KAUNO TECHNOLOGIJOS UNIVERSITETAS

Praktinės informatikos katedra

J.Blonskis, V.Bukšnaitis
J.Končienė, D.Rubliauskas

C++ PRAKTIKUMAS

Kaunas 2001

Turinys

Įvadas

C kalba ir jos modifikacija C++ pasižymi labai dideliu lakoniškumu, sintaksinių struktūrų lankstumu ir universalumu, todėl šią kalbą dažniausiai pradedama mokytis jau turint programavimo kitomis kalbomis patyrimą.

Ši knyga skirta skaitytojui, kuriam žinomos pagrindinės programavimo kalbose vartojamos sąvokos, programų struktūrizavimo priemonės, kuris sugeba naudoti standartines duomenų struktūras, valdančias struktūras, programavimo aplinkas. Knygelėje glaustai pateikiamos C++ pagrindinės preimonės (sakinių konstrukcijos, duomenų tipai, funkcijos). Gausiai pavyzdžiais iliustruojamos pagrindinės C++ priemonės, kurios, autorių nuomone, būtinos pradinei pažinčiai su šia kalba. Pagrindinis dėmesys skirtas klasėms ir jų savybėms.

Programavime plačiai vartojamas dinaminis atminties skirstymas. Tai leidžia kurti efektyvesnes programas. Dinaminės duomenų struktūros leidžia kurti programuotojui norimos konfigūracijos ir sudėtingumo duomenų tipus. Tai labai lanksti ir efektyvi priemonė, naudojama duomenims saugoti bei apdoroti.

Šiame leidinyje pristatomos pagrindinės klasikinės dinaminės duomenų struktūros: tiesiniai vienkrypčiai sąrašai ir jų modifikacijos (stekas, dekas, eilė, žiedas), dvikrypčiai tiesiniai sąrašai, tiesinių sąrašų rinkiniai, medžio tipo sąrašas. Nagrinėjamos pagrindinės operacijos su sąrašais: formavimas, peržiūra, paieška, rikiavimas, šalinimas, įterpimas, naikinimas.

Visi veiksmai iliustruojami realiomis programomis, parašytomis ir patikrintomis su C++4.5**.** Knyga skirta studentams, studijuojantiems duomenų struktūras, tačiau stengtasi pristatyti dinamines duomenų struktūras detaliai su kuo daugiau pavyzdžių, kad būtų suprantama ir žingeidžiam moksleiviui ar programavimo mėgėjui.

Ši knyga skirta praktinei pažinčiai su C++, todėl išsamesnėms kalbos studijoms reikalinga specialiai tam skirta literatūra. C++ kalbai skirtų knygų daug, tačiau jos siūlomos užsienio kalbomis. Lietuviškai parašytų knygų labai mažai. Autoriai tikisi, kad ši knyga palengvins pradedančiam programuotojui pasirinkti jam tinkamą literatūrą.

1 skyrius. C++ elementai

1.1. Programos struktūra

Pradedant programuoti su C++ kalba, siūloma atkreipti dėmesį į tokias šios kalbos ypatybes:

- yra tik viena programos struktūrizavimo priemonė – funkcija;
- programos sąvoką atitinka pagrindinė funkcija, kuri žymima vardu `main`;
- identifikatoriuose didžiosios ir mažosios raidės nesutapatinamos;
- programos objektų (struktūrų, kintamųjų, funkcijų) aprašai gali būti bet kurioje programos vietoje – svarbu tik tai, kad objektas būtų apibrėžtas prieš jį naudojant;
- aprašuose plačiai vartojami funkcijų prototipai;
- C++ kalboje nedidelis standartizuotų operatorių skaičius, todėl naudojamos įvairios bibliotekų sistemos;
- kreipiantis į kintamuosius ir struktūras, plačiai vartojamos rodyklės;
- daugelį duomenų tipų galima interpretuoti keliais įvairiais būdais.

Tipinė programos C++ kalba (programos failo) struktūra:

```
/* Instrukcijos pirminiam procesoriui          */
/* Globaliniai aprašai                         */
/* Funkcijų prototipai                         */
main()
  {
     /*  Pagrindinės funkcijos tekstas         */
  }
/*  Funkcijų aprašai                           */
```

Aprašant programos struktūrą, panaudotos tokios C++ kalbos sintaksinės konstrukcijos:

- komentarai – paaiškinimams skirtas simbolių rinkinys, kurio ribos žymimos simbolių poromis `/*` ir `*/`; parašius eilutėje du simbolius be

tarpo `//` toliau esantis tekstas iki eilutės pabaigos suprantamas kaip komentarai;

- `main()` – pagrindinės programos funkcijos antraštė;
- sudėtinis sakinys – tarp skliaustų `{}` įrašytas sakinių rinkinys, su kuriais programoje elgiamasi taip kaip su vienu sakiniu;

Programos struktūros aprašyme komentarais parodyta, kokia tvarka joje turi būti surašyti struktūriniai elementai:

- Programos pradžioje surašomos instrukcijos pirminiam procesoriui (preprocessor), kuriose nurodoma, kokius programos teksto pertvarkymus reikia atlikti prieš jos kompiliavimą.
- Globaliniais aprašais apibrėžiami tie programos objektai, kurie gali būti vartojami visame rengiamos programos faile (tiek pagrindinėje, tiek vartotojo funkcijose).
- Vartotojo funkcijos gali būti aprašomos ne tik programos gale, bet ir globalinių aprašų dalyje.

Surašius pagalbines funkcijas programos gale, jos tekstą lengviau skaityti ir analizuoti; tada nesilaikoma reikalavimo, kad funkcijas reikia pirma aprašyti, o tik po to vartoti. Šio prieštaravimo išvengiama globalinių aprašų dalyje pateikiant programos gale surašytų funkcijų prototipus. Funkcijų prototipais vadinami jų antraščių sakiniai.

Programai tikslinga suteikti vardą, užrašomą teksto pirmoje eilutėje kaip komentarą. Čia naudinga parašyti daugiau paaiškinimų apie programą. Visa tai leis lengviau ir greičiau atpažinti reikalingą programą.

1.1 pratimas. Programa klausia vartotojo, ar giedras dangus. Kaip atsakymą vartotojas paspaudžia klavišą 1, jeigu giedras dangus, ir 0 priešingu atveju. Programa atspausdina ekrane įvestą reikšmę ir ją perduoda funkcijai, kuri analizuoja atsakymą ir išveda ekrane pilną pranešimą.

```
#include <iostream.h>                  // Įvedimo/išvedimo srautai
#include <conio.h>                     // Tekstinio ekrano tvarkyklė

void Atsakymas(int);                   // Funkcijos prototipas
//                    Pagrindinė programa
void main() {                          // Pagrindinė funkcija
  char simbolis;                       // Kintamojo aprašas
  int Dangus;                          // Kintamojo aprašas
  cout << "Ar saule sviecia?\n";       // Išvedimas ekrane
```

```
  cout << " Taip - 1  Ne - 0 \n";
  cin >> Dangus;                    // Įvedimas klaviatūra
  cout << "Atsakymas:" << Dangus << "\n";
  cout << "Dabar ";
  Atsakymas(Dangus);                // Kreipinys į funkciją
  cout << endl << "Paspauskite bet kokį simbolini klavišą"
       << " ir ENTER" << endl;      // endl – eilutės pabaiga
  cin >> simbolis;
}
//                          Funkcija
void Atsakymas(int Ats) {
  if (Ats == 1) cout << "giedras dangus!!!\n";
  else cout << "danguje labai daug debesu\n";
}
```

Programos pradžioje surašytos instrukcijos pirminiam procesoriui, kurios žymimos simboliu #. Įterpimo instrukcijomis `include` nurodoma, kokių failų tekstai turi būti įterpti instrukcijų pažymėtose vietose pirminio apdorojimo metu. Įterpiamų failų vardai rašomi tarp simbolių < > arba tarp kabučių (""), jeigu failas yra sukurtas vartotojo. Jeigu programoje vartojamos asmeninės pagalbinės bibliotekos, instrukcijomis `include` turi būti nurodomi šiose bibliotekose esančių priemonių prototipų failai, kurių vardai turi plėtinius `.h` (header).

1.2. Kintamųjų tipai ir aprašymas. Pradinių reikšmių priskyrimas

Kintamųjų aprašų sintaksė:

⟨tipas⟩ ⟨kintamųjų sąrašas⟩ **[=** ⟨pradinė reikšmė⟩ **]**

Kaip ir kitose kalbose, duomenų tipai nurodomi baziniais žodžiais. C^{++} kalboje yra vartojami šie baziniai elementarūs duomenų tipai:

1.1 lentelė. Pagrindiniai tipai

Tipas	Galimos reikšmės
char	–128..127
int	–32768..32768
long	–2147483648..2147483647
short	–32768..32768
unsigned	0..65535
float	$-3.4\times10^{-38}..3.4\times10^{38}$
double	$1.7\times10^{-308}..1.7\times10^{308}$

Tokio aprašo pavyzdys programoje `Pratimas1` yra pagrindinėje funkcijoje:

```
int Dangus;
```

Programoje visi kintamieji kiekvienu momentu privalo būti apibrėžti, t.y. turėti konkrečią reikšmę. Kalbos kompiliatorius kintamiesiems skiria atmintį, tačiau nepasirūpina, kas ten yra. Programuotojas privalo tai numatyti. Rekomenduojama programos kintamųjų aprašų dalyje nurodyti pradines jų reikšmes. Programos pavyzdyje galima buvo parašyti:

```
int Dangus = 0;
```

arba atskiru priskyrimo sakiniu:

```
Dangus = 0;
```

Aprašuose vienodo tipo kintamuosius galima grupuoti, pavyzdžiui:

```
int Dangus = 0,  Medis = 0,  Katinas = 0;
```

Tai galima aprašyti ir taip:

```
int  Dangus, Medis, Katinas;
Dangus = Medis = Katinas = 0;
```

Nors abu būdai duoda tą patį rezultatą, tačiau pirmasis tinkamesnis. Jis vaizdesnis ir suprantamesnis, nes vienoje vietoje viskas pasakyta apie kintamąjį. Jį naudojant bus mažiau klystama, nes nereikės dukart kartoti kintamojo vardo. Jis leidžia lengviau suprasti ir modifikuoti programą, nes kintamųjų reikšmės pasakomos jų aprašo vietoje ir kiekviena inicializuojama atskirai, taigi, ir pakeitimus darant, pradines reikšmes galima skirti skirtingas.

Sveikojo tipo kintamųjų aprašyme galima taikyti nuorodą `unsigned`. Taip sukuriami kintamieji, galintys turėti tik teigiamas sveiko tipo reikšmes, pavyzdžiui:

```
unsigned int X;     // reikšmių intervalas: 0..65535
```

Programose labai naudingos konstantos. Jos aprašomos panaudojant kalbos rezervuotą žodį `const`, pavyzdžiui:

```
const  int     A = 125;

const  float   B = 13.5;
```

1.2 pratimas. Programa parodo, kad C^{++} kalboje leidžiama "maišyti" skirtingų tipų kintamuosius. Maišant tipus `char` ir `int`, naudojamas simbolio ASCII lentelės skaitmenis atitikmuo, o maišant tipus `int` ir `float`, paprasčiausiai pridedama arba atmetama trupmeninė skaičiaus dalis.

```
void main() {
  int a, b, c;
  char x, y;
  float num, toy;

  a = b = c = -27;
  x = y = 'A';
  num = toy = 3.6792;

  a = y;                  // a bus lygus 65 (simbolis A)
  x = b;                  // x bus lygus –27 (simbolis Õ)
  num = b;                // num bus lygus –27.00
  c = toy;                // c bus lygus 3
}
```

Kintamasis galioja nuo jo paskelbimo vietos. Kintamuosius galima aprašyti bet kurioje vietoje. Tikslinga prisiminti, kad:

- programos teksto pradžioje prieš `main` funkciją surašyti kintamieji yra globalūs ir galioja visame tolesniame tekste;
- kintamieji, kurių aprašas yra funkcijoje, vadinami lokaliais ir galioja tik joje;
- esant vienodam globalaus ir lokalaus kintamojo pavadinimams, pirmenybė suteikiama lokaliam, t.y. toje funkcijoje globalus negalioja;
- kintamieji gali būti aprašomi jų panaudojimo vietoje, tačiau tai **nerekomenduojama**.

