizacija (International Organization for Standardization (ISO)).

1992: STL tapo C++ dalimi.

1998: Pirmasis C++ standartas C++98. Oficialus pavadinimas - *Information Technology — Programming Languages — C++* (ISO/IEC 14882:1998)

1999: Komiteto nariai Įkuria Boost (`http://www.boost.org/`), kurio tikslas pateikti naujas bibliotekas ateities standartams.

2003: Atnaujintas C++03 standartas, vadinamas "*technical corrigendum*" (TC) - ankstesnių C++98 klaidų ištaisymai (ISO/IEC 14882:2003).

2007: Parengtas TR1. Officialus pavadinimas - *Information Technology — Programming Languages — Technical Report on C++ Library Extensions* (ISO/IEC TR 19768:2007). Realizuoti vardų erdvėsje (angl. namespace) std::tr1.

2011: Antrasis C++11 standartas. C++11 turėjo svarbius atnaujinimus tiek C++ kalbai tiek standartinei bibliotekai, kuriai TR1 atnaujinimai tapo namespace std:: dalimi. Oficialus pavadinimas *Information Technology — Programming Languages — C++* (ISO/IEC 14882:2011).

2014: C++14 standartas yra ankstesniojo C++11 standarto atnaujinimai ir klaidų pataisymai (ISO/IEC 14882:2014).

2017: C++17 - naujausias C++ standartas (ISO/IEC 14882:2017).

C++11 ir C++98 suderinamumas

- C++11 buvo kuriamas taip, kad išliktų pilnai suderinamas su C++98.
- Iš principo, jeigu programa veikė (kompiliavosi) su C++98 ar C++03 standartais, turėtų veikti ir su C++11. Tačiau yra kelios išimtys
  - Kintamieji daugiau negali turėti naujai įvestų raktinių žodžių (**keywords**).
- Atgalinis suderinamumas (**backward compatibility**) taikomas tik programos kodui (**source code**).
- Todėl programas, parengtas naudojant C++98 standartą, turėtume be problemų sukompiliuoti naudojant ir C++11 kompiliatorių.
- Tačiau sukompiliuoto kodo su C++98 kompiliatoriumi apjungimas (**linking**) naudojant C++11 kompiliatorių, gali ir neveikti.

> "*C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.*" [Stroustrup, 2013]

C++11 ir C++98 suderinamumas

- C++11 buvo kuriamas taip, kad išliktų pilnai suderinamas su C++98.
- Iš principo, jeigu programa veikė (kompiliavosi) su C++98 ar C++03 standartais, turėtų veikti ir su C++11. Tačiau yra kelios išimtys
  - Kintamieji daugiau negali turėti naujai įvestų raktinių žodžių (**keywords**).
- Atgalinis suderinamumas (**backward compatibility**) taikomas tik programos kodui (**source code**).
- Todėl programas, parengtas naudojant C++98 standartą, turėtume be problemų sukompiliuoti naudojant ir C++11 kompiliatorių.
- Tačiau sukompiliuoto kodo su C++98 kompiliatoriumi apjungimas (**linking**) naudojant C++11 kompiliatorių, gali ir neveikti.

> "*C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.*" [Stroustrup, 2013]

C++11 ir C++98 suderinamumas

- C++11 buvo kuriamas taip, kad išliktų pilnai suderinamas su C++98.
- Iš principo, jeigu programa veikė (kompiliavosi) su C++98 ar C++03 standartais, turėtų veikti ir su C++11. Tačiau yra kelios išimtys
  - Kintamieji daugiau negali turėti naujai įvestų raktinių žodžių (**keywords**).
- Atgalinis suderinamumas (**backward compatibility**) taikomas tik programos kodui (**source code**).
- Todėl programas, parengtas naudojant C++98 standartą, turėtume be problemų sukompiliuoti naudojant ir C++11 kompiliatorių.
- Tačiau sukompiliuoto kodo su C++98 kompiliatoriumi apjungimas (**linking**) naudojant C++11 kompiliatorių, gali ir neveikti.

