Programavimo kaita ir aktualijos 2015
IT VBE programavimo uždavinių sprendimas

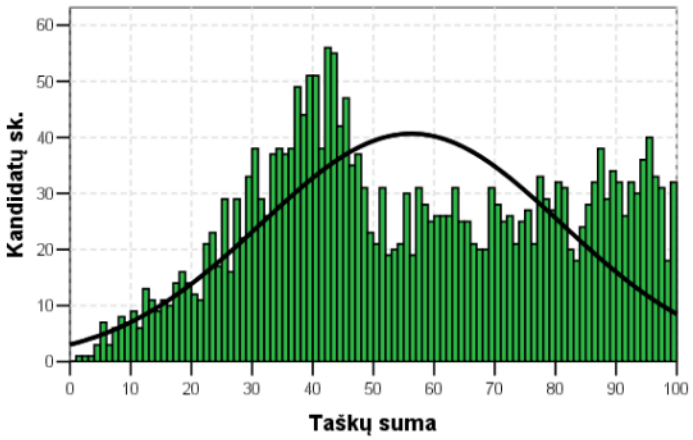
Albertas Dinda, 2015

1. Prielaidos

1.1. IT VBE rezultatai

- Egzaminą leista laikyti 2872 kand.
- Neatvyko 253 kand.
- Išlaikymo riba 20 taškų
- Neišlaikė 5,95% laikiusiųjų
- Pakartotinės sesijos IT VBE laikė 17 kand.
- Surinktų taškų vidurkis 56,2
- Didžiausias įvertinimas 100 balų
- NEC informacija

1.2. IT VBE rezultatai



1 diagrama. Informacinių technologijų valstybinį brandos egzaminą laikiusių kandidatų surinktų taškų pasiskirstymas

1.3. IT VBE rezultatai

- Visų laikiusių kandidatų populiacija skyla į dvi dalis
- Kairioji – greičiausiai B lygiu mokinęsi, nesprendę programavimo užduočių
- Dešinioji – sprendę ir programavimo užduotis

1.4. IT VBE rezultatai

- Sprendę programavimo užduotis buvo pasiruošę labai skirtingai
- Mokant spręsti programavimo uždavinius pagal tam tikrą sistemą, galima gauti geresnius rezultatus
- Programavimo sistemos gana aiškiai išdėstytos mokyklų turimuose programavimo kurso vadovėliuose
- Mokiniai vangiai naudojasi daugelio mokymosi dalykų sisteminiu taikymu

1.5. IT VBE programavimo užduotys

- Sudarytojai kuria užduotis, kurios nėra panašios į jau buvusias

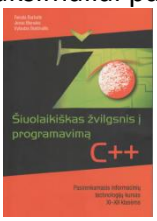
- Stengiamasi neišeiti už IT VBE programavimo užduočių reikalavimų
- Programavimo užduočių pateikimas ilgas ir painus (2013 m. – 4 psl., 2014 m. – 4,2 psl., 2015 m. – 3,5 psl.)
- Vadovėliuose dėl vietos stokos sąlygos, netgi skirtos kurso sisteminimui, pateikiamos glaustai

1.6. Pagrindiniai mokinio gebėjimai

- Greitai perskaityti sąlygą ir suvokti jos esmę ir reikalavimus
- Įsigilinus sugalvoti uždavinio duomenų struktūras ir sprendimo algoritmą
- Algoritme prireikus remtis jau žinomais pagrindiniais algoritmais, mokėti juos keisti ir pritaikyti esamai situacijai
- Mokėti algoritmą skaidyti dalimis ir jas apiforminti funkcijų pavidalu
- Planuoti laiko paskirstymą užduotims atlikti. Suvokti kiekvieno užduoties elemento svarbą ir sąnaudas jį sukurti

1.7. Seminaro uždaviniai

- Parodyti svarbias programavimo kurso ypatybes, į kurias mokiniai mažai kreipia dėmesį
- Parodyti užduoties analizės nuoseklumą ir ir svarbiausius momentus remiantis 2015 m. IT VBE programavimo užduotimis
- Parodyti programavimo užduoties atlikimo nuoseklumą, leidžiantį maksimaliai panaudoti savo galimybes
- Aptarti programavimo mokymo problemas
- Atsakyti į seminaro dalyvių klausimus



2.1. Svarbu

2.1.1. Svarbu

- Algoritmų savybės
- Programos kūrimo etapai
- Pateikto uždavinio sprendimo etapai
- Programos rašymo ypatybės
- Duomenų skaitymas
- Apie true ir false

2.1.2. Algoritmų savybės

- Kiekvienas algoritmas privalo turėti pradinius duomenis ir rezultatus.
- Algoritmas atliekamas įvykdžius baigtinį žingsnių skaičių.
- Algoritmo žingsniai sudaro baigtinę seką.
- Algoritmo žingsnius vienareikšmiškai turi suprasti ir programuotojas, ir vykdytojas.
- Algoritmas turėtų veikti su įvairiais pradiniais duomenimis.

2.1.3. Programos kūrimo etapai

1. Suformuluojama problema ar uždavinys (jei jie nesuformuluoti).

2. Nustatoma, kas yra pradiniai duomenys ir kas – rezultatai.
3. Randamas pradinių duomenų ir rezultatų sąryšis.
4. Sąryšis išreiškiamas algoritmu.
5. Algoritmas užrašomas programavimo kalbos žymenimis.
6. Parengiami testai. Parenkami pradiniai duomenys, su kuriais algoritmo rezultatai yra žinomi.
7. Programa testuojama. Jos veikimas patikrinamas su parinktais duomenimis.
8. Jei testavimas nepavyko, programos veikimo rezultatai ne tokie, kokių tikėtasi, reikia patikrinti, ar nepadaryta klaidų vykdant 1–7 etapus.
9. Programa tobulinama. Gali tekti grįžti prie 1–6 etapų. Taip atsitinka tada, kai realizuotos programos galimybės netenkina poreikių arba kai reikia tikslinti uždavinį.

