

# Objektinis Programavimas

## 2-oji praktinė užduotis



# Duomenų apdorojimas

Turinys

- 👉 Užduoties formuluotė
- 👉 Reikalavimai versijai (v0.1)

# Praktinės užduoties formuliuotė

- ☞ Parašykite programą, kuri nuskaito šiuos studentų duomenis:
  - ☞ **vardą** ir **pavardę**
  - ☞ n atliktų **namų darbų** (nd) rezultatus (10-baleje sistemoje), o taip pat galutinio **egzamino** (egz) rezultatą.
- ☞ Tuomet iš šių duomenų, suskaičiuoja galutinių balų (galutinis):

$$\text{galutinis} = 0.4 \times \frac{\sum_{i=1}^n \text{nd}_i}{n} + 0.6 \times \text{egz}$$

# Reikalavimai (v0.1) versijai (1)

- ☞ Pagal užduoties reikalavimus realizuokite programą ir atspausdinkite ekrane aktualią informaciją:
  - ☞ studento vardą ir pavardę,
  - ☞ namų darbų ir egzamino rezultatus
  - ☞ bei galutinį balą **dviejų skaičių po kablelio tikslumu**
- ☞ Papildykite programą, kad vietoj **vidurkio** būtų galima naudoti **medianą**.

## **Reikalavimai (v0.1) versijai (2)**

- ☞ Papildykite programą taip, kad ji veiktu ir tokiu atveju, kai namų darbų skaičius (n) yra nežinomas iš anksto, t.y. tik įvedimo metu vartotojas nusprendžia kuomet jis jau įvedė visų namų darbų rezultatus. Šią užduotį realizuokite dviem būdais:
  - ☞ naudojant C masyvus.
  - ☞ naudojant `vector` ar kito tipo konteinerių.
- ☞ Papildykite, kad būtų galimybė, jog mokinio gautieji balai už namų darbus bei egzaminą būtų generuojami atsitiktinai.

# **Reikalavimai (v0.2) versijai (1)**

**Terminas: 2018-02-27**

- ☞ Papildykite (v0.1) taip, kad būtų galima duomenis ne tik įvesti bet ir nuskaityti iš failų, t.y., sukurkite ir užpildykite failą **kursiokai.txt**, kurio (pleriminari) struktūra:

Pavardė	Vardas	ND1	ND2	ND3	ND4	ND5	Egzaminas
Pocius	Paulius	8	9	10	6	10	9
Makevičius	Augustinas	7	10	8	5	4	6
...							

# **Reikalavimai (v0.2) versijai (2)**

**Terminas: 2018-02-27**

- ☞ Papildykite programą taip, nuskaičiuos duomenis iš failo, galutinis rezultatas (output'as) pleriminariai atrodytų taip:

Pavardė	Vardas	Gal.-vidurkis	Gal.-mediana
Makevičius	Augustinas	x . xx	y . yy
Pocius	Paulius	z . zz	q . qq
...			

**Reikalavimai output'ui:** studentai turi būtui surūšiuoti pagal vardus (ar pavardes) ir stulpeliai būtų gražiai "išlygiuoti".

# Reikalavimai (v0.3) versijai (1)

**Terminas: 2018-03-06**

- ☞ Atlikite versijos (v0.2) kodo reorganizavimą (refactoring'ą):
  - ☞ Kur tikslina, programoje naudokite (jeigu dar nenaudojote) struct'ūras
  - ☞ Funkcijas, naujus duomenų tipus (struct'ūras) perkelkite į antraštinius **(header** (\*.h)) failus, t.y. tokiu būdu turėtumete projekte turėti kelis \*.cpp failus, kaip ir kelis \*.h failus.

# Reikalavimai (v0.3) versijai (2)

**Terminas: 2018-03-06**

👉 Panaudokite išimčių valdymą (**Exception Handling**):

```
try { // išimtys yra apdorojamos žemiau
      // kodas, kuris atlieka tam tikras užduotis
} catch (std::exception& e) {
      // kodas, kuris apdoroja išimtis
}
```

**Kam viso to reikia?** Kas atsitiks, jei failas, kurį bandote atidaryti neegzistuoja; arba bandote gauti masyvo elementą, kurio nėra?

