Objektinis Programavimas

Įvadas

Vilnius University

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

Turinys

1. Dalyko (modulio) apžvalga
   - Dalyko (modulio) tikslas
   - Temos
   - Vertinimo strategija
   - Rekomenduojama literatūra
2. Kodėl C++?
   - C++ istorija
3. Naujos C++11 standarto galimybės
   - Smulkūs, bet svarbūs sintaksės patobulinimai
   - Automatinis tipo nustatymas su `auto`
   - Bendroji inicializacija ir inicializavimo sąrašai
   - Diapazoniniai (angl. range-based) `for` ciklai

Dalyko (modulio) tikslas

- **Dalyko (modulio) tikslas**: siekiama, kad studentai susipažintų su objektinio programavimo (OP) koncepcija ir ugdytų praktinius gebėjimus kurti efektyvias, objektiškai orientuotas programas.
- **Dalyko (modulio) studijų siekiniai:**
  - Gebės naudoti kolektyvinio ir kodo versijavimo sistemas atliekant objektinio programavimo užduotis
  - Gebės paaiškinti objektinio programavimo sąvokas, išmanys jų taikymo sritis.
  - Gebės įvertinti objektiškai orientuotos programinės įrangos kūrimo tikslus, taikyti automatinius
    programinio kodo dokumentavimo įrankius.
  - Gebės taikyti pagrindinius objektinio programavimo architektūrinio stiliaus principus užduočių sprendime.
  - Gebės parinkti efektyvius algoritmus ir į spartą orientuotas (priklausomai nuo uždavinio specifikos) duomenų struktūras, pritaikyti objektiškai orientuotų sistemų projektavimo ir įgyvendinimo žinias sprendžiant užduotis.

Temos (1)

Tema	Paskaitos / Lab. darbai (kont. val.)
1. Kurso apžvalga, C++ standartai, kas yra objektiškai orientuotas programavimas (OOP)?, objekto koncepcija, pažintis su programavimo aplinka: kompiliatorių ir įrankių apžvalga, versijų kontrolės sistemos (git), make/cmake įrankiai, „unit“-testai.	6 / 4
2. Baziniai duomenų tipai, tipų transformacija, l-reikšmės ir r-reikšmės, rodyklės ir nuorodos, konstantinės nuorodos, dinaminis atminties valdymas, aritmetika su adresais, “išmaniosios” rodyklės, funkcijų persidengimas, direktyvos, įvesties ir išvesties operatoriai.	8 / 8
3. Vartotojo tipai, klasės ir objektai, struktūros, konstruktoriai, pagrindinis konstruktorius, destruktoriai, objektinis projektavimas, „RAII“ paradigma, inkapsuliavimas, matomumo kontrolė, UML diagramos, dokumentacijos kūrimas su `Doxygen`.	8 / 8