1.3 pratimas. Ši programa iliustruoja, kaip galima aprašyti kintamuosius, ir kuriose programos vietose jie galios.

```
#include <stdio.h>

void head1(void);
void head2(void);
void head3(void);

int count;                // Globalus kintamasis
```

```
//                      Pagrindinė programa
main() {
int index;     // index galioja tik funkcijoje main

head1();
head2();
head3();

for (index = 8; index > 0; index--) {
  int stuff;            // stuff galioja tik šių skliaustelių ribose
  for (stuff = 0; stuff < 7; stuff++)
    printf("%d ", stuff);
  printf (" indeksas = %d\n", index); }
}

int counter;            // counter galioja tik nuo šios vietos
//                     Pirmoji antraštė
void head1(void) {
  int index;            // index galioja tik funkcijoje head1
  index = 23;
  printf("Pirmoji antraste: %d\n", index);
}
//                     Antroji antraštė
void head2(void) {
  int count;            // count galioja tik funkcijoje head2
                        // ir panaikina globalaus kintamojo galiojimą
  count = 53;
  printf("Antroji antraste: %d\n", count);
  counter = 77;
}
//                     Trečioji antraštė
void head3(void) {
  printf("Trecioji antraste: %d\n", counter);
}
```

Įvykdžius programą ekrane matysime tokius rezultatus:

```
Pirmoji antraste: 23
Antroji antraste: 53
Trecioji antraste: 77
0 1 2 3 4 5 6  indeksas = 8
0 1 2 3 4 5 6  indeksas = 7
0 1 2 3 4 5 6  indeksas = 6
0 1 2 3 4 5 6  indeksas = 5
```

```
0 1 2 3 4 5 6  indeksas = 4
0 1 2 3 4 5 6  indeksas = 3
0 1 2 3 4 5 6  indeksas = 2
0 1 2 3 4 5 6  indeksas = 1
```

Priskyrimo sakinyje panaudojome operatorių = (lygybės ženklas). Kiti dažniausiai vartojami operatoriai parodyti 1.2 lentelėje. Operacijoms žymėti naudojami ženklai surašyti 1.3 lentelėje.

1.2 lentelė. Dažniausiai naudojami operatoriai

Operatorius		**Pavyzdžiai ir paaiškinimai**
++	k++ ++k	k reikšmė panaudojama, po to didinama vienetu. k reikšmė didinama vienetu, po to panaudojama.
––	k–– ––k	k reikšmė panaudojama, po to mažinama vienetu. k reikšmė mažinama vienetu, po to panaudojama.
+=	s += k;	s = s + k;
–=	s –= k;	s = s – k;

1.3 lentelė. Operacijų ženklai

Aritmetinės operacijos			
+	sudėtis	*	daugyba
–	atimtis	/	dalyba
%	dalybos modulis		
Sulyginimo operacijos			
==	ar lygios dvi reikšmės?	<	ar pirma mažesnė už antrą?
!=	ar nelygios dvi reikšmės?	>=	ar pirma nemažesnė antrą?
>	ar pirma didesnė už antrą?	<=	ar pirma nedidesnė už antrą?
Loginės operacijos			
&&	loginė daugyba ir (and)	||	loginė sudėtis arba (or)
!	loginis neigimas ne (not)		

Rašant aritmetines išraiškas naudojami paprasti skliaustai. Standartinės matematinės funkcijos yra surašytos bibliotekoje `math.h`.

1.4 pratimas. Programoje naudojamos įvairios palyginimo operacijos.

```
void main() {
  int x, y, z;
  char a, b, c;
  float r, s, t;
```

```
  x = y = z = 11;
  a = b = c = 40;
  r = s = t = 12.987;

  if (x == y) z = -13;   printf("1. z = %d\n", z);
  if (x > y) a = 'A';    printf("2. a = %d\n", a);
  if (!(x > y)) a = 'B'; printf("3. a = %d\n", a);
  if (b <= c) r = 0.0;   printf("4. r = %f\n", r);
  if (r != s) t = c/2;   printf("5. t = %f\n", t);

  if (x = (r != s)) z = 1000;
  printf("6. x = %d z = %d\n", x, z);
  if (x = y) z = 222;
  printf("7. x = %d z = %d\n", x, z);
  if (x != 0) z = 333;
  printf("8. z = %d\n", z);
  if (x) z = 444;
  printf("9. z = %d\n", z);

  x = y = z = 77;
  if ((x == y) && (x == 77)) z = 33;
  printf("10. z = %d\n", z);
  if ((x > y) || (z > 12)) z = 22;
  printf("11. z = %d\n", z);
  if (x && y && z) z = 11;
  printf("12. z = %d\n", z);
  if ((x = 1) && (y = 2) && (z = 3)) r = 12.00;
  printf("13. x = %d y = %d z = %d r = %f\n", x, y, z, r);
  if ((x == 2) && (y = 3) && (z = 4)) r = 14.56;
  printf("14. x = %d y = %d z = %d r = %f\n", x, y, z, r);

  if (x == x) z = 1.234;         // visada vykdoma sąlyga
  printf("15. z = %d\n", z);
  if (x != x) z = 5.678;         // niekada nevykdoma sąlyga
  printf("16. z = %d\n", z);
}
```

Programos rezultatai:

```
1. z = -13
2. a = 40
3. a = 66
4. r = 0.000000
5. t = 20.000000
6. x = 1 z = 1000
```

```
7. x = 11 z = 222
8. z = 333
9. z = 444
10. z = 33
11. z = 22
12. z = 11
13. x = 1 y = 2 z = 3 r = 12.000000
14. x = 1 y = 2 z = 3 r = 12.000000
15. z = 1
16. z = 1
```

1.3. Funkcijos

Programos `Pratimas1` pabaigoje parašyta funkcija `Atsakymas`**,** kurios prototipas užrašytas prieš pagrindinę funkciją `main`. Funkcijos prototipo užrašas baigiamas simboliu **`;`** (kabliataškis). Funkcijos aprašo sintaksė:

[⟨reikšmės tipas⟩**]** ⟨vardas⟩
([⟨formalių parametrų sąrašas⟩]**)**
{
⟨funkcijos tekstas⟩
}

Jeigu funkcijos vardui yra suteikiama reikšmė, jos tekste privalo būti sakinys

return ⟨reikšmė⟩**;**

Jeigu tokio sakinio nėra, funkcijai reikšmė nesuteikiama. Funkcijos reikšmės tipas yra nurodomas žodžiu `void` (tuščias), jeigu funkcija negrąžina reikšmės. Reikšmių neturinčios C^{++} kalbos funkcijos atitinka kitose kalbose (pavyzdžiui Paskalio) procedūras. Kreipiniai į tokias funkcijas programose sudaro atskirus sakinius.

Jeigu funkcija neturi argumentų, argumentų sąrašas gali būti paliekamas tuščias arba žymimas žodžiu `void`. Pavyzdžiui,

```
void Atsakymas(void);
```

Funkcijos prototipe nėra tikslo įvardinti parametrus: kompiliatoriui pakanka žinoti, kiek yra parametrų ir kokie jų tipai, todėl apraše vardai gali būti praleidžiami.

Pagrindinė funkcija `main()`, kuri programoje gali būti tik viena, nurodo kompiliatoriui, kur prasideda programoje vykdomų veiksmų aprašymai.

Pagrindinės funkcijos ir pagalbinių funkcijų tekstai yra sudaromi pagal tas pačias taisykles. Tekstą sudaro funkcijoje vartojamų objektų aprašai, programos vykdomų skaičiavimų ir valdymo sakiniai, komentarai.

Funkcija skaičiavimų rezultatus gali perduoti jos vartotojui dviem būdais:

- per parametrų sąrašą;
- per funkcijos vardą.

Norint gauti funkcijos skaičiavimų rezultatus per parametrų sąrašą, reikia perduoti kreipinio metu adresą vietos, kur turi būti patalpinti grąžinami duomenys. Adresus gali saugoti rodyklės tipo parametrai (žr. 26 psl.). Funkcijos prototipe nurodomi rodyklės tipo parametrai, pavyzdžiui:

```
void Keisti(int *, int);
```

Čia pirmasis parametras rodo, kad kreipinio metu reikia perduoti funkcijai `int` tipo kintamojo adresą, kai tuo tarpu antrasis parametras reikalauja reikšmės, kuri gali būti nurodoma kreipinio metu kaip konstanta, kintamasis ar rodyklė į reikšmę. Pavyzdžiui:

```
Keisti(&a, 5);   Keisti(&a, x);   Keisti(&a,*p);
```

1.5 pratimas. Funkcija **`Keisti`** demonstruoja abi parametrų formas. Pagrindinė funkcija parodo ekrane gaunamus rezultatus.

```
#include <iostream.h>
#include <conio.h>

void  Keisti(int *, int);
//                  Pagrindinė programa
void main() {
  int  a = 20, b = 10, *c = &a;
  cout << "                      a  b  *c \n";
  cout << "Pradines reiksmes  : "
       << a << " " << b << " " << *c << "\n";
  Keisti(&a, b);
  cout << "Po pirmo keitimo   : "
       << a << " " << b << " " << *c << "\n";
  Keisti(&b, 20);
  cout << "Po antro keitimo   : "
       << a << " " << b << " " << *c << "\n";
  Keisti(c, b);
  cout << "Po trecio keitimo  : "
       << a << " " << b << " " << *c << "\n";
  Keisti(&b, *c);
```

```
  cout << "Po ketvirto keitimo: "
       << a << " " << b << " " << *c << "\n";
}
//                    Funkcija Keisti
void Keisti(int *p1,  int p2)
{  *p1 = *p1 + 10;   p2 = p2 + 10;  }
```

Įvykdžius programą ekrane matomi rezultatai:

```
                          a  b  *c
Pradines reiksmes  : 20 10 20
Po pirmo keitimo   : 30 10 30
Po antro keitimo   : 30 20 30
Po trecio keitimo  : 40 20 40
Po ketvirto keitimo: 40 30 40
```

1.4. Duomenų įvedimas/išvedimas

Aprašant programos ir vartotojo dialogą, duomenų įvedimui bei išvedimui panaudojama išorinių srautų bibliotekos `iostream.h` nukreipimo operatoriai << ir >>. Pranešimų išvedimui į ekraną yra vartojama struktūra:

`cout <<` ⟨`simbolių eilutė`⟩

Vardas `cout` žymi standartinį išvedimo srautą (ekraną). C^{++} kalbos simbolių eilutės yra tarp dvigubų kabučių (`"`) įrašyti kompiuterio alfabeto ir valdančių simbolių rinkiniai. Valdantys simboliai gali būti įterpiami bet kurioje eilutės vietoje (1.4 lentelė).

Jų sintaksė: **`\`**⟨`simbolis`⟩

1.4 lentelė. Valdantys simboliai

Pavadinimas	C kalboje	ASCII kodas
Skambutis	`\a`	7
Eilutės pabaiga	`\r`	13
Į kitą eilutę	`\f`	12
Į kitos eilutės pradžią	`\n`	10
Horizontali tabuliacija	`\t`	9
Vertikali tabuliacija	`\v`	11
Nulinis simbolis	`\0`	0

Dažniausiai vartojamas valdantis simbolis `\n`, kuris perkelia ekrano žymeklį į naujos eilutės pradžią. Žymeklio perkėlimui į naujos eilutės

pradžią rekomenduojama naudoti operatorių `endl`. Jeigu norime, kad simbolių eilutėje būtų įterptas simbolis **"** arba \, vartojami atitinkamai \" arba \\.