# Reikalavimai (v0.4) versijai (1)

**Terminas: 2018-03-13**

- 👉 Programos veikimo greičio (spartos) analizė:
- 👉 Sūrušiuokite (padalinkite) studentus į dvi kategorijas:
- 👉 Studentai, kurie **surinko < 60%** už namų darbų užduotis ("vargšiukai", "nuskriaustukai" ir pan.)
- 👉 Studentai, kurie **surinko >= 60%** ("kietiakai", "galvočiai", "vizunčikai" ir pan.) ir buvo prileisti prie egzamino.

# Reikalavimai versijai (v0.5)

Preliminarus terminas: 2018-03-15

☞ Išmatuokite programos veikimo spartą (be failų generavimo, nes visais atvejais **privalote** naudoti tuos pačius sugeneruotus studentų duomenų failus) priklausomai nuo naudojamo vieno iš trijų konteinerių:

☞ vector

☞ list

☞ deque

T.y., jeigu Jūs susikurėte struktūrą **Studentai** (ar kaip jūs ją pavadinote) ir iki šiol naudojote std::vector<Studentai>, tai turite ištirti ar pasikeistų ir kaip pasikeistų sparta, jei naudotumėte std::list<Studentai> ir std::deque<Studentai>.

# Skaičių po kablelio tikslumo valdymas

- ☞ <iomani> antraštės (header) faile deklaruotas manipulatorius setprecision, kuris leidžia kontroliuoti kiek reikšmingų skaitmenų išvedima informacija naudotu.
- ☞ Kai naudojame endl, kuris taip pat yra manipulatorius, mums nereikia ištraukti <iomani> antraštės. **Kodėl?**

# **std::setprecision() pavyzdys<sup>1</sup>**

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <limits>
int main() {
    const long double pi = std::acos(-1.L);
    std::cout << "default precision (6): " << pi << '\n'
        << "std::setprecision(10): " << std::setprecision(10) << pi << '\n'
        << "max precision: "
        << std::setprecision(std::numeric_limits<long double>::digits10 + 1) << pi << '\n';
}
```

default precision (6): 3.14159

std::setprecision(10): 3.141592654

max precision: 3.141592653589793239

---

<sup>1</sup> <http://en.cppreference.com/w/cpp/io/manip/setprecision>

# Versijos (v0.1) kūrimas (1)

## **Rezultatų spausdinimas 2 skaičių po kablelio tikslumu**

```
// išsaugome numatytaį (default) tikslumą
streamszie prec = cout.precision(); // C++11: auto prec
cout << "Galutinis balas yra " << setprecision(3)
      << 0.6 * egzaminas + 0.4 * suma / n
      << setprecision(prec) << endl;
```

# **EOF (end-of-file) signalas**

- ☞ Vienas iš būdų, kaip užbaigti duomenų įvedimą (cikle) yra **end-of-file (EOF)** signalo pasiuntimas.
- ☞ Skirtingos C++ realizacijos siūlo skirtingus būdus kaip tokį signalą į programą pasiūsti. Labiausiai paplitęs būdas yra:
  - ☞ pradėti naują eilutę (spaudžiame Enter)
  - ☞ tuomet spaudžiame Ctrl+z (Windows) arba Ctrl+d (Unix).

# **EOF (end-of-file) pavyzdžiai<sup>2</sup>**

```
while(!cin.eof()) { // EOF false čia
    cin >> x;          // kai nuskaityti nepavyksta, EOF tampa true
    // use x           // naudojame x, nors nieko nenuskaitėme
}
```

**VS.**

```
while(cin >> x) { // Bando nuskaityti į x, grąžina false kai nepavyksta
                  // ciklą vykdys tik kai nuskaityti pavyko
}
```

---

<sup>2</sup> <https://stackoverflow.com/questions/4533063/how-does-ifstreams-eof-work>

# **while (cin >> x) reiškinio analizė**

Norint nuskaityti reikšmę į **x** ir patikrinti ar pavyko:

```
if (cin >> x) { /*...*/ }
```

Tai ekvivalentu:

```
cin >> x;  
if (cin) { /* ... */ }
```

Todėl `cin` naudojimas **if** ar **while** sąlygose ekvivalentus: ar paskutinis bandymas nuskaityti iš `cin` buvo sėkmingas.

# Kada skaitymas iš srauto (cin) gali būti nesėkminges?

- 👉 Mes galėjome pasiekti įvesties failo pabaiga.
- 👉 Galbūt susidūrēme su įvestimi, nesuderinama su kintamojo tipu, kurį bandome perskaityti, pvz., taip gali atsitikti, jei mes bandome perskaityti int bet reikšmė yra nėra skaičius.
- 👉 Sistemoje gali būti aptikta įvesties įrangos gedimas.

Visais šiaiš atvejais: naudojant sąlygoje cin reikšmę bus false.

# Srauto (stream) būsenos (1)

Srautas turi būseną (konstantą), kuri nusako, ar I/O (**Input/Output**) buvo sėkminga, o jei ne - kokia nesėkmės priežastis.

Būsena	Reikšmė
goodbit	Viskas OK
eofbit	Pasiekta failo pabaiga
failbit	Klaida; I/O operacija nebuvo sėkminga
badbit	Esminė klaida; neapibrėžta būsena

## Srauto (stream) būsenos (2)

- ☞ failbit - reiškia, kad operacija nebuvo tinkamai apdorota, bet srautas yra OK. Pavyzdžiui, tai nutinka, jei skaitmuo turėjo būti nuskaitomas, bet gautas simbolis yra raidė.
- ☞ badbit - reiškia, kad srautas yra sugadintas arba ryšys su srautu yra prarastas.
- ☞ eofbit - tradiciškai nutinka kartu su failbit, nes failo pabaiga (end-of-file) nustato eofbit, bet tuo pačiu ir failbit, nes nieko nuskaityti nepavyko.

# Versijos (v0.1) kūrimas (2)

## **Nežinomo skaičiaus namų darbų (nd) nuskaitymas**

```
int n = 0;           // nd skaitiklis
double suma = 0.;    // nd įverčiu suma
double x;            // kintamasis iš kurį nuskaityti
/* invariantas: iki šiol nuskaitėme `n' nd rezultatų,
   ir juo įverčiu suma lygi `suma' */
while (cin >> x) {
    ++n;             // reikalauja pirmoji dalis invarianto
    suma += x;        // reikalauja antroji dalis invarianto
}
```

# Versijos (v0.1) kūrimas (3)

## **Nežinomo skaičiaus namų darbų (nd) nuskaitymas į vektorių**

```
// Aukščiau: using std::vector
vector<double> nd; // vektorius su namų darbų rezultatais
double x;           // kintamasis į kuri nuskaityti
// invariantas: `nd' kaupia visus iki šiol nuskaitytus ND
while (cin >> x) {
    nd.push_back(x); // Pridedame nuskaitytą rezultatą
}
```

# Versijos (v0.1) kūrimas (4)

## Apsauga nuo tuščio nd vektoriaus

```
// patikriname ar įvedė nd rezultatus
typedef vector<double>::size_type vecSize;
vecSize size = nd.size();
if (size == 0) { // `ekvivalentu': if (nd.empty())
    cout << "Privalote įvesti ND rezultatus."
    << "Bandykite iš naujo. \n";
    return 1;
}
```

👉 Kodėl `return 1;` ?

# Tipų apibrėžimai naudojant `typedef` ir `using keyword`'us

“A `typedef`-name can also be introduced by an alias-declaration. The identifier following the **using** keyword becomes a `typedef`-name and the optional attribute-specifier-seq following the identifier appertains to that `typedef`-name. It has the same semantics as if it were introduced by the **typedef** specifier. In particular, it does not define a new type and it shall not appear in the type-id.<sup>3</sup>”

---

<sup>3</sup> <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3690.pdf>

## **typedef ir using pavyzdžiai**

```
typedef int MyInt;
// ekvivalentu:
using MyInt = int; // Nuo C++11
// Tuomet:
MyInt x = 5;