2.1.4. Sprendžiant pateiktą uždavinį

1. Perskaitome sąlygą (bendram supratimui). Dar kartą labai įdėmiai perskaitome sąlygą, daugiau kreipiame dėmesį į kertinius elementus, išsiryškiname svarbiausias užduoties atlikimo ypatybes.
2. Parenkame reikiamas konstantas, tipus, kintamuosius ir juos aprašome.
3. Nustatome, kas yra pradiniai duomenys ir kas – rezultatai. Rašome skaitymo, skaičiavimo ir rašymo funkcijas.
4. Randame pradinių duomenų ir rezultatų sąryšį.
5. Sąryšį išreiškiame algoritmu. Algoritmą skaidome į smulkesnes dalis – funkcijas pagal savo poreikius ar užduoties reikalavimus.
6. Algoritmą užrašome programavimo kalbos žymenimis.
7. Parengiame testus. Parenkame pradinius duomenis, su kuriais algoritmo rezultatai yra žinomi, modeliuojame svarbias skaičiavimo situacijas.
8. Programą testuojame. Jos veikimas patikrinamas su parinktais duomenimis.
9. Jei testavimas nepavyko, programos veikimo rezultatai ne tokie, kokių tikėtasi, reikia patikrinti, ar nepadaryta klaidų vykdant 1–7 etapus.

2.1.5. Programos rašymas

- Programą rašome taip, kad bet kuriame etape ją galima būtų kompiliuoti
- Programos tekstą iš karto reikiamai lygiuokite
- Programos tekste iš karto rašome komentarus
- Parašius { tuoj parašykime }
- Parašius **ifstream D(...);** parašykime **D.close();**
- Parašius **ofstream R(...);** parašykime **R.close();**
- Neužmirškite direktyvų **#include <...>**
- Parašius ciklo **while** antraštę neužmirškime jo ypaytbių:
  - Pasiruošimas
  - Sąlyga
  - Kintamųjų kitimas
- Sudėtinguose loginiuose reiškiniuose naudokite skliaustus, nes jie išryškina veiksmų atlikimo tvarką
- Be reikalo nesumaišykite sveikųjų ir realiųjų skaičių. Jei tai būtina – naudokite tipo keitimo operatorių.
- Jei naudojate simbolių eilutes pasitikslinkime, ar pakaks simbolių masyvų, ar naudosite **string** klasę.
- Pasitikslinkite, gal skaitant duomenis teks skaityti tekstą su tarpais.
- Su priskyrimo operatoriumi galima struktūrai priskirti to paties tipo struktūrą.

- Masyvui negalima priskirti masyvo!
- Eiluei string galima priskirti eilutę arba simbolių masyvą.

2.1.6. Didžiausios taktinės klaidos

- Mokinys neįsigilina į užduočių sąsiuvinio turinį
- Nepaskirsto brangaus laiko. Gaišta ties smulkmenomis. Praranda laiką svarbiems dalykams
- Nepaiso pateiktų reikalavimų
- Neturėdamas algoritmo (bent galvoje) pradeda chaotiškai rašyti programą
- Nestruktūrizuoja programos (bent formaliai)
- Nerašo komentarų
- Nerašo programos teksto tvarkingai, tikisi sutvarkyti vėliau
- Netvarkingai parašytoje programoje negali rasti klaidų, pasitiki kompiliatoriumi, kuris aptinka ne visas sintaksės klaidas. O kur loginės klaidos?

2.1.7. Apie operatorių >>

- Juo skaitomi
  - Simboliai atskirti skirtukais
  - Visų tipų ir pavidalų skaičiai atskirti skirtukais
  - Tekstas be tarpų atskirtas skirtukais

Skirtukas: tarpo ženklas, tabuliatorius, eilutės pabaigos simbolis

2.1.8. Teksto su tarpais skaitymas

- Duomenys turi būti formatuoti, t.y. turime žinoti, kiek simbolių reikia perskaityti
- Jei tekstas duomenų eilutės pradžioje, tačiau ne pirmoji duomenų eilutė, teks „perlipti“ iš eilutės pabaigos į jos pradžią su funkcija **ignore()**

2.2. Pavyzdys

2.2.1. Duomenys

```
5
Jonas Gugis     182
Petras Kukis    167
Ona Galvonaitė  172
Rima Aukštaitė  183
Jurgis Žemaitis 166
```

2.2.2. Struktūros

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
const int NMax = 10;
const int VIlg = 16;
struct Tzm {
   string v;
   int ug;
};
```

2.2.3. Skaitymo principas

```
#include <iostream>
```

```
#include <fstream>
#include <string>
using namespace std;
const int NMax = 10;
const int VIlg = 16;
struct Tzm {
   string v;
   int ug;
};
 int main() {
    int n;
    Tzm M[NMax];
    char v[VIlg+1];
    ifstream D("duom.txt");
    D >> n;
    for(int i = 0; i < n; i++){
        D.ignore();      // perlipame per eilutės pabaigą
        D.get(v, VIlg);  // skaitome tekstą
        M[i].v = v;
        D >> M[i].ug;    // skaitome skaičių
    }
    D.close();
    //...
    return 0;
}
```

2.2.4. Kodėl taip? Todėl, kad...

- Funkcijos get prototipai iš **<istream>**:
- Simbolių masyvams

istream& get (char* s, streamsize n);
istream& get (char* s, streamsize n, char delim);

- Eilutėms **string**

istream& get (streambuf& sb);
istream& get (streambuf& sb, char delim);

2.3. Apie true ir false

2.3.1. Neapibrėžtumas

- Tipas bool apibrėžtas kaip vieno baito sveikieji

true – 1
false – 0

- Veiksmuose

false – 0

true – bet kokia kita reikšmė, išskyrus 0

2.3.2. Dažna klaida