Rezultatų formatavimui bibliotekoje `iomanip.h` saugomi manipuliatoriai, kurie gali būti įterpiami į išvedimo srautus:

`setw (``n`**`)`** `n` **–** lauko plotis simboliais,
`setprecision (``n`**`)`** `n` **–** skaitmenų skaičius.

Jei `setw` nurodyto lauko dydžio skaičiui nepakanka, manipuliatorius ignoruojamas. Nuoroda `setw` galioja artimiausiai išvedamai reikšmei, o `setprecission` – iki naujo nurodymo.

1.6 pratimas. Programa demonstruoja, kaip veikia manipuliatoriai **`setw`** ir **`setprecission`**.

```
#include <iostream.h>
#include <iomanip.h>

main() {
  float A = 18.236;
  cout << "1. A=" << setw(9) << A << endl;
  cout << "2. A=" << setprecision(3) << A << endl;
  cout << "3. A="
       << setw(10) << setprecision(5) << A <<endl;
  A = 123.45678;
  cout << "4. A=" << A << endl;
}
```

Programos darbo rezultatas bus toks:

```
1. A=   18.236
2. A=18.2
3. A=    18.236
4. A=123.46
```

Skaitymas iš standartinio įvedimo srauto (klaviatūros):

`cin >>` `Kintamasis`

Klaviatūroje renkami duomenys sudaro ASCII kodo simbolių srautą, kurio interpretavimo būdą nurodo kintamojo, kuriam nukreipiami duomenys, tipas.

Kelių kintamųjų reikšmes galima įvesti taip:

```
cin >> a >> b >> c;
```

Be šių srautų klasių, C++ kalboje yra vartojamos ir klasikinės įvedimo/išvedimo bibliotekos. Tokios bibliotekos yra trys: `stdio.h` (klasikinis buferizuotas I/O `input/output`), io.h (žemo lygio nebuferizuotas UNIX standarto I/O), `conio.h` (specialios I/O funkcijos). Pratimuose analizuojamos tik bibliotekos `stdio.h` funkcijos.

Aprašant I/O operacijas, vartojama "srauto" sąvoka. Srautas – tai abstrakcija, leidžianti atsiriboti nuo konkretaus I/O įrenginio (disko, juostos, terminalo). Srautai būna dviejų tipų:

- **Tekstinis srautas** į eilutes, atskiriamas specialiais simboliais (`CR` – *carriage return*, `LF` – *line feed*), suskaidyta simbolių seka. Tekstinio srauto informacija nebūtinai yra identiška duomenims matomiems konkrečiame atvaizdavimo įrenginyje (spausdintuve, displėjuje), nes dalis simbolių (`CR`, `LF`) yra skirta ne vaizdavimui, o srauto perdavimo valdymui;
- **Dvejetainis srautas** – bitų seka, tiksliai atitinkanti informaciją konkrečiame įrenginyje.

Kiekviena programa automatiškai atidaro 5 standartinius tekstinius srautus:

`stdin` (įvedimas),
`stdout` (išvedimas),
`stderr` (pranešimai apie klaidas),
`stdaux` (papildomas),
`stdprn` (spausdintuvas).

Jie susiejami su standartiniais sisteminiais I/O įrenginiais. Dažniausiai srautai `stdin`, `stdout`, `stderr` yra susiejami su konsole (klaviatūra ir ekranas). Standartinį I/O įrenginį galima pakeisti (perskirti) DOS priemonėmis.

Standartinių srautų (klaviatūros ir ekrano) valdymui vartojamos funkcijos `printf` ir `scanf`. Jų prototipai:

```
int printf (const char *Šablonas, [<Kintamųjų sąrašas>]);
int scanf(const char *Šablonas, <Atminties laukų sąrašas>);
```

Šablonas – tai simbolių eilutė su įterptais rašomų arba skaitomų duomenų formatais, kurie aprašo duomenims skiriamo lauko struktūrą arba jų interpretavimo būdą. Funkcijoje `scanf` skaitomiems duomenims skiriami atminties laukai yra nurodomi adresais. Tipinė formato struktūra yra tokia:

%[<Požymis**>][<**Lauko dydis**>]**
[.<Tikslumas**>]<**Duomenų tipas**>**

Požymiai dažniausiai naudojami tokie:
– – sulygiuoti pagal kairįjį kraštą;
+ – išvedant skaičių, nurodyti jo ženklą (+ ar –);

Duomenų tipai žymimi raidėmis. Pagrindiniai tipai yra tokie:

`d` arba `i` – sveikasis,	`o` – aštuntainis sveikasis,
`u` – sveikasis, be ženklo,	`x` – šešioliktainis sveikasis,
`c` – simbolinis tipas,	`f` – slankaus kablelio,
`s` – simbolių eilutė,	`e` – rodiklinė forma,
	`p` – rodyklė.

1.7 pratimas. Programa iliustruoja komandos **`printf`** veikimą.

```
void main() {
  int a;
  long int b;
  short int c;
  unsigned int d;
  char e;
  float f;
  double g;

  a = 1023;
  b = 2222;
  c = 123;
  d = 1234;
  e = 'X';
  f = 3.14159;
  g = 3.1415926535898;

  printf("1. a = %d\n", a);        // dešimtainis skaičius
  printf("2. a = %o\n", a);        // aštuntainis skaičius
  printf("3. a = %x\n", a);        // šešioliktainis skaičius
  printf("4. b = %ld\n", b);       // dešimtainis ilgas skaičius
```

```
  printf("5. c = %d\n", c);         // dešimtainis trumpas
  printf("6. d = %u\n", d);         // be ženklo
  printf("7. e = %c\n", e);         // simbolis
  printf("8. f = %f\n", f);         // realus skaičius
  printf("9. g = %f\n", g);         // dvigubo tikslumo realus sk.
  printf("\n");
  printf("10. a = %d\n", a);        // paprastas sveikas skaičius
  printf("11. a = %7d\n", a);       // lauko plotis – 7 simboliai
  printf("12. a = %-7d\n", a);      // lygiuojama pagal kairį kraštą
  c = 5;
  d = 8;
  printf("13. a = %*d\n", c, a);    // lauko plotis – 5 simboliai
  printf("14. a = %*d\n", d, a);    // lauko plotis – 8 simboliai
  printf("\n");
  printf("15. f = %f\n", f);        // paprastas realus skaičius
  printf("16. f = %12f\n", f);      // lauko plotis – 12 simbolių
  printf("17. f = %12.3f\n", f);    // realiai daliai – 3 simboliai
  printf("18. f = %12.5f\n", f);    // realiai daliai – 5 simboliai
  printf("19. f = %-12.5f\n", f); // lygiuojama pagal kairį kraštą
}
```

Įvykdžius programą ekrane matysime:

```
1. a = 1023
2. a = 1777
3. a = 3ff
4. b = 2222
5. c = 123
6. d = 1234
7. e = X
8. f = 3.141590
9. g = 3.141593

10. a = 1023
11. a =    1023
12. a = 1023
13. a =  1023
14. a =     1023

15. f = 3.141590
16. f =     3.141590
17. f =        3.142
18. f =      3.14159
19. f = 3.14159
```

1.5. Ciklai ir sąlyginės išraiškos

Ciklo `while` sintaksė yra tokia:

`while (`⟨`kartojimo sąlyga`⟩**`)`** ⟨`kartojamas sakinys`⟩`;`

Kartojamas sakinys yra vykdomas tol, kol kartojimo sąlyga yra tenkinama (`true`). Jei ciklo viduje kintamasis, nuo kurio priklauso kartojimo sąlygos nutraukimas, nėra keičiamas, ciklas taps "amžinu". Ciklas nei karto nebus vykdomas, jeigu kartojimo sąlyga jau pradžioje bus netenkinama (`false`). Jeigu ciklą sudaro keletas sakinių, tuomet jie skliaudžiami skliaustais `{}`. Taip gaunamas sudėtinis sakinys.

1.8 pratimas. Programa iliustruoja ciklo **`while`** veikimą.

```
void main() {
  int count = 0;
  while(count < 6) {
    printf("count = %d\n", count);
    count++; }
}
```

Įvykdžius programą ekrane matysime:

```
count = 0
count = 1
count = 2
count = 3
count = 4
count = 5
```

Panašus yra ciklas `do-while`. Jo aprašo sintaksė:

`do` ⟨`kartojamas sakinys`⟩ **`while (`**⟨`kartojimo sąlyga`⟩**`);`**

Nuo ciklo `while` šis ciklas skiriasi tuo, kad jis visuomet bus vykdomas bent vieną kartą, nes kartojimo sąlyga tikrinama jau įvykdžius kartojamą sakinį.

1.9 pratimas. Programa iliustruoja ciklo **`do-while`** veikimą.

```
void main() {
  int i = 0;
  do {
    printf("i = %d\n", i);
    i++;
  } while(i < 5);
```

```
}
```

Įvykdžius programą ekrane matysime:

```
i = 0
i = 1
i = 2
i = 3
i = 4
```

Ciklo `for` sintaksinė struktūra:

`for` **(**⟨`i1`⟩`;` ⟨`i2`⟩`;` ⟨`i3`⟩**)** ⟨`kartojamas sakinys`⟩`;`

Sakinyje nurodyta išraiška `i1` skaičiuojama tik vieną kartą, prieš pradedant ciklą, kuomet tikrinama, ar tenkinama `i2` išraiška. Jei taip – vykdomas kartojimo sakinys. Išraiškos `i3` reikšmė perskaičiuojama po kartojimo sakinio vykdymo, ir grįžtama prie išraiškos `i2` tikrinimo.

Taigi, galime apibendrinti, kad `i1` apibrėžia ciklo parametro pradinę reikšmę, `i2` nurodo sustojimo sąlygą, o `i3` – kitimo dėsnį.

1.10 pratimas. Programa iliustruoja ciklo **`for`** veikimą.

```
void main() {
  int i;
  for(i=0; i<6; i++)
    printf("i = %d\n", i);
}
```

Įvykdžius programą ekrane matysime:

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
```

Valdymo struktūros. Pradėsime nuo vienos iš populiariausių – struktūros `if-else`. Jos sintaksė yra tokia:

`if` **(**`sąlyga`**)** **<**`šaka TAIP`**>`;**
`else` **<**`šaka NE`**>`;**

C^{++} kalboje nėra loginio duomenų tipo, todėl santykiams suteikiama skaitmeninė reikšmė, kuri yra `1`, kai santykio operacija tenkinama, ir – `0`,

kai santykio operacija netenkinama. C++ kalboje sąlygas galima aprašyti ne tik santykiais, bet ir bet kokiomis kitomis skaitmeninėmis išraiškomis. Jei tokios išraiškos reikšmė yra `0`, laikoma, kad sąlyga netenkinama, o jei reikšmė kitokia, laikoma, kad sąlyga tenkinama. Dar viena įdomi C++ kalbos savybė – kalboje nėra loginio tipo, bet logines išraiškas vartoti galima. Loginių išraiškų skaitmeniniai argumentai, kurių reikšmės nenulinės ($\neq 0$), interpretuojami kaip loginės reikšmės `TRUE`, o nulinės ($=0$) laikomos reikšmėmis `FALSE`. Loginių išraiškų reikšmės taip pat būna skaitmeninės – `0` (`FALSE`) ir `1` (`TRUE`).

Programuojantiems Paskalio kalba, siūloma atkreipti dėmesį į tai, kad:

- C++ kalbos sakinyje `if` prieš šaką `else` yra rašomas **;** (kabliataškis).
- Įvertinant tai, kad loginės reikšmės C++ kalboje nėra, teisingas bus užrašas:

```
if (Ats) cout << "giedras dangus!!!\n";
else     cout << "danguje labai daug debesu\n";
```

Čia `else` šaka bus vykdoma tik esant atsakymo nulinei reikšmei,– kitais atvejais bus vykdoma šaka `TRUE`. Jeigu programos vartotojas netiksliai atsakys į programos klausimą, tuomet šis sakinys ir programos pavyzdyje esantis sakinys dirbs skirtingai. Aukščiau pateiktoje programoje **1.1 pratimas** `else` šaka bus vykdoma visais atvejais, išskyrus atsakymą lygų `1`.

1.11 pratimas. Programa iliustruoja valdymo struktūros **`if-else`** veikimą.