> "*C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.*" [Stroustrup, 2013]

C++11 ir C++98 suderinamumas

- C++11 buvo kuriamas taip, kad išliktų pilnai suderinamas su C++98.
- Iš principo, jeigu programa veikė (kompiliavosi) su C++98 ar C++03 standartais, turėtų veikti ir su C++11. Tačiau yra kelios išimtys
  - Kintamieji daugiau negali turėti naujai įvestų raktinių žodžių (**keywords**).
- Atgalinis suderinamumas (**backward compatibility**) taikomas tik programos kodui (**source code**).
- Todėl programas, parengtas naudojant C++98 standartą, turėtume be problemų sukompiliuoti naudojant ir C++11 kompiliatorių.
- Tačiau sukompiliuoto kodo su C++98 kompiliatoriumi apjungimas (**linking**) naudojant C++11 kompiliatorių, gali ir neveikti.

> "*C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.*" [Stroustrup, 2013]

C++11 ir C++98 suderinamumas

- C++11 buvo kuriamas taip, kad išliktų pilnai suderinamas su C++98.
- Iš principo, jeigu programa veikė (kompiliavosi) su C++98 ar C++03 standartais, turėtų veikti ir su C++11. Tačiau yra kelios išimtys
  - Kintamieji daugiau negali turėti naujai įvestų raktinių žodžių (**keywords**).
- Atgalinis suderinamumas (**backward compatibility**) taikomas tik programos kodui (**source code**).
- Todėl programas, parengtas naudojant C++98 standartą, turėtume be problemų sukompiliuoti naudojant ir C++11 kompiliatorių.
- Tačiau sukompiliuoto kodo su C++98 kompiliatoriumi apjungimas (**linking**) naudojant C++11 kompiliatorių, gali ir neveikti.

> "*C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.*" [Stroustrup, 2013]

C++11 ir C++98 suderinamumas

- C++11 buvo kuriamas taip, kad išliktų pilnai suderinamas su C++98.
- Iš principo, jeigu programa veikė (kompiliavosi) su C++98 ar C++03 standartais, turėtų veikti ir su C++11. Tačiau yra kelios išimtys
  - Kintamieji daugiau negali turėti naujai įvestų raktinių žodžių (**keywords**).
- Atgalinis suderinamumas (**backward compatibility**) taikomas tik programos kodui (**source code**).
- Todėl programas, parengtas naudojant C++98 standartą, turėtume be problemų sukompiliuoti naudojant ir C++11 kompiliatorių.
- Tačiau sukompiliuoto kodo su C++98 kompiliatoriumi apjungimas (**linking**) naudojant C++11 kompiliatorių, gali ir neveikti.

> "*C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.*" [Stroustrup, 2013]

C++11 ir C++98 suderinamumas

- C++11 buvo kuriamas taip, kad išliktų pilnai suderinamas su C++98.
- Iš principo, jeigu programa veikė (kompiliavosi) su C++98 ar C++03 standartais, turėtų veikti ir su C++11. Tačiau yra kelios išimtys
  - Kintamieji daugiau negali turėti naujai įvestų raktinių žodžių (**keywords**).
- Atgalinis suderinamumas (**backward compatibility**) taikomas tik programos kodui (**source code**).
- Todėl programas, parengtas naudojant C++98 standartą, turėtume be problemų sukompiliuoti naudojant ir C++11 kompiliatorių.
- Tačiau sukompiliuoto kodo su C++98 kompiliatoriumi apjungimas (**linking**) naudojant C++11 kompiliatorių, gali ir neveikti.

> "*C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.*" [Stroustrup, 2013]

C++11 ir C++98 suderinamumas

- C++11 buvo kuriamas taip, kad išliktų pilnai suderinamas su C++98.
- Iš principo, jeigu programa veikė (kompiliavosi) su C++98 ar C++03 standartais, turėtų veikti ir su C++11. Tačiau yra kelios išimtys
  - Kintamieji daugiau negali turėti naujai įvestų raktinių žodžių (**keywords**).
- Atgalinis suderinamumas (**backward compatibility**) taikomas tik programos kodui (**source code**).
- Todėl programas, parengtas naudojant C++98 standartą, turėtume be problemų sukompiliuoti naudojant ir C++11 kompiliatorių.
- Tačiau sukompiliuoto kodo su C++98 kompiliatoriumi apjungimas (**linking**) naudojant C++11 kompiliatorių, gali ir neveikti.