```
int a = 10;
if ( a = 0){
          // ...
}
else {
          // visada skaičiuos čia
}
```

a == 0

2. Klauskite

3. Dalybos

3.1. Pirmosios užduoties analizė. Dalybos

3.1.1. Pirmosios užduoties analizė. Sąlyga

- **Dvidešimt mokinių išsirikiuoja į eilę taip: kairėje stovi 10 mergaičių, o dešinėje – dešimt berniukų. Kiekviena mergaitė rankose laiko po dubenėlį, kiekviename dubenėlyje yra po dešimt slyvų.**
- **Kai kurios mergaitės suvalgo po kelias slyvas.**
- **Kiekviena mergaitė perduoda dubenėlį dešinėje nuo jos esančiam mokiniui. Kiekvienas mokinys, gavęs dubenėlį, suvalgo vieną slyvą ir perduoda jį toliau į dešinę tol, kol slyvos baigiasi.**
- **Parašykite programą, kuri apskaičiuotų, kiek slyvų suvalgė kiekvienas mokinys.**

3.1.2. Pradiniai duomenys

6 3 2 8 0 5 4 9 1 3

Vienoje eilutėje surašyta dešimt sveikųjų skaičių, atskirtų vienu tarpo simboliu.

Šie skaičiai nusako, kiek slyvų suvalgė kiekviena mergaitė prieš joms pradedant vaišinti kitus mokinius.

3.1.3. Rezultatai

6 4 4 11 4 9 8 14 7 8 6 4 4 3 3 2 2 1 0 0

Dvidešimt sveikųjų skaičių, atskirtų vienu tarpo simboliu, nusakančių, kiek slyvų suvalgė kiekvienas mokinys.

3.2. Esminiai reikalavimai

3.2.1 Esminis reikalavimas

Parašykite funkciją, kuri apskaičiuoja, kiek slyvų suvalgo kiekvienas mokinys

3.2.2. Programos vertinimas

Vertinimo kriterijai	Taškai	Pastabos
Testai.	14	Visi taškai skiriami, jeigu programa pateikia teisingus visų testų rezultatus.
Teisingai skaitomi duomenys iš failo.	3	Vertinama tada, kai neskiriama taškų už testus.
Teisingai išvedami rezultatai į failą.	3	
Teisingai nustatoma, kiek slyvų suvalgė mokiniai.	6	
Teisingos kitos funkcijos[1], jeigu jų yra, ir `main()` funkcija[2].	2	
Sukurta ir naudojama funkcija, apskaičiuojanti, kiek slyvų suvalgė mokiniai.	2	Visada vertinama.
Teisingai aprašyti kintamieji ir kitos duomenų saugojimo struktūros.	2	
Prasmingai pavadinti kintamieji. Komentuojamos programos dalys.	1	
Laikomasi rašybos taisyklių. Išlaikomas vientisas programos rašymo stilius, nėra sakinių, skirtų darbui su ekranu.	1	
Iš viso taškų	**20**	

3.3. Pirmosios užduoties programavimas. Dalybos

3.3.1. Pasiruošimas

```
// U1
#include <iostream>
#include <fstream>
using namespace std;
```

Reikia prisiminti 2 skyrių apie tekstinius failus.

Taip pat įterpiamuosius failus. T. y. kuriame yra sutelkti darbo su tekstiniais failais algoritmai.

3.3.2. Konstantos

```
const char FVD[] = "U1.txt";
const char FVR[] = "U1rez.txt";
const int N = 10;
const int N2 = 20;
const int SlSk = 10;
```

Reikia prisiminti:
Konstantų aprašymo tvarką.
C simbolių masyvus. 5 skyrius.
Programa yra gera, jei joje beveik nėra skaičių. Gali pasitaikyti labai paprasti skaičiai 1, 2. Visus kitus - apiforminkime kaip konstantas.

3.3.3. Funkcijų prototipai

```
void skaityk (const char FVD[], int M[]);
void dalink ( int M[]);
void rasyk (const char FVR[], int M[]);
```

Prisiminti 3 skyrių. Funkcijos.

Funkcijų prototipai.

Funkcijų formalieji parametrai, jų rūšys ir ypatybės.

Konstantos – tai, kas nesikeičia funkcijos veikimo metu turėtų būti konstanta.

Kaip perduodami masyvai parametrais – tik adresas, t.y. nuoroda.

3.3.4. Funkcijos

```
void skaityk (const char FVD[], int M[]){
}
// kiek slyvų suvalgė vaikai
void dalink ( int M[]){
}
void rasyk (const char FVR[], int M[]){
}
```

Prisiminti 3 skyrių. Funkcijos.

Funkcijų prototipai.

Funkcijų formalieji parametrai, jų rūšys ir ypatybės.

Konstantos – tai, kas nesikeičia funkcijos veikimo metu turėtų būti konstanta.

Kaip perduodami masyvai parametrais – tik adresas, t.y. nuoroda.

3.3.5. Programos veiksmai

```
int main(){
    int M[N2] = {0}; // kiek suvalgė
    skaityk (FVD, M);
    dalink(M);
    rasyk(FVR, M);
    return 0;
}
```

Jau galima kompiliuoti

Prisiminti 3 skyrių. Funkcijos.

Reikia mokėti užrašyti kreipinius į funkcijas. Formalieji ir faktiniai parametrai.

Lokalių masyvų iniciavimas nuliu. 4 skyrius. Masyvai.

Masyvas į funkciją perduodamas nuoroda.

3.3.6. Skaitymas (1)

```
void skaityk (const char FVD[], int M[]){
   ifstream D(FVD);
   D.close();
}
```

Prisiminti duomenų skaitymą iš tekstinio failo.

Kaip aprašyti srautą susiejant su tekstiniu failu.

Kaip užverti srautą, tuo pačiu ir duomenų failą.

3.3.6. Skaitymas (2)

```
void skaityk (const char FVD[], int M[]){
   ifstream D(FVD);
   for(int i = 0; i < N; i++)
       D >> M[i];
   D.close();
}
```

Duoto ilgio skaičių sekos skaitymas iš tekstinio failo. 2 skyrius. Tekstiniai failai.

3.3.7. Rašymas (1)