```
void main() {
  int data;

  for (data=0; data<10; data++) {
    if (data == 2)  printf("data = %d\n", data);
    if (data < 5)
      printf("data = %d, t.y. maziau uz 5\n", data);
    else
      printf("data = %d, t.y. daugiau uz 4\n", data);
} }
```

Įvykdžius programą ekrane matysime:

```
data = 0, t.y. maziau uz 5
data = 1, t.y. maziau uz 5
data = 2
```

```
data = 2, t.y. maziau uz 5
data = 3, t.y. maziau uz 5
data = 4, t.y. maziau uz 5
data = 5, t.y. daugiau uz 4
data = 6, t.y. daugiau uz 4
data = 7, t.y. daugiau uz 4
data = 8, t.y. daugiau uz 4
data = 9, t.y. daugiau uz 4
```

Dar du nauji valdymo sakiniai, tai `break` ir `continue`. Šie sakiniai neturi jokių parametrų ir dažniausiai yra naudojami cikluose; `break` nutraukia ciklo vykdymą ir veiksmai tęsiami nuo pirmo sakinio, esančio už ciklo, o `continue` nutraukia tik vienos iteracijos vykdymą ir sugrąžina į kartojimo sąlygos tikrinimą.

1.12 pratimas. Programa iliustruoja sakinių **`break`** ir **`continue`** veikimą.

```
void main() {
  int xx;

  for (xx=5; xx<15; xx++) {
    if (xx == 8)
      break;
    printf("Break ciklas; xx = %d\n", xx);
  }

  for (xx=5; xx<15; xx++) {
    if (xx == 8)
      continue;
    printf("Continue ciklas; xx = %d\n", xx);
} }
```

Įvykdžius programą ekrane matysime:

```
Break ciklas; xx = 5
Break ciklas; xx = 6
Break ciklas; xx = 7
Continue ciklas; xx = 5
Continue ciklas; xx = 6
Continue ciklas; xx = 7
Continue ciklas; xx = 9
Continue ciklas; xx = 10
Continue ciklas; xx = 11
Continue ciklas; xx = 12
Continue ciklas; xx = 13
```

```
Continue ciklas; xx = 14
```

`switch-case` struktūra yra tokia:

```
switch (išraiška) {
  case žymė1: sakinys1;
  case žymė2: sakinys2;
  default   : sakinys3;  }
```

Jei išraiška yra lygi `žymė1`, bus vykdomi `sakinys1`, `sakinys2` ir `sakinys3`; jei lygi `žymė2` – `sakinys2` ir `sakinys3`, o visais likusiais atvejais – tik `sakinys3`. Tačiau dažniau ši struktūra yra naudojama kartu su operatoriumi `break`:

```
switch (išraiška) {
  case žymė1: sakinys1; break;
  case žymė2: sakinys2; break;
  default   : sakinys3; break;  }
```

Šiuo atveju, jei išraiška lygi `žymė1`, vykdomas tik `sakinys1`; jei lygi `žymė2` – tik `sakinys2`, o likusiais atvejais – `sakinys3`.

1.13 pratimas. Programa iliustruoja struktūros **`switch-case`** veikimą.

```
void main() {
  int i;

  for(i=3; i<13; i++) {
    switch (i) {
      case 3 : printf("Trys\n");
               break;
      case 4 : printf("Keturi\n");
               break;
      case 5 :
      case 6 :
      case 7 :
      case 8 : printf("Penki - astuoni\n");
               break;
      case 11 : printf("Vienuolika\n");
               break;
      default : printf("Neaprasyta reiksme\n");
               break;
} } }
```

Įvykdžius programą ekrane matysime:

```
Trys
Keturi
Penki - astuoni
Penki - astuoni
Penki - astuoni
Penki - astuoni
Neaprasyta reiksme
Neaprasyta reiksme
Vienuolika
Neaprasyta reiksme
```

2 skyrius. Masyvai, failai, eilutės

2.1. Rodyklė, adresas

Kiekvienam programoje aprašytam kintamajam kompiliatorius skiria atminties lauką, kurį apibūdina visa grupė parametrų: *adresas, lauko dydis, saugomų duomenų tipas ir reikšmė*. Programos tekste panaudotas kintamojo vardas nurodo, kad turi būti manipuliuojama su jo reikšme. Apsiriboti manipuliavimu vien tik kintamųjų reikšmėmis galima tik paprastus skaičiavimo algoritmus aprašančiose programose. Sudėtingesnėse taikomosiose ir sisteminėse programose, kuriose yra aktualūs racionalaus dinaminio atminties paskirstymo, dinaminių struktūrų sudarymo ir kompiuterio įrangos valdymo fiziniame lygyje klausimai, tenka manipuliuoti ne tik atmintyje saugomomis kintamųjų reikšmėmis, bet ir jų adresais. Adresų reikšmių saugojimui yra skirtos rodyklės, kurių aprašų sintaksė:

⟨`lauko tipas`⟩ *****⟨`rodyklės vardas`⟩

Programos tekste struktūra `*`⟨`rodyklės vardas`⟩ reiškia, kad turi būti vartojama rodyklės rodomo lauko reikšmė. Pačioms rodyklėms gali būti suteikiamos tik to paties tipo objektų, kuriems buvo apibrėžtos rodyklės, adresų reikšmės.

Pavyzdys:

```
int x, *px;
px = &x;
```

Priskyrimo operatorius `px = &x` rodyklei `px` suteikia kintamojo `x` adreso reikšmę, todėl į `x` reikšmę dabar galima kreiptis tiek vardu `x`, tiek struktūra `*px`. Rodyklės tipo kintamiesiems būtina nurodyti pradinę reikšmę, kuri dažniausiai būna "tuščias adresas": `px = NULL`.

2.1 pratimas. Programa iliustruoja, kaip naudojamos rodyklės. Kintamieji `pt1` ir `pt2` yra rodyklės (tai rodo žvaigždutės prieš juos). Užrašą "`&index`" galime skaityti kaip "kintamojo `index` adresas". O realiai šioje programoje yra tik vienas kintamasis, kurio reikšmę keičiame.

```
main() {
  int index, *pt1, *pt2;

  index = 39;
  pt1 = &index;
  pt2 = pt1;
  printf("index *pt1 *pt2");
  printf("%4d %4d %4d\n", index, *pt1, *pt2);
  *pt1 = 13;
  printf("%4d %4d %4d\n", index, *pt1, *pt2);
}
```

Įvykdžius programą ekrane matysime:

```
index *pt1 *pt2
  39   39   39
  13   13   13
```

2.2. Vienmatis masyvas

Labai glaudžiai su rodyklėmis yra susijęs masyvo tipas, kuris yra skirtas vienodo tipo duomenų (elementų) rinkiniams saugoti. Masyvų aprašų sintaksė:

⟨elementų tipas⟩ ⟨masyvo vardas⟩ **[**⟨elementų skaičius⟩**]**

Šiame aprašyme skliaustai **[ ]** yra būtini struktūros elementai. Masyvo narius žymi kintamieji su indeksais, kurių vardų sintaksė:

⟨masyvo vardas⟩**[**⟨indeksas⟩**]**

Pavyzdžiui, masyvo aprašas: `int A[10];` masyvo elementas: `A[5]`.

Indeksais gali būti tik intervalo `[0, n-1]` sveikosios reikšmės (`n` – masyvo elementų skaičius).

Pastaba: *Masyvo vardas be indekso reiškia jo nulinio elemento adresą.*

2.2 pratimas. Programa demonstruoja situaciją, kai gretimuose atminties laukuose surašyti vienodo tipo duomenys **(a, b, c, d)** panaudoti masyvo formavimui. Pradžią nurodo pirmojo kintamojo adresas: **pi = &a;**

```
#include <iostream.h>

int  a=1,  b=2,  c=3,  d=4,  *pi;
```

```
main() {
  int i = 0, mas[4];
  pi = &a;                                  // Kintamojo a adresas
  while(i<4)  mas[i++] = *(pi++);           // Masyvo formavimas
                          // Masyvo išvedimas į ekraną atvirkščia tvarka
  while(i)  cout << mas[--i] << "\t";
  cout << endl;
}
```

Kai reikšmių didinimo ar mažinimo operacijos yra taikomos rodyklėms, jų reikšmės yra keičiamos ne vienetu, o su rodykle susieto **lauko dydžiu**. Ši rodyklių savybė pavyzdžio programoje yra panaudota kintamųjų grupės `a, b, c, d` reikšmių perrinkimui.

✓Papildykite programą taip, kad ji parodytų ekrane visų joje vartojamų kintamųjų ir masyvo `mas` elementų adresus. Aprašant adresų išvedimą, naudokite rodyklių šabloną `%p`. Adresai bus išvedami šešioliktainėje skaičiavimo sistemoje.

2.3 pratimas. Reikia surasti masyvo elementų sumą. Duomenys įvedami klaviatūra. Reikšmėms sumuoti padaroma funkcija, kuri rezultatą grąžina funkcijos vardu. Funkcijai perduodamas masyvo pirmojo elemento adresas: **`A`** (galima rašyti **`&A[0]`**).

```
#include <iostream.h>

int Suma(int *, int);
//                  Pagrindinė programa
void main() {
  int  A[10], i, n;              // A[0], A[1], ... , A[9]
  cout << "Iveskite n reiksme: ";
  cin >> n;
  for (i=0; i<n; i++) {
      cout << "A[ " << i << " ]= ";
      cin >> A[i];
  }
  cout << " Suma = " << Suma(A, n) << endl;
}
//                     Funkcija Suma
int Suma(int *pr, int k) {
  int S = 0, i = 0;
  while(i<k) { S += *(pr+i);  i++;  }
  return S; }
```

2.3. Dinaminis atminties išskyrimas

Dirbant su masyvais būtina įsitikinti, kad norimas duomenų skaičius tilps kompiuterio atmintyje. Galimi keli būdai išskirti atminčiai. Vienas paprasčiausių yra funkcija:

```
void *malloc(Kiekis);
```

Čia nurodomas prašomos atminties `Kiekis` baitais. Jeigu atmintis buvo skirta, tai grąžinama rodyklė į pirmąjį tos atminties baitą, kitaip funkcijos reikšmė yra `NULL`.

Kreipinio metu atminties kiekį patogu nurodyti kaip duomenų elementų skaičiaus ir vieno elemento užimamos vietos baitais sandaugą, pavyzdžiui:

```
n * sizeof(float)
```

kur `n` – realių skaičių kiekis, o funkcija `sizeof` grąžina nurodyto duomenų tipo (čia realaus) apimtį baitais.

Funkcija `malloc` grąžina rodyklę į nurodyto tipo atmintį: kreipinio metu nurodomas rodyklės tipas.

2.4 pratimas. Klaviatūra įvedamas duomenų skaičius. Jeigu duomenims saugoti yra vietos atmintyje, tai duomenys įvedami klaviatūra į masyvą, kurio rodyklė yra saugoma kintamuojame **`A`**. Toliau duomenys surikiuojami mažėjimo tvarka.