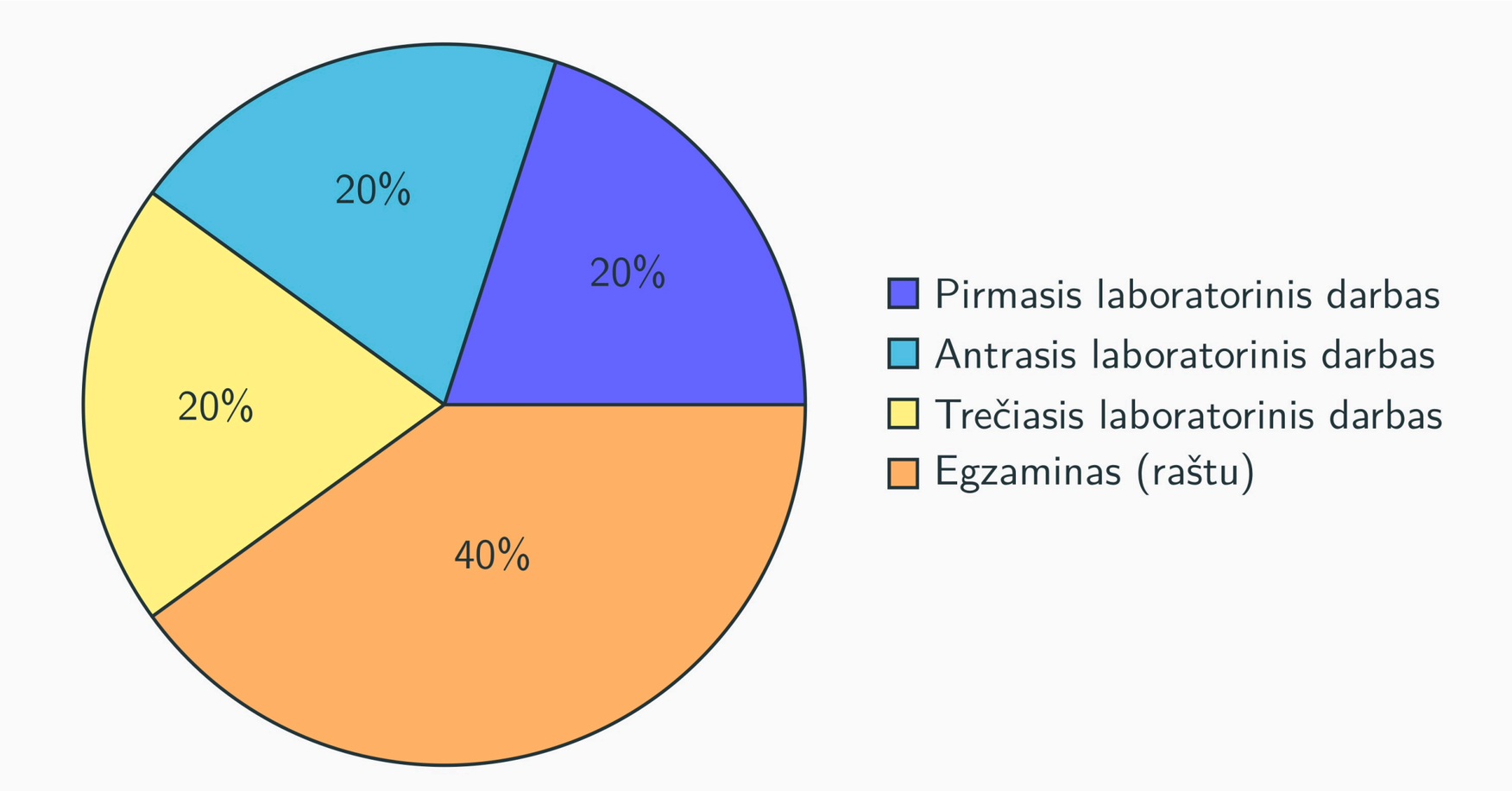
Temos (2)

Tema	Paskaitos / Lab. darbai (kont. val.)
4. Operatorių persidengimas, įvesties/išvesties operatoriai, operatorių persidengimo realizavimo strategijos, kopijavimo konstruktorius, priskyrimo ir kopijavimo konstruktoriaus palyginimas, seklus ir gilus kopijavimas, nuorodos į r-reikšmes, objektų perkėlimo semantika.	8 / 8
5. Kompozicija ir agregavimas, paveldėjimas, paveldėjimo kontrolė, konstruktoriai ir paveldėjimas, polimorfizmas, virtualiosios funkcijos, ankstyvas ir vėlyvas saistymas (angl. binding).	8 / 6
6. Standartiniai išvesties ir įvesties srautai, failų srautai.	6 / 6

Temos (3)

Tema	Paskaitos / Lab. darbai (kont. val.)
7. Klaidų ir išimčių valdymas, laiko matavimas (`std::chrono` biblioteka), programos spartos matavimo įrankiai, atsitiktinių skaičių generavimas.	6 / 8
8. Objektiškai orientuotas dizainas, interfeisai, bendrinis programavimas, šablono klasės ir funkcijos.	7 / 8
9. Konteineriai (vektorius, dekas, sąrašas, žemėlapiai, hash lentelės ir t.t.), iteratoriai. Algoritmų apžvalga: efektyvumas su algoritmais, sparta su duomenų struktūromis.	7 / 8
10. Pasiruošimas egzaminui ir egzamino laikymas.	2* / 0