```
void rasyk (const char FVR[], int M[]){
```

```
    ofstream R(FVR);
    R.close();
}
```

Sukurti srautą rašymui ir susieti su tekstiniu failu. 2 skyrius.

Kaip užverti failą, kai baigti rašyti rezultatai.

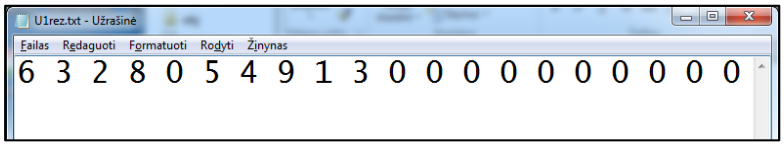
3.3.7. Rašymas (2)

```
void rasyk (const char FVR[], int M[]){
    ofstream R(FVR);
    for(int i = 0; i < N2; i++)
        R << M[i] << " ";
    R << endl;
    R.close();
}
```

Duoto ilgio skaičių sekos rašymas į tekstinį failą, rezultatus atskiriant tarpais. 2 skyrius. Tekstiniai failai.

Eilutės pabaigos simbolis.

3.3.7. Galime pasitikrinti



Kaip pasitikrinti gautus rezultatus?

Atverti failą su užrašine.

Atverti failą su programavimo aplinka (galbūt prijungus jį prie projekto). Darbas su konkrečia programavimo aplinka.

3.3.8. Dalinimas. 1 būdas (1)

```
void dalink ( int M[]){
    int L[N] = {0}; // likučiams po valgymo
    // valgymas
    for (int i = 0; i < N; i++)
        L[i] = SlSk - M[i];
}
```

Lokalus masyvas funkcijoje, jo tik čia ir reikia. Tokio masyvo inicijavimas.

Veiksmai su masyvų elementais. Paprasčiausias ciklinis algoritmas.

4 skyrius. Masyvai.

3.3.8. Dalinimas. 1 būdas (2)

Kiekvienas k-sis dubenėlis keliauja dešinėn kiekvieną kartą paimant po vieną slyvą, jei jų yra

```
for (int k = 0; k < N; k++){
  // pradedame nuo vaiko dešiniau dubenėlio
  int i = k + 1;
  // kol k-tajame dubenėlyje yra slyvų
  while (L[k] > 0){ // kol yra slyvų dubenėlyje
     M[i]++; // i-sis vaikas gauna slyvą
     L[k]--; // k-tajame dubenėlyje mažėja viena slyva
     i++;    // kitas vaikas
  }
}
```

Ciklas cikle 1 skyriaus 7 dalis.

Vienmačiai masyvai 4 skyrius

Masyvų elementų didinimas (mažinimas) vienetu operatoriai ++ ir –.

Ciklų for ir while veikimas.

Lokalūs ciklų kintamieji.

3.3.8. Dalinimas. 1 būdas (3)

```
void dalink ( int M[]){
   int L[N] = {0};
   for (int i = 0; i < N; i++)
      L[i] = SlSk - M[i];
   for (int k = 0; k < N; k++){
      int i = k+1;
      while (L[k] > 0){
         M[i]++; L[k]--;
         i++;
      }
   }
}
```

Kaip turi atrodyti void funkcija? 3 skyrius.

Tačiau geriau šitaip:

```
   for (int k = 0; k < N; k++){
      for (int i = k+1; L[k] > 0; i++){
         M[i]++;    L[k]--;
      }
   }
```

Ciklo for universalumas. Ciklo sąlyga nebūtinai turi būti susijusi su ciklo kintamuoju. Ryšys su ciklu while. Vidinis for ciklas panaudojamas netradiciškai.

3.3.8. Dalinimas. 2 būdas

i-sis vaikas gauna slyvą iš k-tojo dubenėlio, jei M[k] >= i – k;

Idėja

3.3.8. Dalinimas. 2 būdas (2)

```
for (int k = 0; k < N; k++){
  for(int i = k+1; i < N2; i++)
    if(L[k] >= i-k) M[i]++;
}
```

Ciklai for vienas kitame. Tuščia ciklo eiga (ypatybė). Galima būtų nutraukti ciklą su break. Operatorius ++.

Skyriai 1.5 – 1.7. 4 ir 4.

3.3.8. Dalinimas. 2 būdas (3)

```
void dalink ( int M[]){
   int L[N] = {0}; // liks po valgymo
   // valgymas
   for (int i = 0; i < N; i++)
      L[i] = SlSk - M[i];
   // dalinimas
   for (int k = 0; k < N; k++){
      for(int i = k+1; i < N2; i++)
         if(L[k] >= i-k) M[i]++;
   }
}
```

Kaip atrodo void funkcija. 3 skyrius. Funkcijos.

3.3.8. Dalinimas. 3 būdas

```
for(int i = 1; i < N2; i++)
      for (int k = 0; (k < i) && k < N; k++){
         if(L[k] >= i-k) M[i]++;
   }
```

Loginiai operatoriai, palyginimo operatoriai, šakojimas, operatorius ++.

Skyriai 1.4 – 1.5, 1.7.

3.3.9. Būdų palyginimas

- Pirmasis būdas modeliuoja patį dalinimą, pirmiausiai atėjo į galvą. Ciklą **while**

transformuotį į ciklą **for**. Geras

- Antrasis – vertina situaciją ir dalina. Geras
- Trečiasis būdas skaičiuoja kita tvarka, sudėtingesnė ciklo sąlyga, rizikinga
- Antrame būde ciklai **for** – algoritme mažiau klaidų

3.3.10. Visas programos kodas su keturiais būdais