// Patogu, kai ilgi tipai:
using IntVectorIt = std::vector<int>::iterator;
// Ekvivalentu:
typedef std::vector<int>::iterator IntVectorIt;
```

# **Medianos radimo procedūra**

- 👉 Norėdami apskaičiuoti medianą, privalome:
  - 👉 Išsaugoti nuskaitytas reikšmes, nežinant iš anksto, kiek reikšmių bus.
  - 👉 Kai visos reikšmės nuskaitytos - išrūšiuoti jas.
  - 👉 Gauti vidutinę(-es) reišmę(-es) efektyviai.

# Versijos (v0.1) kūrimas (5)

## Medianos radimas

```
// surūšiuojame ND, naudojant `sort` iš <algorithm>
sort(nd.begin(), nd.end());
// apskaičiuojame medianą
vecSize vid = size/2;    // vidurinis elementas
double mediana;
mediana = size % 2 == 0 ? (nd[vid-1] + nd[vid]) / 2
                         : nd[vid];
```

- ☞ Salyginis operatorius yra santrumpa **if-then-else** išraiškai ir dažnai vadinamas ? : operatoriumi.

# Potencialiai didelės programos organizavimas

## Skaidymas į mažesnes dalis

- ☞ Kaip ir daugumoje programavimo kalbų, C++ siūlo du pagrindinius didelių programų organizavimo būdus:
  - ☞ panaudojant funkcijas, kartais dar vadinamas paprogramėmis.
  - ☞ panaudojant duomenų struktūras.
- ☞ Be to, C++ leidžia apjungti funkcijas ir duomenų struktūras į **klases**, su kuriomis "draugausime" nuo kito darbo.
- ☞ Galiausiai, C++ leidžia išskaidyti programą į mažesnius atskirai kompiliuojamus failus, kuriuos apjungia po kompiliavimo.

# Versijų (v0.2) ir (v0.3) kūrimas (1)

## **Galutinis balas (1)**

```
/* pagal egzamino ir namų darbu (nd) iverti (vidurki, medianą)
   apskaičiuoja galutinį balą */
double galBalas(double egzaminas, double nd) {
    return 0.6 * egzaminas + 0.4 * nd;
}
```

# Versijų (v0.2) ir (v0.3) kūrimas (2)

## Medianos radimas

```
// Apskaičiuojame medianą: funkcija nukopijuoją visą vektorių
double mediana(vector<double> vec) {
    typedef vector<double>::size_type vecSize;
    vecSize size = vec.size();
    if (size == 0) // std::domain_error deklaruota <stdexcept>
        throw std::domain_error("negalima skaičiuoti medianos tuščiam vektoriui");
    sort(vec.begin(), vec.end()); // surūšiuojame vektorių į variacinę eilutę
    vecSize vid = size / 2; // vidurinis vektoriaus elementas
    return size % 2 == 0 ? (vec[vid] + vec[vid-1]) / 2 : vec[vid];
}
```

- ☞ Kadangi `mediana()` funkciją galime naudoti ne tik ND rezultatams, todėl **universaliau** informuojame (naudojant `throw`) kai vektorius `vec` yra tuščias.

# Išimtys (exceptions)

- ☞ Kai vektorius yra tuščias, paleidžiame (**throw**) išimti (**exception**).
- ☞ Kai programa paleidžia išimti, programos vykdymas baigiasi toje programos vietoje, kur throw buvo įvykdyta, ir **kartu su išimties objektu** pereina į **kitą** programos vietą.
- ☞ Šiuo atveju išimtis yra `std::domain_error` tipo, kuris apibrėžtas `<stdexcept>` šablone, kurios tikslas informuoti, kad funkcijos argumentas yra **už leistinų ribų**.
- ☞ Kai mes sukuriame `std::domain_error` išimti, o `string/const char*` informuojame, kas blogo įvyko. Ši informacija vėliau gali būti panaudota, pvz. diagnostiniame pranešime.

# Versijų (v0.2) ir (v0.3) kūrimas (3)

## Galutinis balas (2)