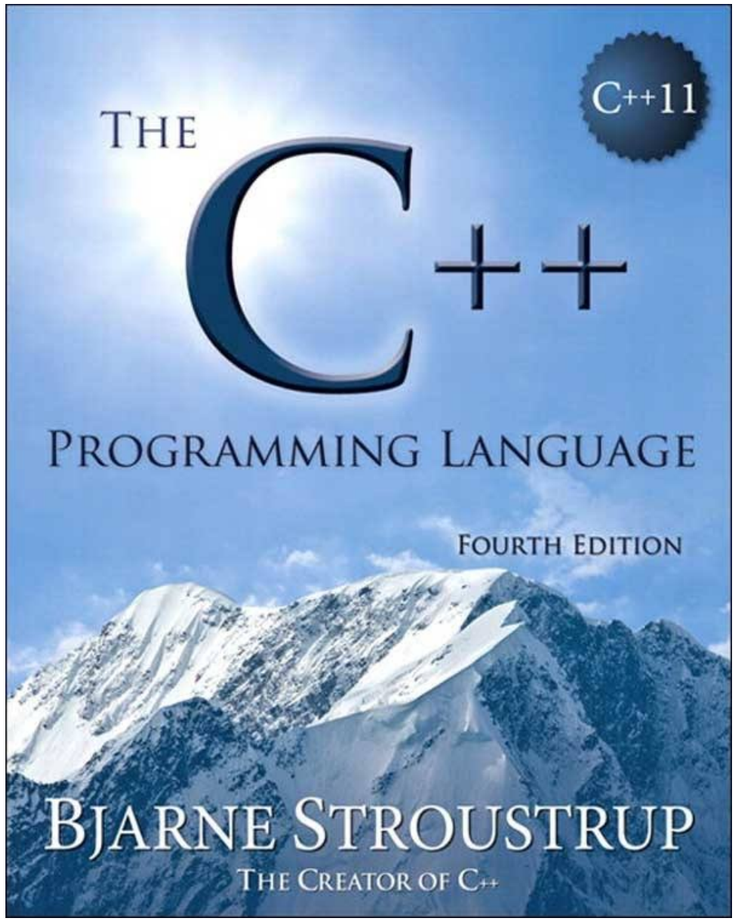
Vertinimo strategija (1)