```
#include <iostream>
#include <fstream>

using namespace std;
const char FVD[] = "U1.txt";
const char FVR[] = "U1rez.txt";
const int N = 10;
const int N2 = 20;
const int SlSk = 10; // tiek slyvų buvo dubenėlyje
void skaityk (const char FVD[], int M[]);
int vienam(int i, int L[]);
void dalink (int M[]);  // kiek suvalgė kiekvienas
void rasyk (const char FVR[], int M[]);

int main(){
    int M[N2] = {0};
    skaityk (FVD, M);
    dalink(M);
    rasyk(FVR, M);
    return 0;
}

void skaityk (const char FVD[], int M[]){
   ifstream D(FVD);
   for(int i = 0; i < N; i++)
       D >> M[i];
   D.close();
}

/* 1 būdas
void dalink ( int M[]){
   int L[N] = {0}; // liko po valgymo
   // valgymas
   for (int i = 0; i < N; i++)
      L[i] = SlSk - M[i];

   for (int k = 0; k < N; k++){
      int i = k+1;
      while (L[k] > 0){
```

```
            M[i]++;
            L[k]--;
            i++;
         }
      }
}
*/
/*
// 1 būdas su for ciklu
void dalink ( int M[]){
   int L[N] = {0}; // liko po valgymo
   // valgymas
   for (int i = 0; i < N; i++)
      L[i] = SlSk - M[i];
   // dalinimas
   for (int k = 0; k < N; k++){
      for (int i = k+1; L[k] > 0; i++){
         M[i]++;
         L[k]--;
      }
   }
}

*/
/* 2 būdas su for ciklu
void dalink ( int M[]){
   int L[N] = {0}; // liko po valgymo
   // valgymas
   for (int i = 0; i < N; i++)
      L[i] = SlSk - M[i];

   for (int k = 0; k < N; k++){
      for(int i = k+1; i < N2; i++)
         if(L[k] >= i - k) M[i]++;
   }
}
*/

// 2 būdas su for ciklu ir break
/*
void dalink ( int M[]){
   int L[N] = {0}; // liko po valgymo
   // valgymas
   for (int i = 0; i < N; i++)
      L[i] = SlSk - M[i];

   for (int k = 0; k < N; k++){
      for(int i = k+1; i < N2; i++)
         if(L[k] >= i - k) M[i]++;
```

```
         else break;
   }
}
*/

/* 3 būdas
void dalink ( int M[]){
   int L[N] = {0}; // liko po valgymo
   // valgymas
   for (int i = 0; i < N; i++)
      L[i] = SlSk - M[i];

   for(int i = 1; i < N2; i++)
      for (int k = 0; (k < i) && k < N; k++){
         if(L[k] >= i - k) M[i]++;
   }
}
*/
// 4 būdas su funkcija atskiram vaikui
// dalinam iš likučių esančių kairiau dubenėlių!
int vienam(int i, int L[]){
   int s = 0;
   for(int k = 0; k < N; k++)
       if((L[k] >= i - k) && (i > k)) s++;
   return s;
}

void dalink ( int M[]){
   int L[N] = {0}; // liko po valgymo
   // valgymas
   for (int i = 0; i < N; i++)
      L[i] = SlSk - M[i];
   // dalinimas
   for(int i = 1; i < N2; i++)
         M[i] += vienam(i, L); // juk kai kurie suvalgė iki
dalinimo
}

void rasyk (const char FVR[], int M[]){
   ofstream R(FVR);
   for(int i = 0; i < N2; i++)
      R << M[i] << " ";
   R << endl;
   R.close();
}
```

3. Klauskite

4.1. Antrosios užduoties analizė. Avys

4.1.1. Sąlyga

DNR molekulėje yra užkoduota genetinė informacija, dalijimosi metu perduodama naujos ląstelėms.

Siekiant išsiaiškinti avių giminystės ryšius, yra lyginami jų DNR fragmentai

Parašykite programą, kuri palygintų tiriamą avį su likusiomis avimis:

- Nustatykite DNR fragmentų sutapimo koeficientą – kiek sutampa raidėmis A, T, G ir C pažymėtų DNR nukleotidų, esančių tose pačiose pozicijose
- Surikiuokite likusias avis pagal DNR sutapimo koeficientą mažėjimo tvarka (nuo didžiausio iki mažiausio), o jei koeficientai sutampa, – pagal avies vardą abėcėlės tvarka.

4.1.2. Pradiniai duomenys

```
4 6
3
Baltukas    TAGCTT
Bailioji    ATGCAA
Doli        AGGCTC
Smarkuolis AATGAA
```

Kaip sudaryti kontrolinių duomenų failą?

2 skyrius. Tekstiniai failai.

Pirmoje eilutėje yra avių skaičius n (2 ... 20) ir DNR fragmento ilgis m (4 ... 20);

Antroje eilutėje – tiriamos avies numeris;

Tolesnėse n eilučių yra šie duomenys, atskirti vienu tarpo simboliu:

- Pirmose 10 pozicijų – avies vardas (pirmoji raidė – didžioji)
- DNR fragmentas, užkoduotas raidėmis A, T, G ir C

Visi DNR fragmentai yra skirtingi

4.1.3. Rezultatai