```
#include <iostream.h>
#include <alloc.h>
//           Atminties skyrimas ir duomenų įvedimas
int *Duomenys(int);
void Tvarka (int *, int);    // Rikiavimas
void Sp     (int *, int);    // Išvedimas ekrane
//                  Pagrindinė programa
void main() {
  int n, *A;
  cout << "Iveskite n reiksme: ";
  cin >> n;
  A = Duomenys(n);
  if (A != NULL) {
    cout << " n= " << n << endl;
    cout << " Ivestas masyvas:\n";
    Sp(A, n);
    Tvarka(A, n);
    cout << " Sutvarkytas masyvas:\n";
```

```
    Sp(A, n);
    free(A); }              // Masyvo užimama atmintis atlaisvinama
 cout << endl;
}
//                      Duomenų įvedimas
int *Duomenys(int n) {
  int *x, i;
  x = (int *) malloc(n * sizeof(int));
  if (x == NULL)
     cout << "Truksta atminties" << endl;
  else {
     for(i=0; i<n; i++) {
        cout << "Iveskite " << (i+1) << "-a elementa ";
        cin >> *(x+i); }
     cout << endl;    }
  return x;
}
//                      Rikiavimas
void Tvarka(int *x, int n) {
  int i, j, c;
  for(i=0; i<n-1; i++)
     for(j=i+1; j<n; j++)
        if (*(x+j) > *(x+i) ) {
           c = *(x+i);
           *(x+i) = *(x+j);
           *(x+j) = c;  }
}
//                      Išvedimas ekrane
void Sp(int *x, int n) {
  int i = 0;
  while(i < n) {
     cout << (i+1) << "-elementas: " << *(x+i) << endl;
     i++;  }
}
```

✓ Atminties išskyrimui patogu naudoti operatorių `new`. Pakeiskite **2.3 pratimo** funkcijos `Duomenys` eilutę:

```
x = (int *) malloc(n * sizeof(int));
```

nauja eilute:

```
x = new int[n];
```

Išbandykite programą. Išsiaiškinkite `new` galimybes, taikymą dvimačiams masyvams.

2.4. Dvimatis masyvas

Dvimačio masyvo (matricos) aprašas:

```
⟨duomenų tipas⟩ ⟨masyvo vardas⟩
          [⟨eilučių skaičius⟩] [⟨stulpelių skaičius⟩];
```

Pavyzdžiui, užrašas `int A[10][8]` rodo, kad bus 10 eilučių (0, 1, 2, … , 9) ir kiekviena jų turės po 8 elementus (stulpeliai 0, 1, 2, …, 7).

Matricos elementas veiksmuose nurodomas dviem indeksais, kurie rašomi atskiruose laužtiniuose skliaustuose: `A[2][3]`.

Sudėtingesniuose uždaviniuose tikslinga sukurti savo duomenų tipus. Masyvo tipas sukuriamas taip:

```
typedef ⟨masyvo elementų tipas⟩ ⟨masyvo tipo vardas⟩
         ⟨indeksų aprašas⟩;
```

Pavyzdžiui:

```
typedef int Masyvas [55];
typedef float Matrica [10][15];
const Kiek = 15
typedef double Lentele [Kiek][Kiek];
```

Čia buvo sukurti fiksuotos apimties masyvų tipai.

Dabar galima aprašyti keletą to paties tipo masyvų:

```
Masyvas A, B, C;
Matrica P, D, R;
Lentele Pirma, Antra;
```

2.5 pratimas. Klaviatūra įvedami duomenys, kurie saugomi matricoje. Suskaičiuojama matricos kiekvienos eilutės elementų suma. Rezultatai surašomi į vienmatį masyvą.

```
#include <iostream.h>
#include <alloc.h>
typedef int mas [10];
typedef int matr[10][15];

int  Duomenys(matr, int *, int *);  // Duomenų įvedimas
void Sp      (mas, int);            // Spausdinimas
int  Suma    (mas, int);            // Masyvo elementų suma
```

```
//                    Pagrindinė programa
void main() {
  matr A;   mas S;
  int n, m, i, j;

  Duomenys(A, &n, &m);          // Duomenų įvedimas klaviatūra
  if (A != NULL) {
    cout << "----------------------\n";
    for (i=0; i<n; i++)         // Matricos spausdinimas
      for (j=0; j<m; j++) cout << A[i][j] << endl;
    cout << "----------------------\n";
    for (i=0; i<n; i++) {       // Matricos spausdinimas
      cout << "Eilute Nr " << (i+1) <<": ";
      Sp(A[i], m);  }           // Vienos eilutės spausdinimas
    cout << "----------------------\n";     // Skaičiavimai
    for (i=0; i<n; i++) S[i] = Suma(A[i], m);
    Sp (S, n);  }               // Suformuoto masyvo spausdinimas
  cout << endl;
}
//                    Duomenų įvedimas
int Duomenys(matr D, int *n, int *m) {
  int i, j;
  cout << "Iveskite n reiksme: ";
  cin >> *n;
  cout << "Iveskite m reiksme: ";
  cin >> *m;
  for (i=0; i<*n; i++) {
    cout << "Eilute Nr:" << (i+1) << endl;
    for (j=0; j<*m; j++) {
      cout << "Iveskite " << (j+1) << "-a elementa ";
      cin >> D[i][j];  }  }
  cout << endl;
  return 1;  }
}
//                    Spausdinimas
void Sp(mas x, int n) {
  int i = 0;
  while (i<n) cout << x[i++] << "  " << endl;  }
//            Masyvo elementų sumos skaičiavimas
int Suma(mas D, int k) {
  int i, Sk=0;
  for (i=0; i<k; i++) Sk += D[i];
  return Sk;  }
```

✓ Galima turėti funkciją, kuri suformuotų rezultatų masyvą. Išbandykite šią funkciją:

```
// funkcijos prototipas
void Suma (mas, matr, int, int);
```

Ši funkcija, visų matricos eilučių elementų sumas surašo į vienmatį masyvą, kurio vardas nurodomas kreipinio metu pirmuoju parametru.

```
void Suma(mas R, matr D, int eil, int st ) {
  int i, j;
  for (i=0; i<eil; i++) {
    *(R+i) = 0;
    for(j=0; j<st; j++) *(R+i) += D[i][j]; }
}
```

2.5. Duomenų failai

Apdorojant didesnės apimties duomenų srautus, patogu duomenis saugoti failuose. Failo kintamieji aprašomi tipo `FILE` rodykle:

```
FILE *F;
```

Failai paruošiami darbui funkcija, kurios prototipas toks:

```
#include <stdio.h>
FILE *fopen(const char *FailoVardas, const char *M);
```

Čia `FailoVardas` nurodomas kaip simbolių eilutė: konstanta tarp kabučių, arba konstantos vardu, arba kintamuoju, turinčiu tą reikšmę. Failo paruošimo darbui būvis nurodomas antruoju parametru `M` (simbolinė eilutė: konstanta, jos vardas arba kintamasis). Galimos būvio reikšmės surašytos 2.1 lentelėje.

2.1 lentelė. Failo paruošimo darbui būsenos

`r`	tik skaitymui;
`w`	tik rašymui; jeigu failas egzistavo, tai naikinamas ir sukuriamas naujai;
`a`	papildymui gale; jeigu failo nebuvo, tai sukuriamas;

Būvio reikšmę papildžius raide `t (rt, wt, at)`, failas bus paruošiamas darbui kaip tekstinis. Būvio reikšmę papildžius raide `b (rb, wb, ab)`, failas bus paruošiamas darbui kaip binarinis. Jeigu nebus panaudota nei `t`, nei `b` raidės, tai failas bus atidaromas darbui priklausomai nuo globalinio kintamojo – `fmode` reikšmės (žr. `fcntl.h`). Kiekvienas būvio variantas

gali būti papildytas ženklu + (plius) (pvz.: `r+`, `w+`, `wt+`). Tai reiškia, kad failai paruošiami atnaujinimui (leidžiama skaityti ir rašyti).

Toliau pristatome pagrindinių buferizuotų I/O funkcijų prototipus.

```
int fclose(FILE *rod);                          //      failo uždarymas
int putc(int simbolis, FILE *rod);              //    simbolio rašymas
int getc(FILE *rod);                            //  simbolio skaitymas
int putw(int sk, FILE *rod);                    //    skaičiaus rašymas
int getw(FILE *rod);                            //  skaičiaus skaitymas
char *fgets(char *s, int n, FILE *rod);
          // skaito eilutę s iš n simbolių
int fputs(const char *s, FILE *rod);    //    eilutės s rašymas
int fread(void *buf, int t, int n, FILE *rod);
          // siunčia n po t baitų turinčių blokų iš failo rod į buferį buf;
          // funkcijos reikšmė rodo perskaitytų blokų skaičių
int fwrite(void *buf, int t, int n, FILE *rod);
          // Siunčia n po t baitų turinčių blokų iš buferio buf į failą rod;
          // funkcijos reikšmė rodo perskaitytų blokų skaičių
int fprintf(FILE *rod, const *char f [,<sąrašas>]);
          // į failą rod, naudojant šabloną f, rašomi sąrašo elementai;
          // šablonas ir argumentų sąrašas sudaromi taip, kaip ir funkcijai
          // printf
int fscanf(FILE *rod, const *char f [,<sąrašas>]);
          // skaitymas pagal šabloną. Funkcijos reikšmė –
          // sėkmingai perskaitytų elementų skaičius
int feof(FILE *rod);    //failo pabaiga – nenulinė funkcijos reikšmė
int ferror(FILE *rod); //        klaida – nenulinė funkcijos reikšmė
int remove(const *char vardas);         //      šalinamas failas
```

Failų apdorojimo programoms apdorojamų failų vardai gali būti perduodami pagrindinės funkcijos parametrais. Funkcijos prototipas:

```
int main(int n, char *argv[ ]);
```

Čia `n` – argumentų skaičius, o `argv` – argumentams skirto eilučių masyvo rodyklė. Argumentų reikšmės yra nurodomos programos iškvietimo komandoje ir yra atskiriamos tarpais. Nulinė argumentų masyvo eilutė – tai pačios programos pavadinimas.

2.6 pratimas. Duomenų faile "`duom6.dat`" surašyti skaičiai. Reikia duomenis iš failo surašyti į masyvą ir atspausdinti ekrane. Suskaičiuoti masyvo elementų sumą ir atspausdinti ekrane.

```
#include <io.h>
#include <stdio.h>
#include <iostream.h>
#include <conio.h>

const Kiek = 15;
typedef int mas [Kiek];
FILE *F;

void Sp   (mas, int);      // Spausdina masyvą
int  Suma (mas, int);      // Sumuoja masyvo elementus
//                  Pagrindinė programa
void main() {
  mas A;
  int n = 0;

  F = fopen("duom6.dat", "r");
  if (F == NULL) printf("Failas neatidarytas");
  else {
    while (!feof(F) && ( n < Kiek)) {
      fscanf(F, "%d", &A[n]); n ++; }
    Sp(A, n);
    printf("Suma=%5d\nPabaiga", Suma(A, n)); }
}
//                  Spausdina masyvą
void Sp(mas X, int n) {
  int i = 0;
  while (i<n) cout << X[i++] << "  " << endl;
}
//                  Sumuoja masyvo elementus
int Suma(mas R, int k) {
  int i = 0, S = 0;
  while (i<k) S += R[i++];
  return S;
}
```

2.7 pratimas. Duomenų faile "duom7.dat" surašyti skaičiai. Reikia duomenis iš failo surašyti į masyvą. Masyvo elementų reikšmės atspausdinamos kitame faile.