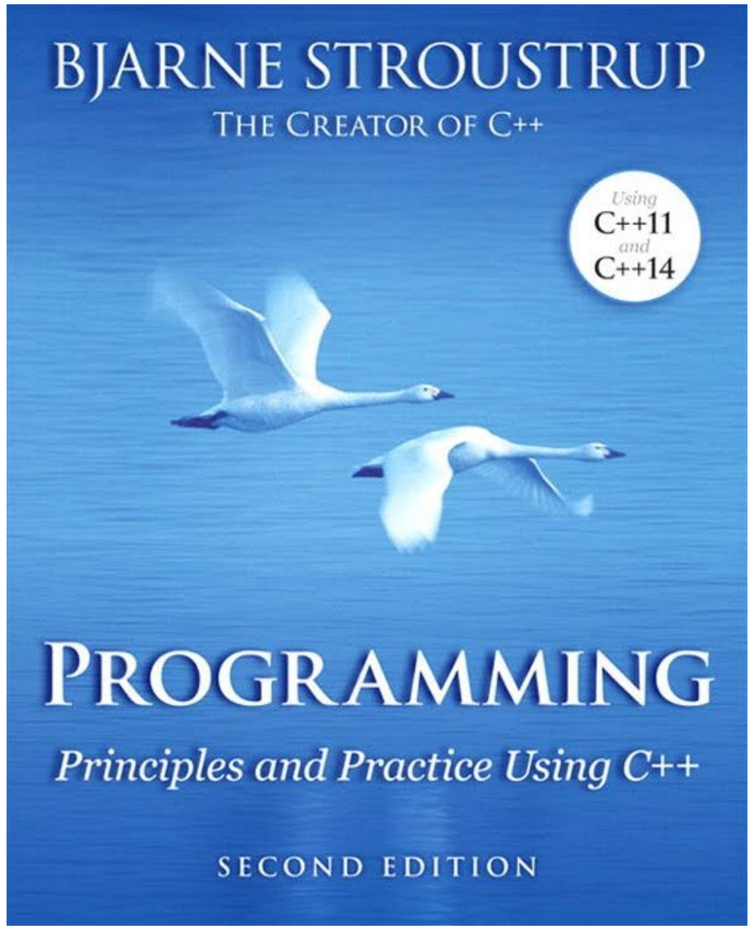


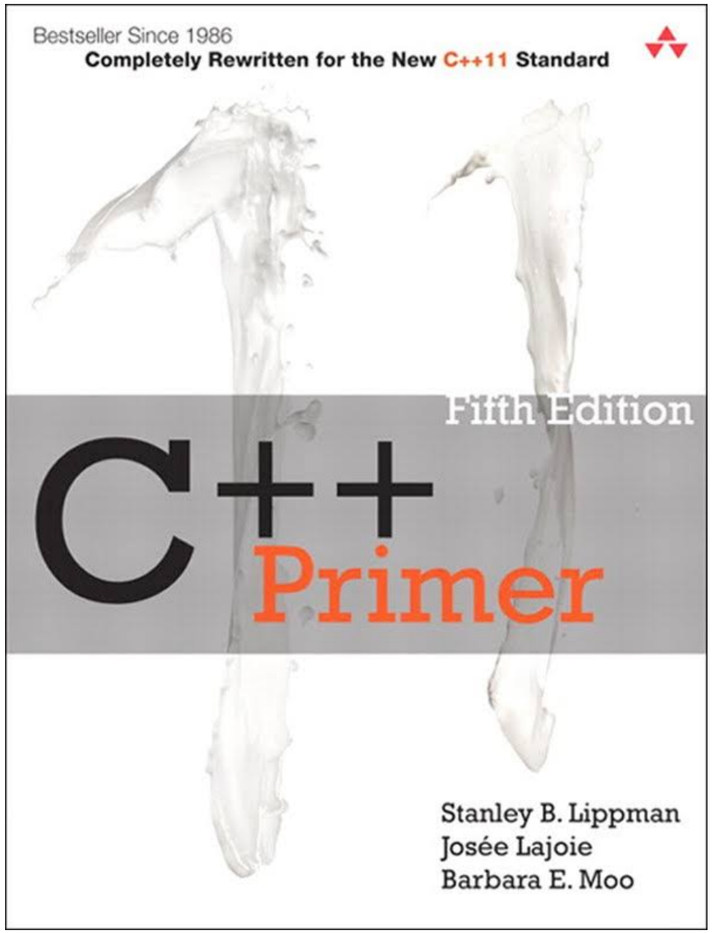
Vertinimo strategija (2)

Vertinimo strategija	Vertinimo kriterijai
Pirmasis laboratorinis darbas	Studentams skiriamos individualios užduotys, apimančios 1-3 temas. Maksimalus įvertinimas už puikiai atliktas užduotis yra 10 balų (atitinkantys 20 % bendrojo svorio). Skiriami papildomi balai (iki 20 % maksimalaus įverčio svorio), jei užduotys atsiskaitomos anksčiau nurodyto termino. Analogiškai, vėluojant atsiskaityti galutinis įvertinimas yra mažinamas (iki 20 % maksimalaus įverčio svorio).
Antrasis laboratorinis darbas	Studentams skiriamos individualios užduotys, apimančios 4-6 temas. Maksimalus įvertinimas už puikiai atliktas užduotis yra 10 balų (atitinkantys 20% bendrojo svorio). Skiriami papildomi balai (iki 20 % maksimalaus įverčio svorio), jei užduotys atsiskaitomos anksčiau nurodyto termino. Analogiškai, vėluojant atsiskaityti galutinis įvertinimas yra mažinamas (iki 20 % maksimalaus įverčio svorio).
Trečiasis laboratorinis darbas	Studentams skiriamos individualios užduotys, apimančios 7-9 temas. Maksimalus įvertinimas už puikiai atliktas užduotis yra 10 balų (atitinkantys 20% bendrojo svorio). Skiriami papildomi balai (iki 20 % maksimalaus įverčio svorio), jei užduotys atsiskaitomos anksčiau nurodyto termino. Analogiškai, vėluojant atsiskaityti galutinis įvertinimas yra mažinamas (iki 20 % maksimalaus įverčio svorio).
Egzaminas (raštu)	Egzaminą laikyti leidžiama semestro metu surinkus ne mažiau 7.5 balų skaičių, atitinkantį 25% laboratoriniams darbams skirtojo svorio. Egzamino metu galima surinkti iki 10 taškų, kurie atitinka 40% galutinio įvertinimo. Egzaminas susideda iš dviejų etapų. Pirmiausia, studentas turi atsakyti į klausimus iš paskaitose pateiktų temų (iki 2 taškų). Antroje egzamino dalyje studentas turi pateikti praktinį pateiktos problemos sprendimą C++ kalboje (iki 8 taškų), motyvuojant naudojamų priemonių efektyvumą bei analizuojant alternatyvius užduoties sprendimo būdus.