```
Doli
Bailioji    3
Baltukas    3
Smarkuolis 1
```

Kaip atsiranda rezultatų failas?

Kur jį rasti?

Kaip atverti?

2 skyrius. Tekstiniai failai.

4.2. Esminiai reikalavimai

4.2.1 Esminiai reikalavimai

- Programoje naudokite struktūros duomenų tipą vienos avies duomenims (vardui, DNR fragmentui ir DNR sutapimo koeficientui) saugoti
- Programoje naudokite masyvo duomenų tipą avių duomenims saugoti
- Sukurkite funkciją, dviejų avių DNR sutapimo koeficientui apskaičiuoti
- Sukurkite avių rikiavimo pagal DNR sutapimo koeficientą funkciją
- Sukurkite funkciją duomenims skaityti ir rezultatams spausdinti

Kas yra struktūra. 7 skyrius. Struktūros.

Struktūrų masyvai. 8 skyrius.

Simbolių eilutės, pvz. string. 6 skyrius.

Funkcijos. 3 skyrius, 8 skyrius.

Rikiavimas, masyvų rikiavimas. Skyriai 1.6, 4.4, 8.4.

Kokios yra programos struktūrinimo priemonės? Skyrius 3.3.

4.2.2. Programos vertinimas

Vertinimo kriterijai	Taškai	Pastabos
Testai.	20	Visi taškai skiriami, jeigu programa pateikia teisingus visų testų rezultatus.
Teisingai skaitomi duomenys iš failo.	4	Vertinama tada, kai neskiriama taškų už testus.
Teisingai spausdinami rezultatai į failą.	4	
Teisingai apskaičiuojamas dviejų avių DNR sutapimo koeficientas.	3	
Teisingai atliekamas rikiavimas.	5	
Teisingos kitos funkcijos[1], jeigu jų yra, ir `main()` funkcija[2].	4	
Teisingai aprašyti ir naudojami masyvai ir kiti kintamieji.	2	Visada vertinama.
Teisingai aprašyti ir naudojami struktūros duomenų tipai.	2	
Teisingos funkcijų[1] antraštės.	4	
Prasmingai pavadinti kintamieji. Komentuojamos programos dalys.	1	
Laikomasi rašybos taisyklių. Išlaikomas vientisas programos rašymo stilius, nėra sakinių, skirtų darbui su ekranu.	1	
Iš viso taškų	30	

4.3. Antrosios užduoties programavimas

4.3.1. Pasiruošimas

```
// U2
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;
```

Įterpiamieji failai. Kokie algoritmai yra įterpiamuosiuose failuose. Priedai p. 231.

Manipuliatoriai. Priedai, p. 229.

4.3.2. Konstantos

```
const char FVD[] = "U2.txt";
```

```
const char FVR[] = "U2rez.txt";
const int NMaxAv = 20;
const int MMaxDNR = 20;
const int VIlg = 10; // vardo ilgis
```

Įvairių tipų konstantų aprašymas.

Simbilių eilučių konstantos.

Sveikojo tipo konstantos.

Priedai, p. 226, 3.4.

4.3.3. Struktūra

```
struct TAvis{
   string v;   // vardas
   string DNR; // DNR fragmentas
   int k;      // sutapimo koeficientas
};
```

Vardus ir DNR paėmėme **string** tipo. Šio tipo duomenys lengviau palyginami, nereikia naudoti specialios funkcijos

Struktūra. 7 skyrius.

Simbolių eilutės string. 6 skyrius.

4.3.4. Kintamieji

```
TAvis A[NMaxAv]; // avys
int n;           // avių skaičius
int m;           // DNR fr. Ilgis
int k;           // kuri avis pasirinkta
```

Lokalieji kintamieji.

Komentavimas. 1 skyrius.

Sveikojo tipo kintamieji. 1 skyrius.

Struktūrų masyvas. 8 skyrius.

4.3.5. Funkcijų prototipai

```
void skaityk(const char FVD[], int &n, int &m, int &k, TAvis A[]);
void rasyk(const char FVR[], int n, int k, TAvis A[]);
void skaiciuok(int n, int m, int k, TAvis A[]);
int SKoef(TAvis a, TAvis b, int m);
void rikiuok(int n, TAvis A[]);
```

Funkcijų prototipai. 3 skyrius.

4.3.6. Funkcijos

```
void skaityk(const char FVD[], int &n, int &m, int &k, TAvis A[]){
}
```

```
void rasyk(const char FVR[], int n, int k, TAvis A[]){
}

void skaiciuok(int n, int m, int k, TAvis A[]){
}

int SKoef(TAvis a, TAvis b, int m){
    int k = 0;
    return k;
}

void rikiuok(int n, TAvis A[]){
}
```

Funkcijos, kurios grąžina reikšmę per parametrus – void funkcijos.

Ir kurios grąžina per funkcijos vardą. Kaip užrašyti funkciją, kad kompiliuotų? 3 skyrius. Funkcijos.

4.3.7. Programos veiksmai

```
int main(){
    TAvis A[NMaxAv]; // avys
    int n;           // avių skaičius
    int m;           // DNR fr. ilgis
    int k;           // kuri pasirinkta
    skaityk(FVD, n, m, k, A);
    skaiciuok(n, m, k, A);
    rasyk(FVR, n, k, A);
    return 0;
}
```

Jau galima kompiliuoti

Kintamųjų aprašai. 1, 8 skyriai.

Kreipiniai į funkcijas. Formalieji ir faktiniai parametrai. 3 skyrius.

4.3.8. Kontroliniai duomenys

```
4 6
3
Baltukas    TAGCTT
Bailioji    ATGCAA
Doli        AGGCTC
Smarkuolis AATGAA
```

Kaip paruošti kontrolinius duomenis?

Su užrašine.

Su programavimo terpe.

4.3.9. Kontroliniai rezultatai

```
Doli
Bailioji    3
```

```
Baltukas   3
Smarkuolis 1
```

Iš kur atsiranda rezultatų failas?

2 skyrius. Tekstiniai failai.

4.3.10. Skaitymo funkcija (1)

```
void skaityk(const char FVD[], int &n, int &m, int &k, TAvis A[]){
    ifstream D(FVD);
    D.close();
}
```

Funkcijos void parametrai (grąžinamieji). Skyriai 3, 8.

Įvesties srauto kintamojo aprašymas ir susiejimas su tekstiniu failu. 2 skyrius.

Įvesties srauto (failo) užvėrimas. 2 skyrius.

4.3.11. Skaitymo funkcija (2)

```
void skaityk(const char FVD[], int &n, int &m, int &k, TAvis A[]){
    ifstream D(FVD);
    D >> n >> m;
    D >> k;
    k--; // numeriai masyvuose 1 mažesni
    D.close();
}
```

Numeruoti nuo 0 yra patogiau, niekur numerių nereikia spausdinti, jie pagalbiniai

Operatoriai **>> ir – –**.

1 skyrius, 2 skyrius.

4.3.12. Skaitymo funkcija (3)