"*C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.*" [Stroustrup, 2013]

Naujos C++11 kalbos galimybės

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Kas naujo ir svarbaus atsirado C++11 lyginant C++98?

- Kalbos ir standartinės bibliotekos priemonės, skirtos lygiagrečiai programuoti sistemas (pvz, naudojant kelių branduolių kompiuterius)
- Reguliarių išraiškų (regular expression) tvarkymas
- Išmaniosios rodyklės (smart pointers)
- Patobulintas (pseudo)atsitiktinių skaičių generavimas
- Patobulinti konteineriai (įskaitant, hash lenteles)
- Bendroji initializacija
- Paprastesnė sintaksė for-ciklams
- Perkėlimo (move) semantika
- Unicode palaikymas
- Lambda funkcijos
- Bendrosios konstantinės išraiškos
- Variantiniai šablonai (variadic templates)

Smulkūs, bet svarbūs sintaksės patobulinimai

Pradėkime nuo dviejų smulkių, bet svarbių C++11 sintaksės patobulinimų.

Tarpai šabloninėse išraiškose
Dingo reikalavimas palikti tarpą tarp skliaustų šabloninėse išraiškose:

```
1 vector<list<int> >; // OK visose C++ versijose
2 vector<list<int>>;  // OK nuo C++11
```

`nullptr` ir `std::nullptr_t`

C++11 atsirado **naujas raktinis žodis** `nullptr` (kurio tipas `std::nullptr_t`, apibrėžtas `<cstddef>`) vietoj `0` ar `NULL` norint pasakyti, kad rodyklė neturi reikšmės (kas gerokai skiriasi nuo neapibrėžtos reikšmės (**undefined value**)). Tai padeda išvengi klaidų situacijose, kuriose **null** rodyklė buvo interpretuojama kaip sveikas skaičius:

Smulkūs, bet svarbūs sintaksės patobulinimai

Pradėkime nuo dviejų smulkių, bet svarbių C++11 sintaksės patobulinimų.

Tarpai šablonineše išraiškose
Dingo reikalavimas palikti tarpą tarp skliaustų šablonineše išraiškose:

```
1 vector<list<int> >; // OK visose C++ versijose
2 vector<list<int>>;  // OK nuo C++11
```

`nullptr` ir `std::nullptr_t`

C++11 atsirado **naujas raktinis žodis** `nullptr` (kurio tipas `std::nullptr_t`, apibrėžtas `<cstddef>`) vietoj `0` ar `NULL` norint pasakyti, kad rodyklė neturi reikšmės (kas gerokai skiriasi nuo neapibrėžtos reikšmės (**undefined value**)). Tai padeda išvengi klaidų situacijose, kuriose **null** rodyklė buvo interpretuojama kaip sveikas skaičius:

Smulkūs, bet svarbūs sintaksės patobulinimai

Pradėkime nuo dviejų smulkių, bet svarbių C++11 sintaksės patobulinimų.

Tarpai šabloninėse išraiškose
Dingo reikalavimas palikti tarpą tarp skliaustų šabloninėse išraiškose:

```
1 vector<list<int> >; // OK visose C++ versijose
2 vector<list<int>>;  // OK nuo C++11
```

`nullptr` ir `std::nullptr_t`

C++11 atsirado **naujas raktinis žodis** `nullptr` (kurio tipas `std::nullptr_t`, apibrėžtas `<cstddef>`) vietoj `0` ar `NULL` norint pasakyti, kad rodyklė neturi reikšmės (kas gerokai skiriasi nuo neapibrėžtos reikšmės (**undefined value**)). Tai padeda išvengi klaidų situacijose, kuriose **null** rodyklė buvo interpretuojama kaip sveikas skaičius:

```
1 void f(int);
2 void f(void*);
3 f(0);         // calls ?
4 f(NULL);      // calls ?
5 f(nullptr);   // calls ?
```

Smulkūs, bet svarbūs sintaksės patobulinimai

Pradėkime nuo dviejų smulkių, bet svarbių C++11 sintaksės patobulinimų.