Privaloma literatūra[1] (1)

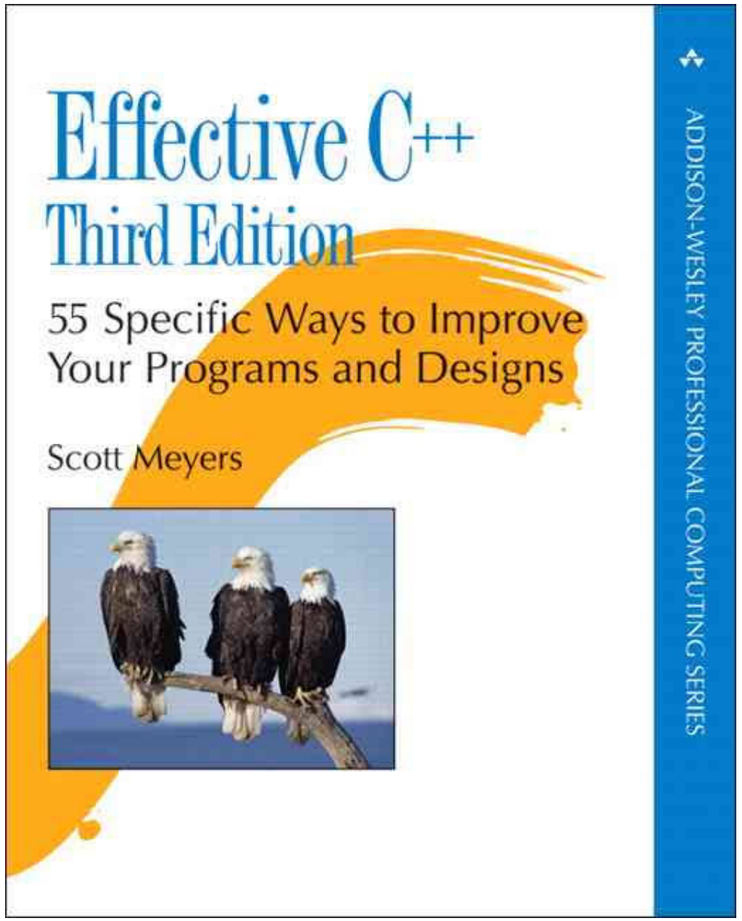


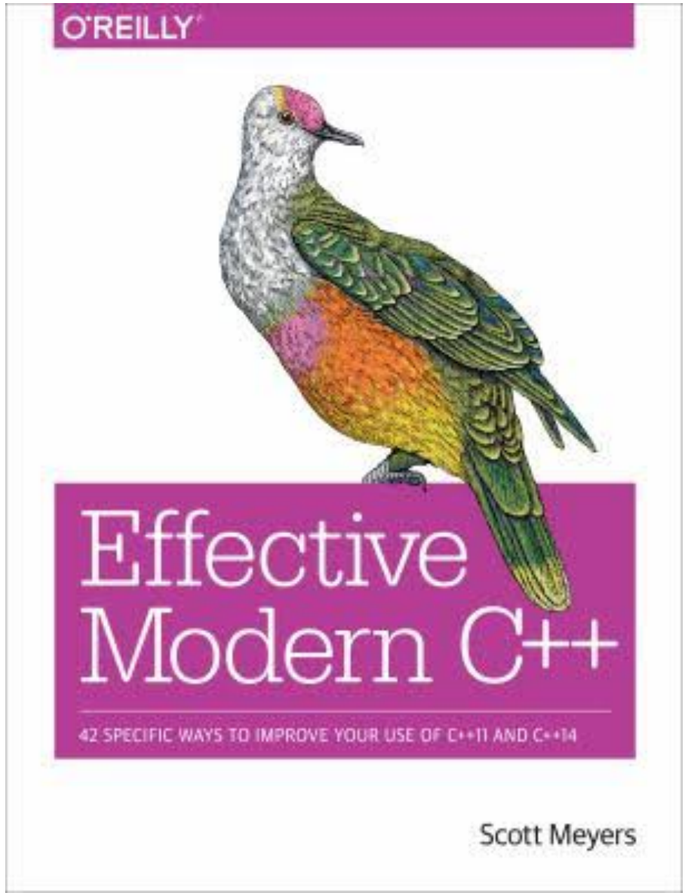


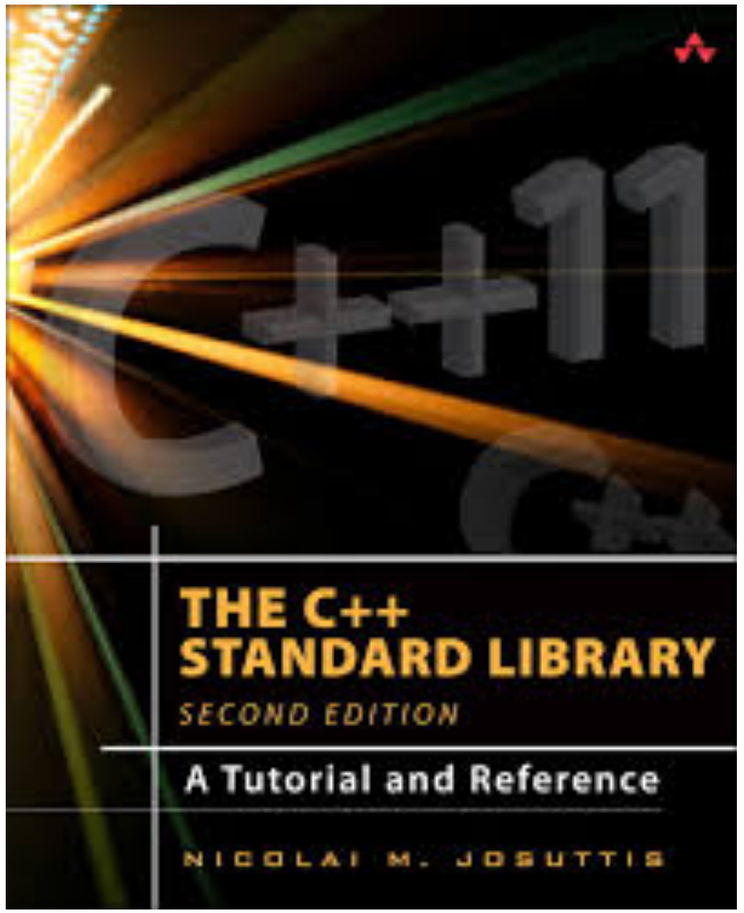


[1] Paremta: The Definitive C++ Book Guide and List

Papildoma literatūra (2)







Kodėl C++?

TIOBE indeksas, 2019 m. sausis

Jan 2019	Jan 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.904%	+2.69%
2	2		C	13.337%	+2.30%
3	4	↑	Python	8.294%	+3.62%
4	3	↓	C++	8.158%	+2.55%
5	7	↑	Visual Basic .NET	6.459%	+3.20%
6	6		JavaScript	3.302%	-0.16%
7	5	↓	C#	3.284%	-0.47%
8	9	↑	PHP	2.680%	+0.15%
9	-	⇈	SQL	2.277%	+2.28%
10	16	⇈	Objective-C	1.781%	-0.08%
11	18	⇈	MATLAB	1.502%	-0.15%
12	8	⇊	R	1.331%	-1.22%
13	10	↓	Perl	1.225%	-1.19%
14	15	↑	Assembly language	1.196%	-0.86%
15	12	↓	Swift	1.187%	-1.19%
16	19	↑	Go	1.115%	-0.45%
17	13	⇊	Delphi/Object Pascal	1.100%	-1.28%

Apklausa dėl programavimo kalbos

https://goo.gl/R5BBk7

C++ istorija[2] (1)

— **1979**: Pirminė C++ 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 assign- ment 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.