```
void skaityk(const char FVD[], int &n, int &m, int &k, TAvis A[]){
    ifstream D("U1.txt");
    D >> n >> m;
    D >> k;
    k--; // numeriai masyvuose 1 mažesni
    for (int i = 0; i < n; i++){
       D >> A[i].v >> A[i].DNR;
       A[i].k = 0;
    }
    D.close();
}
```

Tekstą skaitome operatoriumi >>

Masyve esančių struktūros laukų skaitymas iš tekstinio failo. 8 Skyrius.

Operatorius **>>** string duomenų skaitymui, ypatybės. 6 skyrius.

Reikšmių priskyrimas masyve esančios struktūros laukui. 8 skyrius .

4.3.13. Rašymo funkcija (1)

```
void rasyk(const char FVR[], int n, int k, TAvis A[]){
  ofstream R(FVR);
  R.close();
}
```

Rašymas į tekstinį failą. Pasirengimas rašyti – Srauto kintamojo aprašymas ir susiejimas su tekstiniu failu. 2 skyrius.

Srauto (rezultatų failo) užvėrimas. Ten pat.

4.3.14. Esminis taškas

- Visi DNR fragmentai yra skirtingi
- Apskaičiavus gimininguмo koeficientus Doli pati su savimi bus „giminiškiausia“
- Rikiuosime koeficiento mažėjimo tvarka, Doli vis tiek atsidurs masyvo pradžioje.
- Tai supaprastina algoritmą

4.3.15. Rašymo funkcija (2)

```
void rasyk(const char FVR[], int n, int k, TAvis A[]) {
    ofstream R(FVR);
    R << A[0].v << endl;
    R.close();
}
```

Struktūrų masyvo elemento lauko rašymas į tekstinį failą. 8 skyrius

4.3.16. Rašymo funkcija (3)

```
void rasyk(const char FVR[], int n, int k, TAvis A[]){
  ofstream R(FVR);
  R << A[0].v << endl;
  for(int i = 1; i < n; i++)
     R << setw(VIlg) << left
       << A[i].v << " " << A[i].k
       << endl;
    R.close();
}
```

Manipuliatoriai setw ir left. Priedai, p. 229-230.

Struktūrų masyvo elemento lauko rašymas į tekstinį failą cikle. 8 skyrius.

4.3.17. Koeficiento skaičiavimas (1)

```
int SKoef(TAvis a, TAvis b, int m){
    int k = 0;
    return k;
}
```

Sveikojo tipo reikšmių funkcija. Reikšmės grąžinimas. 2 skyrius.

4.3.18. Koeficiento skaičiavimas (2)

```
int SKoef(TAvis a, TAvis b, int m){
    int k = 0;
    for (int i = 0; i < m; i++)
       if(a.DNR[i] == b.DNR[i]) k++;
    return k;
}
```

Žinomo kartojimų skaičiaus ciklas for.

Kiekio skaičiavimo algoritmas. Skyriai 1.6, 4.3, 8.4

Sąlyginis sakinys, operatorius ==. Skyrius 1.4, 8

Operatorius ++. 1 skyrius.

4.3.19. Skaičiavimo funkcija (1)

```
void skaiciuok(int n, int m, int k, TAvis A[]) {
}
```

Tipo void funkcija. 2 skyrius. Reikšmių grąžinimas masyvui. 4, 8 skyriai.

4.3.20. Skaičiavimo funkcija (2)

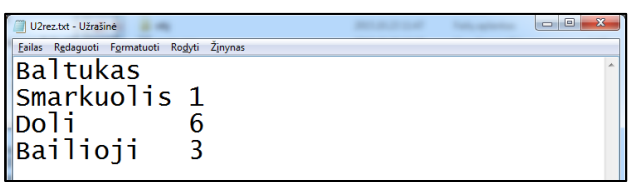
```
void skaiciuok(int n, int m, int k, TAvis A[]){
   // skaičiuojam koeficientus
   for (int i = 0; i < n; i++)
      A[i].k = SKoef(A[k], A[i], m);
}
```

Ciklas for. Skyrius 1.5.

Kreipinys į funkciją. 3 skyrius.

Struktūrų masyvo elemento kauko reikšmės keitimas. Skyriai 7, 8.

4.3.21. Pasitikrinam



4.3.22. Rikiavimo funkcija (1)

```
void rikiuok(int n, TAvis A[]){
}
```

Tipo void funkcija. Rikiavimas. 3 skyrius, 4.4 skyrius, 8.4 skyrius.

4.3.23. Rikiavimo funkcija (2)

```
void rikiuok(int n, TAvis A[]){
   TAvis a;  // sukeitimui
   for(int i = 0; i < n-1; i++)
      for(int j = i+1; j < n; j++)
         if(A[j].k > A[i].k) {
             a = A[i];
             A[i] = A[j];
             A[j] = a;
         }
}
```

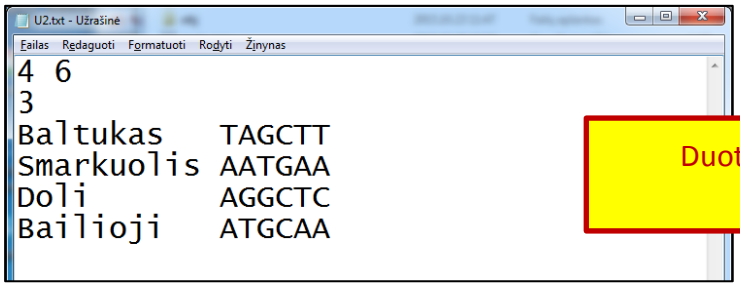
Rikiavimas sukeitimu. Skyrius 4.4.

4.3.24. Koreguojame skaičiavimo funkciją