```
// pagal egzamino ir namų darbų rezultatus suskaičiuoja galutinį balą
// ši funkcija nekopijuoja vektoriaus; tą atlieka mediana()
double galBalas(double egzaminas, const vector<double>& nd) {
    if (nd.size() == 0) // patikriname, ar atliko bent vieną ND
        throw std::domain_error("studentas neatliko nė vieno namų darbo");
    return galBalas(egzaminas, mediana(nd)); // overloading: galBalas(double, double)
}
```

👉 Čia `const vector<double>&` reiškia "reference to vector of const double":

```
vector<double> namuDarbai;
vector<double>& nd = namuDarbai;           // nd yra `namuDarbai` sinonimas
const vector<double>& cnd = namuDarbai; // cnd yra read-only `namuDarbai` sinonimas
```

# Nekonstantinės nuorodos (non-const references)

Žvilgsnis į praeitį 😮

- 👉 Nekonstantinės nuorodos **inicializuojamos** sukūrimo metu ir tik i "non-const l-values"
- 👉 l-values - objektai, turintys dedikuotą atminties adresą (pvz., kintamieji).
- 👉 r-values - laikini neturi dedikuotos atminties objektai.

```
int x = 10;
const int y = 20;
int &ref = x;           // OK, ref ir x - tas pats; x yra non-const l-value
int &wrongRef;         // negalima, nuoroda turi būti inicializuota!
int &ref2 = y;          // negalima!, y yra const l-value
int &ref3 = 10;          // negalima!, 10 yra r-value
```

# Konstantinės nuorodos (const references)

- ☞ Konstantinės nuorodos **inicializuojamos** sukūrimo metu i "non-const l-value", "const l-values", ir "r-values":

```
int x = 10;
const int y = 20;
const int &ref = x;      // OK, x yra non-const l-value
const int &ref2 = y;    // OK, y yra const l-value
const int &ref3 = 10;   // OK, 10 yra r-value
```

# Nuorodos (references) vs. rodyklės (pointers)

- ☞ Nuorodos ir rodyklės yra stipriai susijusios - nuorodos elgiasi, analogiškai dereferenced `const rodyklėms.
- ☞ Iprastai nuorodos kompiliatorių realizuotos naudojant rodykles:

```
int x = 10;  
int &ref = x;  
int * const ptr = &x; // galime keisti tik *ptr reikšmę
```

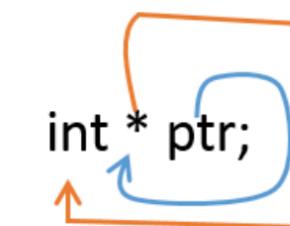
- ☞ čia ref ir \*ptr elgiasi taip pat, t.y. ref == \*ptr.

# Įvairūs int \* const variantai

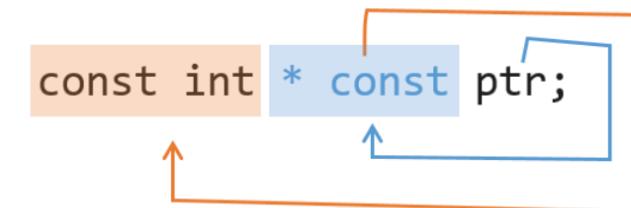
☞ Skaitome "žydiškai" (paremta **Clockwise/Spiral Rule**)<sup>4</sup>

int*	- pointer to int
int const *	- pointer to const int
int * const	- const pointer to int
int const * const	- const pointer to const int

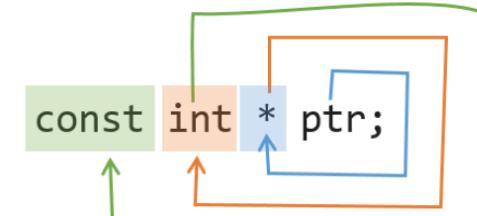
```
// pirmasis const gali būti abejose tipo pusėse
const int *      == int const *
const int * const == int const * const
```



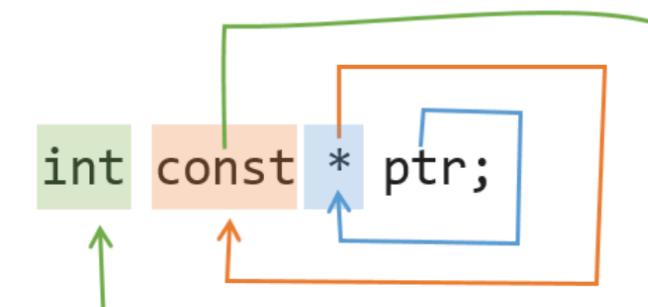
ptr is a pointer to int



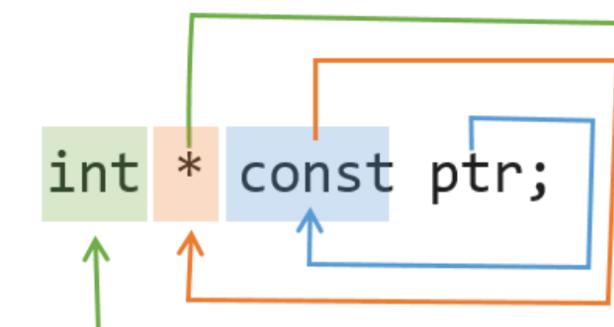
ptr is a constant pointer to const int



ptr is a pointer to int constant (i.e. const int)



ptr is a pointer to const int



ptr is a const pointer to int

<sup>4</sup> <https://stackoverflow.com/questions/1143262/what-is-the-difference-between-const-int-const-int-const-and-int-const>

# Versijų (v0.2) ir (v0.3) kūrimas (4)

## Namų darbų rezultatų nuskaitymas (1)