[2] https://en.cppreference.com/w/cpp/language/history

C++ istorija (2)

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

  — **Naujos galimybės:** “RTTI (dynamic_cast, typeid), covariant return types, cast operators, mutable, bool, declarations in conditions, template instantia- tions, member templates, export”

  — **Naujos bibliotekos (Library) galimybės:** “containers, algorithms, iterators, function objects (based on STL), locales, bitset, valarray, auto_ptr, templa- tized string, iostream, and complex.”

C++ istorija (3)

— **1999**: Komiteto nariai įkuria **Boost**[b], 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 erdvėje (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).

[b] http://www.boost.org/

C++ istorija (4)

— **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 (1)

- `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:
  - Kintamieji daugiau negali būti pavadinti naujai įvestais raktiniais žodžiais (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.

C++11 ir C++98 suderinamumas (2)

— 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.” **B. Stroustrup**

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

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

Smulkūs, bet svarbūs sintaksės patobulinimai

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

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

— `nullptr` ir `std::nullptr_t`: C++11 atsirado naujas raktinis žodis nullptr (tipas `std::nullptr_t`, apibrėžtas `<cstddef>`) ir naudojamas vietoj 0 ar `NULL` kai rodyklė neturi reikšmės. Tai padeda išvengi klaidų situacijose, kuriose `null` rodyklė buvo interpretuojama kaip sveikas skaičius:

```
void f(int);
void f(void*);
f(0);       // calls ?
f(NULL);    // calls ?
f(nullptr); // calls ?
```

Smulkūs, bet svarbūs sintaksės patobulinimai

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

```
vector<list<int> >; // OK visose C++ versijose
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` kai rodyklė neturi reikšmės (kas gerokai skiriasi nuo neapibrėžtos reikšmės (*undefined value*)). Tai padeda išvengi klaidų situacijose, kuriose `NULL` rodyklė buvo interpretuojama kaip sveikas skaičius:

```
void f(int);
void f(void*);
f(0);       // calls f(int)
f(NULL);    // calls f(int) jeigu NULL is 0, neapibrėžta priešingu atveju
f(nullptr); // calls f(void*)
```

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 tipas yra nustatomas iš inicializuotos reikšmės:

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

— Papildomi raktažodžiai yra leidžiami:

```
static auto vat = 0.21;
```

— `auto` yra ypatingai naudingas, kai susiduriame su ilgomis išraiškomis:

```
vector<string> v;
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 tipas yra nustatomas iš inicializuotos reikšmės:

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

— Papildomi raktažodžiai yra leidžiami:

```
static auto vat = 0.21;
```

— `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, kuri reiškia kad viskam inicializuoti galima naudoti tą pačią sintaksę panaudojant {} 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 rodyklė):

```
int i;    // i neapibrėžta reikšmė
int j{};  // j=0
int* p;   // p neapibrėžta reikšmė
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 inic.
```

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

Diapazoniniai (range-based**) for ciklai (1)**

— 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;
}
```

Diapazoniniai (range-based**) for ciklai (2)**

— Padauginkime 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).

Konstantinės išraiškos (constexpr**)**

const

nusako, kad reikšmė yra nekintanti (konstanta), ir ji gali būti inicializuota tiek kompiliavimo metu, tiek ir programos paleidimo (run time) metu;

constepr

nusako, kad reikšmė yra nekintanti (konstanta), ir ji privalo būti inicializuota kompiliavimo metu;

Klausimai !?

`C` makes it easy to shoot yourself in the foot; `C++` makes it harder, but when you do it blows your whole leg off.