```
void skaiciuok(int n, int m, int k, TAvis A[]){
  for (int i = 0; i < n; i++)
     A[i].k = SKoef(A[k], A[i], m);
  rikiuok(n, A);
}
```

Kreipinys į funkciją. 3, 8 skyriai.

4.3.25. Keičiame duomenis



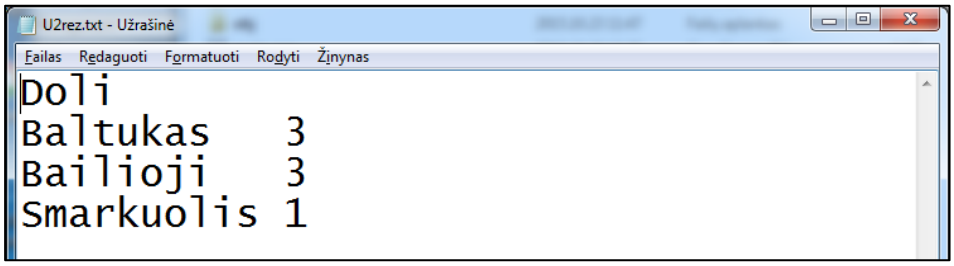
Duotoji duomenų tvarka nėra informatyvi .
Sukeičiame 1 ir 4 avių duomenis

Kokia duomenų tvarkos įtaka rezultatui?

Kokia nauda sukeisti atitinkamas duomenų eilutes?

Ar visada toks sukeitimas naudingas?

4.3.26. Tikriname



4.3.27. Koreguojame rikiavimo funkciją

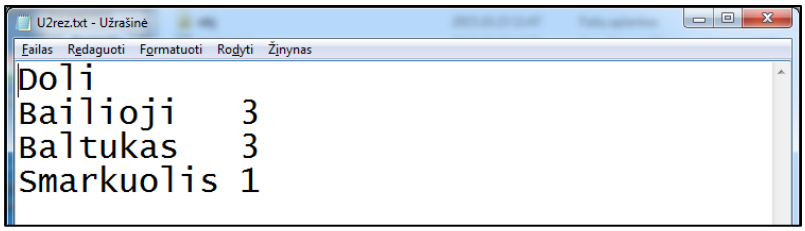
```
void rikiuok(int n, TAvis A[]){
  TAvis a;
  for(int i = 0; i < n-1; i++)
    for(int j = i+1; j < n; j++)
      if((A[j].k > A[i].k)||
         ((A[j].k == A[i].k)&&
         (A[j].v < A[i].v))) {
           a = A[i]; A[i] = A[j]; A[j] = a;
      }
}
```

Struktūrų masyvo rikiavimas pagal du raktus. Skyriai 4.4, 8.4.2.

Simbolių eilučių string palyginimas operatoriais <, >, ==, !=, <=, >=.

Priskyrimo operatorius = struktūrų masyvo elementams. 6 Skyrius.

4.3.28. Tikriname



Kaip ir su kokia programa atverti tekstinį rezultatų failą?

4.3.29. Visas programos kodas

```
// U2
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

const char FVD[] = "U2.txt";
const char FVR[] = "U2rez.txt";
const int NMaxAv = 20;
const int MMaxDNR = 20;
const int VIlg = 10; // vardo ilgis
```

```
struct TAvis{
   string v;   // vardas
   string DNR; // DNR fragmentas
   int k;      // sutapimo koeficientas
};

void skaityk(const char FVD[], int &n, int &m, int &k, TAvis A[]);
void rasyk(const char FVR[], int n, int k, TAvis A[]);
void skaiciuok(int n, int m, int k, TAvis A[]);
int SKoef(TAvis a, TAvis b, int m);
void rikiuok(int n, TAvis A[]);

int main(){
    TAvis A[NMaxAv]; // avys
    int n;           // avių skaičius
    int m;           // DNR fr. ilgis
    int k;           // kuri pasirinkta
    skaityk(FVD, n, m, k, A);
    skaiciuok(n, m, k, A);
    rasyk(FVR, n, k, A);
    return 0;
}

void skaityk(const char FVD[], int &n, int &m, int &k, TAvis A[]){
    ifstream D(FVD);
    D >> n >> m;
    D >> k;
    k--;
    for (int i = 0; i < n; i++){
       D >> A[i].v >> A[i].DNR;
       A[i].k = 0;
    }
    D.close();
}

void rasyk(const char FVR[], int n, int k, TAvis A[]){
    ofstream R(FVR);
    R << A[0].v << endl;
    for(int i = 1; i < n; i++)
       R << setw(VIlg) << left << A[i].v << " " << A[i].k << endl;
    R.close();
}

void skaiciuok(int n, int m, int k, TAvis A[]){
    for (int i = 0; i < n; i++)
        A[i].k = SKoef(A[k], A[i], m);
    rikiuok(n, A);
}

int SKoef(TAvis a, TAvis b, int m){
    int k = 0;
    for (int i = 0; i < m; i++)
```

```
        if(a.DNR[i] == b.DNR[i]) k++;
    return k;
}

void rikiuok(int n, TAvis A[]){
   TAvis a;
   for(int i = 0; i < n-1; i++)
      for(int j = i+1; j < n; j++)
         if((A[j].k > A[i].k) ||
             ((A[j].k == A[i].k) && (A[j].v < A[i].v))) {
             a = A[i]; A[i] = A[j]; A[j] = a;
         }
}
```

4. Klauskite

5. Šaltiniai

- NEC informacija
- VBE rezultatai 2015 metais (NEC)
- Jonas Blonskis, Vytautas Bukšnaitis, Renata Burbaitė. Šiuolaikiškas žvilgsnis į programavimo pagrindus C++. Informacinių technologijų pasirenkamasis kursas IX-X klasėms, TEV, 2011.
- Renata Burbaitė, Jonas Blonskis, Vytautas Bukšnaitis. Šiuolaikiškas žvilgsnis į programavimą C++. Informacinių technologijų pasirenkamasis kursas XI-XII klasėms, TEV, 2011.
- Albertas Dinda. Informacinės technologijos. Pasirenkamasis modulis. Programavimas C++ kalba. Vadovėlis XI-XII klasėms, Šviesa, 2012

6. Kontaktai

albertas_dinda@hotmail.com

http://1drv.ms/1RAfFl8