```
#include <iostream.h>
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
#include <stdlib.h>

const Kiek = 10;
typedef int mas [Kiek];

void SpF(mas, int, char);     // Masyvo elementų išvedimas į failą
int Ivesti(mas, int *);       // Skaitymas iš failo
//                    Pagrindinė programa
void main() {
  mas A;
  int n = 0;
  if (Ivesti(A, &n)) {
    cout << "\nDuomenų failo nėra\n";
    getch();
    exit(1);  }
  SpF(A, n, 'A');
}
//                      Skaitymas iš failo
int Ivesti(mas A, int *n) {
  FILE *F;
  F = fopen("duom7.dat", "r");
  if (F == NULL) {
    printf( "Failas neatidarytas");
    return 1; }
  else {
    while(!feof(F) && (*n < Kiek)) {
      fscanf(F, "%d", &A[*n]);
      n++; }          // Borland C++4.5 realizacijoje rašoma (*n)++
    fclose(F);
    return 0; }
}
//                Masyvo elementų išvedimas į failą
void SpF(mas X, int n, char R) {
  FILE *F;
  int i = 0;
  F = fopen("duom7.rez", "w");
```

```
  if (F == NULL)
    printf("Failas neatidarytas spausdinimui");
  else {
    while (i<n)
      fprintf(F, "%c[%2d]=%4d\n", R, i, X[i++]);
    fclose(F);  }
}
```

Duomenų failas: "Duom7.dat"
5 6 98 -5 12 3 4 5 21 -3 4 5 23 1 23 34 45 5 6 -123

Rezultatų failas: "Duom7.rez"
A[1]= 5 A[2]= 6 A[3]= 98 A[4]= -5 A[5]= 12 A[6]= 3 A[7]= 4 A[8]= 5 A[9]= 21 A[10]= -3

✓ Duomenų įvedimo funkcijoje **Ivesti** skaitymo ciklą pakeiskite tokiu:

```
while((fscanf(F, "%d", &A[*n]) != EOF ) &&
                                  (*n < Kiek)) *n++;
```

✓ Dabar duomenų įvedimo funkcijoje skaitymo ciklą pakeiskite tokiu:

```
while((fscanf(F, "%d", &A[*n]) != EOF ) &&
                                     (*n++ < Kiek));
```

Palyginkite visus tris variantus ir pasirinkite, kuris Jums suprantamiausias. Siūlome programas rašyti paprastesnes ir aiškesnes, nes jas lengviau ir greičiau suvokti ne tik Jums, po kurio laiko modifikuojant, bet ir kitiems, turintiems mažiau praktinio programavimo įgūdžių.

2.8 pratimas. Duomenų faile surašytos taškų, esančių koordinačių plokštumoje, koordinatės **(x, y)**. Reikia duomenis surašyti į matricą, kur pirmame stulpelyje būtų taškų **x** koordinatės, antrame stulpelyje **y** koordinatės. Parašyti funkciją, kuri matricos trečiame stulpelyje surašytų taškų atstumus iki koordinačių pradžios taško. Ketvirtame stulpelyje pažymėti, kokiam ketvirčiui taškas priklauso. Nuliuku žymėti taškus, esančius ant ašių. Rezultatų faile atspausdinti

matricos duomenis, užrašant kiekvieno stulpelio prasmę nusakančius pavadinimus ir numeruojant eilutes. Gale parašyti, kiek kuriame ketvirtyje yra taškų, ir kiek yra taškų ant ašių.

```
#include <iostream.h>
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

typedef float mas[][4];
const   char  *Duomenys   = "Duom8.dat";
const   char  *Rezultatai = "Duom8.rez";

void RezFailas();                    // Rezultatų failo paruošimas
int  Failas();                       // Randa kiek yra duomenų
mas *Ivesti(int);                    // Duomenų įvedimas
void Spausdinti(mas *, int, int);    // Rezultatų spausdinimas
void Atstumai(mas *, int);           // Taškų atstumai
void Vieta(mas *, int);              // Taškų padėtis plokštumoje
//                  Pagrindinė programa
  void main() {
  mas *A;
  int n;
  RezFailas();
  if((n = Failas()) == 0) {
    cout<<" Duomenu nerasta\n";
    getch();
    exit(0);  }
  A = Ivesti(n);
  if (A == NULL) {
    cout<<"NERA NERA \n";
    getch();
    exit(1);  }
  cout << "Masyvas ivestas n = "<< n << endl;
  Spausdinti(A, n, 0);
  Atstumai(A, n);
  Spausdinti(A, n, 1);
  Vieta(A, n);
  Spausdinti(A, n, 2);
}
```

```
//          Paruošia rezultatų failą. Užrašo datą ir laiką
void RezFailas() {
  FILE *F;
  char Data[9], Laikas[9];
  if(F = fopen(Rezultatai, "w")) {
    _strdate(Data);
    _strtime(Laikas);
    fprintf(F, "%s %s\n", Data, Laikas);
    fclose(F);  }
}
//       Patikrina, ar yra duomenų failas ir kiek jame taškų
int Failas() {
  FILE *F;
  int n = 0, k;
  if ((F = fopen(Duomenys, "r")) == NULL ) {
    cout << "Duomenu failo nera\n";
    return 0; }
  while(!feof(F)) {
    n++;
    fscanf(F, "%d%d\n", &k, &k);  }
  fclose(F);
  return n;
}
//              Skaito duomenis iš failo į masyvą
mas *Ivesti(int n) {
  FILE *F;
  mas *A;
  if ((F = fopen(Duomenys, "r")) == NULL) {
     cout << "Duomenu failas neatidarytas\n";
     getch();
     return NULL;  }
  A = (mas *) malloc(n*4*sizeof(float));
  if(A == NULL) {
    cout << "Truksta masyvui atminties\n";
    getch();
    return NULL;  }
  n = 0;
  while(!feof(F)) {
    fscanf(F, "%f%f\n", &(*A)[n][0], &(*A)[n][1]);
    n++;  }
  fclose(F);
  return A;
}
```

```
//                    Spausdina duomenis ir rezultatus
/* Raktas k valdo:
kai 0, spausdina tik taškų koordinates (duomenis);
kai 1, spausdina duomenis ir taškų atstumus iki koordinačių pradžios taško;
kai 2, spausdina duomenis, atstumus ir taškų vietas koordinačių plokštumoje.*/
void Spausdinti(mas *C, int n, int k) {
  FILE *F;
  int i;
  if ((F = fopen(Rezultatai, "a")) == NULL) {
    cout << "Rezultatu failas neatidarytas\n";
    getch();  return;  }
  switch(k) {
    case 0:
      fprintf(F, "Duomenys\n");
      fprintf(F, "**********************\n");
      fprintf(F, "  Nr.   x        y   \n");
      break;
    case 1:
      fprintf(F, "\nDuomenys ir atstumai iki koord.
                                      pradzios\n");
      fprintf(F, "******************************\n");
      fprintf(F, "  Nr.   x        y   Atstumas \n");
      break;
    case 2:
      fprintf(F, "\nDuomenys, atstumai ir tasko vieta
                                koord. plokstumoje\n");
     fprintf(F, "************************************\n");
     fprintf(F, "  Nr.   x        y   Atstumas   Vieta \n");
     break;
  }
  for(i=0; i<n; i++) {
    switch(k){
      case 0: fprintf(F," %3i %7.2f  %7.2f \n", i+1,
                               (*C)[i][0],(*C)[i][1]);
              break;
      case 1: fprintf(F, "%3i %7.2f  %7.2f %7.2f \n",
               i+1, (*C)[i][0],(*C)[i][1],(*C)[i][2]);
              break;
      case 2:
        fprintf(F, "%3i %7.2f  %7.2f %7.2f  %7.2f \n",
    i+1, (*C)[i][0],(*C)[i][1],(*C)[i][2],(*C)[i][3]);
        break;
  }}
  switch(k) {
```

```
    case 0:
      fprintf(F, "********************\n");
      break;
    case 1:
      fprintf(F, "****************************\n");
      break;
    case 2: fprintf(F,
          "************************************\n");
      break;
  }
  fclose(F);
}
//      Skaičiuojami taškų atstumai iki koordinačių pradžios taško
void Atstumai(mas *D, int n) {
  float x, y;
  for (int i = 0; i<n; i++) {
    x = (*D)[i][0];
    y = (*D)[i][1];
    (*D)[i][2] = sqrt(x*x + y*y);  }
}
//          Nustatoma taško padėtis koordinačių plokštumoje
/* 1–4 nurodo ketvirčio numerį, o 0 – kad taškas yra ant vienos iš ašių. */
void Vieta(mas *D, int n) {
  float x, y;
  for (int i=0; i<n; i++) {
    x = (*D)[i][0];
    y = (*D)[i][1];
    if ((x>0) && (y>0))  (*D)[i][3] = 1;
    else if ((x>0) && (y<0))  (*D)[i][3] = 4;
         else if ((x<0) && (y>0))  (*D)[i][3] = 2;
              else if ((x<0) && (y<0))  (*D)[i][3] = 3;
                   else (*D)[i][3] = 0;  }
}
```

Duomenų failo pavyzdys:

```
 15   12
 13   -5
-15  -45
-22   22
  0    5
  5    0
 -5    0
  0   -5
 -1   -1
```

Rezultatų failo pavyzdys:

```
06/08/99 21:48:34
Duomenys
*********************
  Nr.    x         y
   1   15.00     12.00
   2   13.00     -5.00
   3  -15.00    -45.00
   4  -22.00     22.00
   5    0.00      5.00
   6    5.00      0.00
   7   -5.00      0.00
   8    0.00     -5.00
   9   -1.00     -1.00
*********************

Duomenys ir atstumai iki koord. pradzios
*******************************
  Nr.    x         y   Atstumas
  1   15.00     12.00   19.21
  2   13.00     -5.00   13.93
  3  -15.00    -45.00   47.43
  4  -22.00     22.00   31.11
  5    0.00      5.00    5.00
  6    5.00      0.00    5.00
  7   -5.00      0.00    5.00
  8    0.00     -5.00    5.00
  9   -1.00     -1.00    1.41
*******************************

Duomenys, atstumai ir tasko vieta koord. plokstumoje
************************************
  Nr.    x         y   Atstumas   Vieta
  1   15.00     12.00   19.21      1.00
  2   13.00     -5.00   13.93      4.00
  3  -15.00    -45.00   47.43      3.00
  4  -22.00     22.00   31.11      2.00
  5    0.00      5.00    5.00      0.00
  6    5.00      0.00    5.00      0.00
  7   -5.00      0.00    5.00      0.00
  8    0.00     -5.00    5.00      0.00
  9   -1.00     -1.00    1.41      3.00
************************************
```

2.9 pratimas. Naudodamiesi C++ aplinkos pagalbine informacija (Ctrl+F1) išsiaiškinkite, kaip sudaryta struktūrizuoto tekstinio failo "Duom9.txt" papildymo programa. Failo eilutėse yra saugomi duomenys apie įvairiems asmenims priklausančius telefonus. Duomenų elementams yra skiriamos tokios eilučių atkarpos: pavardė – [1, 20], vardas – [21, 40], telefono numeris – [41,50]. Sukurkite tokios struktūros failą ir patikrinkite programą. Programos vykdymo pabaigos sąlygą išsiaiškinkite nagrinėdami jos tekstą.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

FILE *failas;
char buf[50], e1[20], e2[20], e3[10];
const char vardas[] = "Duom9.txt";

int  Init    (const char *, char *);
void Duomenys(FILE *);
void Rodyti  (FILE *);
//                  Pagrindinė programa
void main() {
  buf[0] = '\0';
  if (!Init(vardas, "a")) exit(1); // Failo paruošimas darbui
  Duomenys(failas);         // Duomenys įvedami klaviatūra į failą
  fclose(failas);

  if (!Init(vardas,"r")) exit(1); //Failo paruošimas skaitymui
  Rodyti(failas);               // Duomenys siunčiami į ekraną
  fclose(failas);
}
//  Naujo failo atidarymas skaitymui/rašymui arba seno papildymui
int Init (const char *v, char *mode) {
  if ((failas = fopen(v, mode)) == NULL) { // Failo nebuvo
                                    // Naujo sukurti nepavyko
    if ((failas = fopen(v, "w+")) == NULL) {
      fprintf(stderr,
          "Atidaryti failą rašymui/skaitymui nepavyko.\n");
      return 0;  }                  // Klaidos požymis
    else return 1; }                // Naujai sukurtas failas
  else return 2;                    // Surastas failas ir atidarytas
}
```

```
//                Duomenys įvedami klaviatūra į failą
void Duomenys(FILE *failas) {
  int i;
  while(buf[0] != 'q') {
             // Įvedimo pabaiga nurodoma eilute, kurios pirmas simbolis yra 'q'
    buf[0]= '\0';
             // Struktūrizuotos eilutės sudarymo ir rašymo ciklai
    printf("Nurodykite pavarde, varda ir telefono numeri:\n");
    scanf("%s %s %s", e1, e2, e3);
    strcat(buf, e1);
    for(i=strlen(buf); i<20; i++)  strcat(buf, " ");
    strcat(buf, e2);
    for(i=strlen(buf); i<40; i++)  strcat(buf, " ");
    strcat(buf, e3);
    strcat(buf, "\n");
    if (buf[0] != 'q')
      fwrite(buf, strlen(buf), 1, failas);  }
}
//                Duomenys siunčiami į ekraną
void Rodyti(FILE *) {
  while(fgets(buf, strlen(buf)+1, failas))
    printf("%s",buf);
}
```

✓Sudarykite pagalbinę funkciją buferinio kintamojo `buf` papildymui formatuotais laukais.