Tarpai šabloninėse išraiškose
Dingo reikalavimas palikti tarpą tarp skliaustų šabloninėse išraiškose:

```
1 vector<list<int> >; // OK visose C++ versijose
2 vector<list<int>>;  // OK nuo C++11
```

`nullptr` ir `std::nullptr_t`

C++11 atsirado **naujas raktinis žodis** `nullptr` (kurio tipas `std::nullptr_t`, apibrėžtas `<cstddef>`) vietoj `0` ar `NULL` norint pasakyti, kad rodyklė neturi reikšmės (kas gerokai skiriasi nuo neapibrėžtos reikšmės (**undefined value**)). Tai padeda išvengi klaidų situacijose, kuriose **null** rodyklė buvo interpretuojama kaip sveikas skaičius:

```
1 void f(int);
2 void f(void*);
3 f(0);         // calls f(int)
4 f(NULL);      // calls f(int) jeigu NULL is 0, neapibrėžta priešingu
5 f(nullptr);   // calls f(void*)
```

Automatinis tipo nustatymas su `auto`

C++11 galima **deklaruoti kintamąjį** ar objektą **nenurodant jo tipo**:

```
1 auto i = 42; // i tipas int
2 double f();
3 auto d = f(); // d tipas double
```

Deklaruoto su auto **kintamojo tipoas** yra nustatymas **iš inicializuotos reikšmės**:

```
1 auto i; // ERROR: negalima nustatyti i tipo
```

Papildomi raktažodžiai yra leidžiami:

```
1 static auto vat = 0.19;
```

auto yra ypatingai naudingas, kai susiduriame su ilgomis išraiškomis:

Automatinis tipo nustatymas su `auto`

C++11 galima **deklaruoti kintamąjį** ar objektą **nenurodant jo tipo**:

```
1 auto i = 42; // i tipas int
2 double f();
3 auto d = f(); // d tipas double
```

Deklaruoto su auto **kintamojo tipoas** yra nustatymas **iš inicializuotos reikšmės**:

```
1 auto i; // ERROR: negalima nustatyti i tipo
```

Papildomi raktažodžiai yra leidžiami:

```
1 static auto vat = 0.19;
```

auto yra ypatingai naudingas, kai susiduriame su ilgomis išraiškomis:

Automatinis tipo nustatymas su `auto`

C++11 galima **deklaruoti kintamąjį** ar objektą **nenurodant jo tipo**:

```
1 auto i = 42; // i tipas int
2 double f();
3 auto d = f(); // d tipas double
```

Deklaruoto su auto **kintamojo tipoas** yra nustatymas **iš inicializuotos reikšmės**:

```
1 auto i; // ERROR: negalima nustatyti i tipo
```

Papildomi raktažodžiai yra leidžiami:

```
1 static auto vat = 0.19;
```

auto yra ypatingai naudingas, kai susiduriame su ilgomis išraiškomis:

Automatinis tipo nustatymas su `auto`

C++11 galima **deklaruoti kintamąjį** ar objektą **nenurodant jo tipo**:

```
1 auto i = 42; // i tipas int
2 double f();
3 auto d = f(); // d tipas double
```

Deklaruoto su auto **kintamojo tipoas** yra nustatymas **iš inicializuotos reikšmės**:

```
1 auto i; // ERROR: negalima nustatyti i tipo
```

Papildomi raktažodžiai yra leidžiami:

```
1 static auto vat = 0.19;
```

auto yra ypatingai naudingas, kai susiduriame su ilgomis išraiškomis:

```
1 vector<string> v;
2 auto pos = v.begin(); // pos tipas yra ?
```

Automatinis tipo nustatymas su `auto`

C++11 galima **deklaruoti kintamąjį** ar objektą **nenurodant jo tipo**:

```
auto i = 42; // i tipas int
double f();
auto d = f(); // d tipas double
```

Deklaruoto su auto **kintamojo tipoas** yra nustatymas **iš inicializuotos reikšmės**:

```
auto i; // ERROR: negalima nustatyti i tipo
```

Papildomi raktažodžiai yra leidžiami:

```
static auto vat = 0.19;
```

auto yra ypatingai naudingas, kai susiduriame su ilgomis išraiškomis:

```
vector<string> v;
auto pos = v.begin(); // pos tipas yra vector<string>::iterator
```

Bendroji inicializacija ir inicializavimo sąrašai (1)