```
// iš įvedimo stream'o nuskaityti namų darbus į vector<double>
istream& readNd(istream& in, vector<double>& nd) {
    if (in) {          // jei stream būsena OK
        nd.clear();   // sunaikina vektoriaus elementus (jeigu buvo)
        double x;      // skaityti namų darbus
        while (in >> x) nd.push_back(x);
        in.clear();   // išvalo srautą, kad veiktu sekančiam studentui
    }
    return in;
}
```

“`std::vector::clear` - Removes all elements from the container. Invalidates any references, pointers, or iterators referring to contained elements. Any past-the-end iterators are also invalidated. Leaves the `capacity()` of the vector unchanged.<sup>5</sup>”

---

<sup>5</sup> <http://en.cppreference.com/w/cpp/container/vector/clear>

# Namų darbų rezultatų nuskaitymas (2)

☞ Srauto istream& readNd() gražinimas leidžia funkciją naudoti tokiu būdu:

```
if (readNd(cin, nd)) { /*...*/ }
```

// Priešingu atveju reiktu:

```
readNd(cin, nd);
if (cin) { /*...*/ }
```

“std::istream::clear - Sets the stream error state flags by assigning them the value of state. By default, assigns std::ios\_base::goodbit which has the effect of clearing all error state flags.<sup>6</sup>”

---

<sup>6</sup> <http://en.cppreference.com/w/cpp/container/vector/clear>

# Versijų (v0.2) ir (v0.3) kūrimas (5)

☞ Po šių atnaujinimų, mūsų realizacija galėtų būti:

```
/* PRIDÉTI: naudojamus `using`; reikiamus <header> failus; bei funkcijų:
 * readNd(), mediana (vector<double>), galBalas(double, double),
 * galBalas(double, const vector<double>&) realizacijas */
int main() {
    /* PRIDÉTI: realizaciją nuskaitytivardą, pavardę, egzamino rezultatai */
    vector<double> nd;
    readNd(cin, nd); // nuskaityti ND
    try { // apskaičiuoti ir atspausdinti galutinį balą (jei įmanoma)
        double galutinis = galBalas(egzaminas, nd);
        streamsize prec = cout.precision();
        cout << "Galutinis balas yra " << setprecision(3)
            << galutinis << setprecision(prec) << endl;
    } catch (std::domain_error) {
        cout << endl << "Privalote įvesti studento rezultatus. "
            "Bandykite iš naujo." << endl;
        return 1;
    }
    return 0;
}
```

# Klausimai!?