✓Pakeiskite programoje blokų rašymo komandą `fwrite` eilučių rašymo komanda `fputs`.

✓Pakeiskite programą taip, kad jos darbą būtų galima nutraukti įvedus vieno simbolio eilutę: `"Q"` arba `"q"`. Norint tai padaryti, reikia iš klaviatūros perskaityti iš karto visą eilutę ir tik po to ją skanuoti su funkcija `sscanf`.

2.10 pratimas. Panaudodami duomenų srautų apdorojimo klasę **`ofstream`** įvedame duomenis į matricą ir atlikę veiksmus (sukeičiami vietomis stulpeliai su didžiausia ir mažiausia matricos reikšme) išvedame rezultatus.

```
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <iomanip.h>
```

```
const num = 15;
typedef int matr[num][num];

void read(char *, matr, int *);          // Duomenų skaitymas
void print(char *, matr, int, char *);   // Spausdinimas
int max(matr, int);                      // Didžiausio paieška
int min(matr, int);                      // Mažiausio paieška
void swap(matr, int, int, int);          // Stulpelių sukeitimas
int test(char *, char *);                // Failų paruošimas
//                  Pagrindinė programa
 void main(void) {
  int n;
  matr a;
  char inbuff[12], outbuff[12];          // Failų vardams saugoti
                        // Rodyklės į failų vardus *duom ir *rezult
  cout << "Koks duomenu failo vardas?\n";
  char *duom = gets(inbuff);
  cout << "Koks rezultatu failo vardas?\n";
  char *rezult = gets(outbuff);
  if (test(duom, rezult)) {
    cout << "Klaida atidarant failus";  exit(0); }
  read(duom, a, &n);                 // Duomenų skaitymas į matricą
                                     // Duomenų spausdinimas į failą
  print(rezult, a, n, "Pradiniai duomenys");
  swap(a, n, min(a, n), max(a, n));
  print(rezult, a, n, "Rezultatai"); // Rezultatų spausdinimas
  cout << "Pabaiga\n\a";
}
//                  Duomenų skaitymas
 void read(char *is, matr a, int *n) {
  ifstream infile(is);               // Duomenų failo atidarymas
  infile >> *n;                      // Įvedama n reikšmė
  (*n)--;                            // Matricos įvedimas
  for(int i=0; i<=*n; i++)
    for(int j=0; j<=*n; j++)
      infile >> a[i][j];
  infile.close();                    // Failo uždarymas
}
//                  Spausdinimas
 void print(char *os, matr a, int n, char *text) {
                              // Rezultatų failas paruošiamas papildymui
  ofstream offile(os, ios::app);
```

```
  offile << '\n' << text << '\n' << "     ";
  for(int j=1; j<=n+1; j++)  offile << setw(5) << j;
  offile << " \n    |";
  for(j=0; j<=n; j++)  offile << "-----";
  offile << " \n";
  for(int i=0; i<=n; i++) {
    offile << setw(3) << (i+1) << " |";
    for(int j=0; j<=n; j++)
      offile << setw(5) << a[i][j];
    offile << " \n";  }
}
```

Randamas stulpelis su didžiausia reikšme

```
int max (matr a, int n) {
  int m = -1000, p = -1;          // p - stulpelio numeris
  for(int i=0; i<=n; i++)
    for(int j=0; j<=n; j++)
      if (a[i][j] > m) { m = a[i][j];  p = j; }
  return p;  }
```

Randama mažiausia reikšmė

```
int min(matr a, int n) {
  int m = 1000,  p = -1;          // p - stulpelio numeris
  for(int i=0; i<=n; i++)
    for(int j=0; j<=n; j++)
      if (a[i][j] < m)  { m = a[i][j];  p = j; }
  return p;  }
```

Sukeičiami du stulpeliai vietomis

```
void swap(matr a, int n, int mine, int maxe) {
  int p;
  for(int i=0; i<=n; i++) {
    p = a[i][mine];
    a[i][mine] = a[i][maxe];
    a[i][maxe] = p; }  }
```

Pagrindinė programa

```
 int test(char *is, char *os) {
  ifstream duomenys(is);         // Patikrinamas duomenų failas
  if (!duomenys)  return 1;
  duomenys.close();
  ofstream rezultatai(os);       // Patikrinamas rezultatų failas
  if (!rezultatai)  return 1;
// Į rezultatų failą išvedamas pranešimas
  rezultatai << "Matrica \n";
  rezultatai.close();
  return 0;
```

```
}
```

Duomenų failas

```
4
15  1  5    6
2  25  6    7
5  -5  5  -45
1   2  3    4
```

Rezultatų failas

```
Matrica

Pradiniai duomenys
          1    2    3    4
    |--------------------
  1 |   15    1    5    6
  2 |    2   25    6    7
  3 |    5   -5    5  -45
  4 |    1    2    3    4

Rezultatai
          1    2    3    4
    |--------------------
  1 |   15    6    5    1
  2 |    2    7    6   25
  3 |    5  -45    5   -5
  4 |    1    4    3    2
```

2.6. Simbolių eilutės

Simbolių masyvai gali būti vartojami simbolių eilutėms apdoroti. Naudojant masyvus simboliams saugoti, reikia atsiminti, kad C^{++} kalboje eilučių reikšmių pabaigas priimta žymėti nulinio ASCII kodo simboliu, kuris žymimas `'\0'`**.** Simbolinio tipo konstantos taip pat yra rašomos tarp apostrofų, pavyzdžiui `'a'`, `'D'`. Eilutės tipo konstantos rašomos tarp kabučių, pavyzdžiui, `"Katinas"`.

2.11 pratimas. Klaviatūra įvedamas žodis (simbolių eilutė be tarpų). Programa įvestoje simbolių eilutėje visas mažąsias raides keičia didžiosiomis.

```
#include <iostream.h>
#include <conio.h>
void main() {
  char Eil[80];
  int  i, c;
  c = 'a' - 'A';              // Atstumas tarp raidžių kodų lentelėje
  cout << "Įveskite eilutę mažosiomis raidėmis ir be
           tarpų\n";
  cin >> Eil;
  for(i=0; Eil[i] != '\0'; i++)
```

```
    if (Eil[i] >= 'a')  Eil[i] -= c;
  cout << Eil << endl;  }
```

> ✓ Sudarykite programą, kuri skaičiuotų ir parodytų ekrane visų klaviatūra įvestos eilutės lotyniškų raidžių pasikartojimo dažnius. Didžiosios ir mažosios raidės turi būti interpretuojamos vienodai. Neužmirškite, kad C kalboje `char` ir `int` tipai yra suderinami

Prieš nagrinėdami tolesnius pavyzdžius, apžvelgsime kelias funkcijas, palengvinančias darbą su eilutėmis.

`char *strcpy(char *dest, const char *src);`

Eilutę `src` kopijuoja į `dest` iki nulinio simbolio. Grąžina rodyklę į `dest`.

`char *stpcpy(char *dest, const char *src);`

Eilutę `src` kopijuoja į eilutę `dest` iki nulinio simbolio. Grąžina `dest + strlen(src)`.

`char *strcat(char *dest, const char *src);`

Eilutės `src` kopiją prijungia prie eilutės `dest` galo. Grąžina rodyklę į sujungtas eilutes (`dest`). Grąžinamos eilutės ilgis yra: `strlen(dest) + strlen(src)`.

`const char *strchr(const char *s, int c);`
`char *strchr( char *s, int c);`

Eilutėje **`s`** ieško simbolio **`c`**. Paieška pradedama nuo eilutės pradžios ir baigiama suradus pirmąjį simbolį, lygų nurodytam. Nulinis simbolis laikomas eilutės dalimi, todėl jo paieška taip pat galima. Pavyzdžiui, `strchr(strs,0)` grąžina rodyklę į nulinį simbolį. Funkcija grąžina rodyklę į surastą simbolį eilutėje, arba `NULL` neradus.

2.12 pratimas. Programa iliustruoja funkcijos **`strchr`** veikimą.

```
#include <string.h>
#include <stdio.h>

int main(void) {
  char string[20];
  char *ptr, c = 'r';
  strcpy(string, "Nagrinelama eilute");
  ptr = strchr(string, c);
  if (ptr)  printf("Simbolis %c yra:
                              %d-as\n", c, ptr-string);
  else  printf("Tokio simbolio eiluteje nera\n");
```

```
  return 0;
}
```

Įvykdę programą, ekrane matysime eilutę:

```
Simbolis r yra: 3-as
```

✓ Pertvarkykite programą taip, kad ji išvardytų ekrane visas dialogo su vartotoju metu nurodytos raidės pozicijas klaviatūra įvestoje eilutėje.

`int strcmp(const char *s1, const char *s2);`

Palygina eilutes. Lygina eilutes, pradedant pirmaisiais simboliais iki tol, kol bus surasti nesutampantys simboliai arba bus surasta vienos iš eilučių pabaiga.

Jeigu `s1` yra...	`strcmp` grąžina reikšmę, kuri yra...
mažesnė už eilutę `s2`	`< 0`
lygi eilutei `s2`	`== 0`
didesnė už eilutę `s2`	`> 0`

`int strcmpi(const char *s1, const char *s2);`

Palygina eilutes, kaip ir ankstesnė, tik mažąsias ir didžiąsias raides laiko vienodomis.

`size_t strlen(const char *s);`

Grąžina simbolių skaičių eilutėje. Nulinis simbolis neskaičiuojamas.

`char *strncpy(char *dest,`
`const char *src, size_t maxlen);`

Kopijuoja iš eilutės `src` į eilutę `dest` nurodytą simbolių skaičių `maxlen`. Jeigu eilutėje `src` simbolių mažiau, negu nurodytas kiekis `maxlen`, tuomet rezultate bus tiek, kiek surasta; jeigu daugiau negu reikia, tai tiek kiek nurodyta. Grąžinama rodyklė į `dest` eilutę.

2.13 pratimas. Programa iliustruoja funkcijos **`strncpy`** veikimą.

```
#include <stdio.h>
#include <string.h>

int main(void) {
  char string[10];
  char *str1 = "abcdefghi";
  strncpy(string, str1, 3);
  string[3] = '\0';
  printf("%s\n", string);
```

```
  return 0;
}
```

Įvykdę programą, ekrane matysime eilutę:

```
abc
```

`char *strtok(char *``s1`**`, const char *`**`s2`**`);`**

Funkcija grąžina rodyklę į `s1` fragmentą (žodį) iš simbolių, kurių nėra eilutėje `s2`, o už fragmento įterpia nulinio kodo simbolį. Kartojant kreipinį su `NULL` vietoje pirmo argumento, grąžinama rodyklė į kitą žodį. Kai naujų žodžių nebėra, grąžinama rodyklė `NULL`.

2.14 pratimas. Programa iliustruoja funkcijos **`strtok`** veikimą.

```
#include <string.h>
#include <iostream.h>
#include <conio.h>

void main() {
  char Eil[30] = "Joju, dairausi ir dainuoju";
  char *p, sep[5]= ", ";
  cout << Eil << endl;              // Visa tiriama eilutė
                    // strtok įterpia ribotuvą NULL už surasto žodžio
   p = strtok(Eil, sep);
   if (p)  cout << p << endl;       // Pirmasis žodis
        // Antras kreipinys su argumentu NULL grąžina antro žodžio adresą
   p = strtok(NULL, sep);
   if (p)  cout << p << endl;       // Antrasis žodis
   cout << Eil << endl;             // Tik pirmasis žodis
}
```

Įvykdę programą, ekrane matysime eilutę:

```
Joju, dairausi ir dainuoju
Joju
dairausi
Joju
```

✓ Pertvarkykite pavyzdžio programą su ciklo operatoriumi taip, kad ji parodytų ekrane visus tiriamos eilutės žodžius. Analizės ciklo pabaigos požymiu vartokite grąžinamos rodyklės reikšmę `NULL`. Atskiriamus iš

eilutės žodžius iš pradžių surašykite į žodžių masyvą ir tik po to parodykite ekrane.