- Prieš C++11, buvo lengva susipainioti, kaip iš tiesų reikia **inicializuoti** kintamąjį ar objektą. Inicializuoti galima naudojant **skliaustus ()**, **skliaustus {}**, **ir/arba priskyrimo operatorių =**.
- Todėl C++11 įvedė bendrosios inicializacijos (**uniform initialization**) koncepsiją, kuria reiškia kad viskam inicializuoti galima naudoti tą pačią sintaksė naudojančią {} **skliaustus**:

```
int values[] { 1, 2, 3 };
std::vector<int> v { 2, 3, 5, 7, 11, 13, 17 };
```

- Inicializavimo sąrašai (**initializer list**) atlieka taip vadinamą reikšmių inicializavimą (**value initialization**), kuris reiškia, kad kiekvienas **bazinio tipo kintamasis**, kuris tradiciškai yra neapibrėžtas (**undefined initial value**), yra inicializuojamas **0** (arba nullptr, jeigu tai rodyklė):

```
int i;     // i neapibrėžta reikšmė
int j{};   // j = 0
int* p;    // p neapibrėžta reikšmė
int* q{}; // q = nullptr
```

Bendroji inicializacija ir inicializavimo sąrašai (1)

- Prieš C++11, buvo lengva susipainioti, kaip iš tiesų reikia **inicializuoti** kintamąjį ar objektą. Inicializuoti galima naudojant **skliaustus ()**, **skliaustus {}**, **ir/arba priskyrimo operatorių =**.
- Todėl C++11 įvedė bendrosios inicializacijos (**uniform initialization**) koncepsiją, kuria reiškia kad viskam inicializuoti galima naudoti tą pačią sintaksė naudojančią {} **skliaustus**:

```
1 int values[] { 1, 2, 3 };
2 std::vector<int> v { 2, 3, 5, 7, 11, 13, 17 };
```

- Inicializavimo sąrašai (**initializer list**) atlieka taip vadinamą reikšmių inicializavimą (**value initialization**), kuris reiškia, kad kiekvienas **bazinio tipo kintamasis**, kuris tradiciškai yra neapibrėžtas (**undefined initial value**), yra inicializuojamas **0** (arba nullptr, jeigu tai rodyklė):

```
1 int i;     // i neapibrėžta reikšmė
2 int j{};   // j = 0
3 int* p;    // p neapibrėžta reikšmė
4 int* q{}; // q = nullptr
```

Bendroji inicializacija ir inicializavimo sąrašai (1)

- Prieš C++11, buvo lengva susipainioti, kaip iš tiesų reikia **inicializuoti** kintamąjį ar objektą. Inicializuoti galima naudojant **skliaustus ()**, **skliaustus {}**, **ir/arba priskyrimo operatorių =**.
- Todėl C++11 įvedė bendrosios inicializacijos (**uniform initialization**) koncepsiją, kuria reiškia kad viskam inicializuoti galima naudoti tą pačią sintaksė naudojančią {} **skliaustus**:

```
1 int values[] { 1, 2, 3 };
2 std::vector<int> v { 2, 3, 5, 7, 11, 13, 17 };
```

- Inicializavimo sąrašai (**initializer list**) atlieka taip vadinamą reikšmių inicializavimą (**value initialization**), kuris reiškia, kad kiekvienas **bazinio tipo kintamasis**, kuris tradiciškai yra neapibrėžtas (**undefined initial value**), yra inicializuojamas **0** (arba nullptr, jeigu tai rodyklė):

```
1 int i;     // i neapibrėžta reikšmė
2 int j{};   // j = 0
3 int* p;    // p neapibrėžta reikšmė
4 int* q{}; // q = nullptr
```

Bendroji inicializacija ir inicializavimo sąrašai (2)

- Pažymėtina, kad siaurinančioji (tikslumą mažinanti) inicializacija (**narrowing initializations**) yra **neleidžiama** naudojant {} skliaustus:

```
int x1(5.3);     // OK, bet OUCH: x1 = 5
int x2 = 5.3;    // OK, bet OUCH: x2 = 5
int x3{5.0};     // ERROR: siaurinanti inicializacija
int x4 = {5.3}; // ERROR: siaurinanti inicializacija
char c1{7};      // OK: šiuo atveju 7 tampa char simboliu
char c2{99999}; // ERROR: siaurinanti (kai 99999 netelpa į char tipą)
std::vector<int> v1 {1, 2, 4, 5};     // OK
std::vector<int> v2 {1, 2.3, 4, 5.6}; // ERROR: siaurinanti inicializacija
```

- Problemos, kylančios dėl kintamųjų reikšmių susiaurėjimo (kaip pvz. iš double į int, ar iš int į char) atsiranda dėl C++ suderinamumo C kalba.

Diapazoniniai (range-based) `for` ciklai

- C++11 atsirado nauja for ciklų forma, kurioje yra perenkami **visi elementai iš nurodytos srities, masyvo, kolekcijos** (`foreach` analogas):

```
for ( decl : coll ) { statement; }
```

kur `decl` yra deklaracija kiekvieno elemento iš kolekcijos `coll`.

- Pvz. žemiau esantis ciklas perrenka ir atspausdina (į standartinį išvedimą cout) visus elementus esančius pateiktame sąraše:

```
for ( int i : { 2, 3, 5, 7, 9, 13, 17, 19 } ) {
  std::cout << i << std::endl;
}
```

- Norint padauginti kiekvieną vektoriaus **vec** elementą **elem** iš 3:

```
std::vector<int> vec { 2, 3, 5, 7, 9, 13, 17, 19 };
for ( auto& elem : vec ) {
  elem *= 3;
}
```

Šiuo atveju yra svarbu deklaruoti `elem` kaip nuorodą (**reference**); priešingu atveju for ciklas vyktų naudojant lokalias vektoriaus elementų kopijas (kas irgi gali būti naudinga, tik kitame kontekste).

Diapazoniniai (range-based) `for` ciklai

- C++11 atsirado nauja for ciklų forma, kurioje yra perenkami **visi elementai iš nurodytos srities, masyvo, kolekcijos** (`foreach` analogas):

```
for ( decl : coll ) { statement; }
```

kur `decl` yra deklaracija kiekvieno elemento iš kolekcijos `coll`.

- Pvz. žemiau esantis ciklas perrenka ir atspausdina (į standartinį išvedimą cout) visus elementus esančius pateiktame sąraše:

```
for ( int i : { 2, 3, 5, 7, 9, 13, 17, 19 } ) {
  std::cout << i << std::endl;
}
```

- Norint padauginti kiekvieną vektoriaus **vec** elementą **elem** iš 3:

```
std::vector<int> vec { 2, 3, 5, 7, 9, 13, 17, 19 };
for ( auto& elem : vec ) {
  elem *= 3;
}
```

Šiuo atveju yra svarbu deklaruoti `elem` kaip nuorodą (**reference**); priešingu atveju for ciklas vyktų naudojant lokalias vektoriaus elementų kopijas (kas irgi gali būti naudinga, tik kitame kontekste).

Diapazoniniai (range-based) `for` ciklai

- C++11 atsirado nauja for ciklų forma, kurioje yra perenkami **visi elementai iš nurodytos srities, masyvo, kolekcijos** (`foreach` analogas):

```
for ( decl : coll ) { statement; }
```

kur `decl` yra deklaracija kiekvieno elemento iš kolekcijos `coll`.

- Pvz. žemiau esantis ciklas perrenka ir atspausdina (į standartinį išvedimą cout) visus elementus esančius pateiktame sąraše:

```
for ( int i : { 2, 3, 5, 7, 9, 13, 17, 19 } ) {
  std::cout << i << std::endl;
}
```

- Norint padauginti kiekvieną vektoriaus **vec** elementą **elem** iš 3:

```
std::vector<int> vec { 2, 3, 5, 7, 9, 13, 17, 19 };
for ( auto& elem : vec ) {
  elem *= 3;
}
```

Šiuo atveju yra svarbu deklaruoti `elem` kaip nuorodą (**reference**); priešingu atveju for ciklas vyktų naudojant lokalias vektoriaus elementų kopijas (kas irgi gali būti naudinga, tik kitame kontekste).

Klausimai?

Literatūros šaltiniai


Group of C++ enthusiasts (2000).
C++ reference.
http://en.cppreference.com/w/cpp/language/history.
[Online; accessed 19-01-2018].


Stroustrup, B. (2013).
The C++ programming language.
Addison-Wesley, 4 edition.