`char *_strtime(char *``buf`**`);`**

Kompiuterio laiką užrašo eilutėje `buf`, kurios ilgis privalo būti `9` Eilutės forma `HH:MM:SS`, kur `HH` –valandos, `MM` – minutės ir `SS` – sekundės. Grąžina eilutės `buf` adresą. Eilutė baigiama nuliniu simboliu.

`char *_strdate(char *``buf`**`);`**

Kompiuterio datą užrašo eilutėje `buf`, kurios ilgis privalo būti `9` Eilutės forma `MM/DD/YY`, kur `MM` – mėnuo, `DD` – mėnesio diena ir `YY` – metų paskutiniai du skaičiai. Grąžina eilutės `buf` adresą. Eilutė baigiama nuliniu simboliu.

2.15 pratimas. Programa iliustruoja funkcijų **`_strdate`** ir **`_strtime`** veikimą.

```
#include <time.h>
#include <stdio.h>

void main(void) {
  char data[9];
  char laikas[9];
  _strdate(data);
  _strtime(laikas);
  printf("Data: %s  Laikas: %s\n", data, laikas);
}
```

Įvykdę programą, ekrane matysime vykdymo dienos datą ir laiką, pavyzdžiui:

```
Data: 10/22/00  Laikas: 11:13:16
```

2.16 pratimas. Klaviatūra įvedama simbolių eilutė, kur žodžiai skiriami bent vienu tarpu. Reikia atspausdinti eilutės žodžius po vieną ekrane, nurodant žodžio pradžios ir pabaigos indeksus. Ankstesniame pratime eilutės įvedimo pabaiga buvo pirmas sutiktas tarpas arba eilutės pabaiga, todėl parašius kelis žodžius, buvo įvedamas tik pirmasis. Visos eilutės įvedimui galima naudoti funkciją:

```
#include <stdio.h>
char *gets(char *s);
```

Skaitomos eilutės pabaigos simbolis '\n' automatiškai pakeičiamas simboliu '\0'. Sudarant programą, tikslinga visą eilutę perskaityti į jai skiriamą simbolių masyvą ir po to analizuoti po vieną simbolį.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>

typedef char Mas[80];
void zodis(Mas, Mas, int *);          // Žodžio atskyrimas
//                  Pagrindinė programa
void main() {
  Mas Eil, Z;
  int i, c;
  cout << "Įveskite eilutę mažosiomis raidėmis\n";
  cout << " Žodžius atskirkite tarpais\n";
  gets(Eil);                  // Simbolių eilutė
  cout << Eil << endl;        // Eilutė spausdinama ekrane
  strcat(Eil, " "); // Po paskutinio žodžio bus tarpas
  i = 0;
  while(Eil[i] != '\0') {     // Žodžių atskyrimas:
    c = i;                    //    žodžio pradžia;
    zodis(Eil, Z, &i);        // surastas žodis pradedant i vieta;
    if (Z[0] != '\0')         // jeigu buvo žodis, tai spausdinamas
      cout << "Pradžia: " << c << " Pabaiga: " << (i-1)
           << " *" << Z << "*" << endl;
    i++;  }                 //    Praleidžiamas tarpas
}
//                  Žodžio atskyrimas
/* Iš eilutės A, pradedant simboliu, numeris n, išskiriamas žodis.
   Surasto žodžio simboliai surašomi į eilutę B. Gale nulinis simbolis */
void zodis(Mas A, Mas B, int *n) {
  int i = 0;
  while((A[*n] != '\0') && (A[*n] != ' '))
    B[i++] = A[(*n)++];
  B[i] = '\0';
}
```

✓ Išbandykite skaitymo po vieną simbolį funkciją:

```
int getc(FILE *stream);
```

Skaitant iš klaviatūros, turi būti vartojamas failo vardas `stdin`**.** Eilutės skaitymo į buferį aprašymo pavyzdys:

```
// Tuščias ciklas!
while((Eil[i++] = getc(stdin)) != '\n');
str[--i]='\0';                    // Eilutės pabaigos žymė
```

✓ Pertvarkykite žodžių atskyrimo programą taip, kad ji ekrane parodytų tik ilgiausią žodį. Žodžio ilgio skaičiavimui siūloma vartoti tokią funkciją:

```
int length(char* x) {
  int i = 0;
  while(x[i++]);
  return   --i;  }
```

Atskiro simbolių eilučių tipo C++ kalboje nėra. Eilutės yra saugomos simbolių masyvuose, kur jų pabaigos yra žymimos nulinio kodo simboliais `'\0'`. Daugumoje C++ kalbos realizacijų yra tokių masyvų apdorojimo bibliotekos, kuriose yra įvairios eilučių struktūros analizės ir jų tvarkymo funkcijos. Borland C++ realizacijoje šios priemonės yra paskirstytos dviejose bibliotekose: `string.h` (struktūros analizė ir tvarkymas) ir `stdlib.h` (tipų keitimas).

2.17 pratimas. Savarankiškai išsiaiškinkite pavyzdžio programėlėje vartojamų eilučių tvarkymo funkcijų paskirtis ir kreipimosi į jas būdus.

```
#include <iostream.h>
#include <string.h>
#include <conio.h>

typedef char zodis[15];
typedef char string[80];

int main() {
  zodis x;
  string z, t;
  cout << "Iveskite savo varda ir pavarde\n";
  cin >> z >> x;
  strcat(z, " ");                  // Eilučių sujungimas
  strcat(z, x);
  strcpy(t, z);
  strlwr(t);                       // Mažosios raidės
  cout << "Mazosiomis raidemis: " << t << endl;
  strupr(t);                       // Didžiosios raidės
```

```
  cout << "Didziosiomis raidemis: " << t << endl;
  cout << "Buvo ivesta: " << z << endl;
  cout << "Buvo raidziu: " << (strlen(z)-1) << endl;
}
```

✓ Pertvarkykite programą taip, kad ji iš klaviatūra įvestos eilutės atskirtų žodžius ir paskirstytų į du masyvus: skaičių ir kitokių žodžių. Abiejų masyvų elementai ir skaičių masyvo suma turi būti parodomi ekrane. Eilučių pertvarkymui į skaičius vartokite bibliotekos `stdlib.h` funkcijas, kurių prototipai:

```
double atof(const char *s);
int atoi(const char *s);
```

Jos grąžina skaičių reikšmes arba `0`, jeigu argumento eilutės negalima pertvarkyti į skaičių.

3 skyrius. Struktūros

Struktūrų aprašų sintaksė:

```
struct <struktūrinio tipo vardas>
       { <laukų aprašų sąrašas> }
                        [<kintamųjų sąrašas>];
```

Laukų aprašų sąrašų elementai atskiriami kabliataškiais, o kintamųjų sąrašo elementai – kableliais. Jei apraše nėra kintamųjų sąrašo, jis apibrėžia tik naujų struktūrinių duomenų tipą, tačiau jo realizacijoms atmintyje vietos neskiria. Atskiro struktūros tipo apibrėžimo ir realizavimo pavyzdys:

```
// Apibrėžimas
struct telefonas { char     vardas[30];
                   unsigned longint  tel;  }
// Realizavimas
struct telefonas Tel, *P, TelMas[100];
```

Kreipiantis į struktūrų elementus vartojami sudėtiniai vardai:

```
<struktūros vardas>.<lauko vardas>
```

Galimos rodyklės į sruktūrų tipo reikšmes:

```
P = &Tel;              // struct telefonas *P;
```

Laukų vardų užrašymo, panaudojant rodykles, pavyzdžiai:

```
(*P)->Tel, P->Tel, P->Tel.vardas
```

3.1 pratimas. Klaviatūra įvedamas žmonių sąrašas spausdinamas lentele faile.

```
#include <stdio.h>
#include <stdlib.h>
#include <iomanip.h>
#include <conio.h>

struct zmogus { char  vardas [10];
                char  pavarde[10];
                int   gim_met;
                float ugis;     };
```

```
FILE *F;
const char Duomenys[]="Prat31.rez";

int Atidaro(const char *, char *);
void Lentele();
//                    Pagrindinė programa
void main(void ) {
  if (Atidaro(Duomenys, "w")) exit(0);
  Lentele();
  fclose(F);
  cout << "Pabaiga\n\a";
}
//                        Failo atidarymas
int Atidaro( const char *Vardas, char *Forma){
  if (( F = fopen( Vardas, Forma )) == NULL )
   { cout<< "Failas neatidarytas"; return 1;}
  else return 0;  }
//        Duomenys įvedami klaviatūra ir spausdinami faile
void Lentele() {
  struct zmogus A;
  int n;
  cout << "Kiek zmoniu bus ?\n";
  cin >> n;
  fprintf(F,"....................................\n");
  fprintf(F,":   Vardas  :  Pavarde  : gim. m. : ugis :\n");
  fprintf(F,"....................................\n");
  while(n--) {
    cout << "Iveskite varda        \n";  cin >> A.vardas ;
    cout << "Iveskite pavarde      \n";  cin >> A.pavarde;
    cout << "Iveskite gimimo metus \n";  cin >> A.gim_met;
    cout << "Iveskite ugi          \n";  cin >> A.ugis;
    fprintf(F, ":%10s :%10s :   %4d  : %3.2f :\n",
               A.vardas, A.pavarde, A.gim_met, A.ugis); }
  fprintf(F,"....................................\n");
  fclose(F);
}
```

Programos darbo rezultatai faile "Prat31.rez":

```
....................................
:   Vardas  :  Pavarde  : gim. m. : ugis :
....................................
:    Petras : Petraitis :   1933  : 2.15 :
:    Jurgis :  Jurgelis :   1582  : 1.58 :
:     Kazys :   Kazelis :   1258  : 1.25 :
....................................
```

Struktūrų masyvai sudaromi taip pat, kaip ir paprastų duomenų tipų. Rekomenduojama sukurti duomenų tipą, po to jį naudoti.

3.2 pratimas. Klaviatūra įvedami duomenys surašomi struktūrų masyve. Po to jie spausdinami lentele.

```
#include <stdio.h>
#include <stdlib.h>
#include <iomanip.h>
#include <conio.h>

struct zmogus { char   vardas [10];
                char   pavarde[10];
                int    gim_met;
                float  ugis;  };
FILE *F;
const char Duomenys[]="Prat32.rez";

int Atidaro(const char *, char *);
void Ivesti(zmogus *, int *);
void Lentele(zmogus *, int);
//                  Pagrindinė programa
void main(void ) {
  zmogus A[10];
  int n = 0;
  Ivesti(A, &n);
  if (Atidaro(Duomenys, "w")) exit(0);
  Lentele(A, n);
  fclose(F);
  cout<<"Pabaiga\n\a";
}
//                  Įvedimas klaviatūra
void Ivesti(zmogus *B, int *n) {
  int i = 0;
  cout << "Kiek zmoniu bus ?\n";
  cin >> *n;
  while(i < *n) {
    cout << "**********************\n";
    cout << (i+1) << "-ojo zmogaus duomenys:\n";
    cout << "Iveskite varda     \n";  cin >> B[i].vardas;
    cout << "Iveskite pavarde   \n";  cin >> B[i].pavarde;
    cout << "Iveskite gim. metus\n";  cin >> B[i].gim_met;
    cout << "Iveskite ugi       \n";  cin >> B[i].ugis;
    cout << "Aciu \n";        i++; }
}
```

```
//                    Failo atidarymas
int Atidaro(const char *Vardas, char *Forma) {
  if ((F = fopen(Vardas, Forma)) == NULL )
    { cout << "Failas neatidarytas";  return 1; }
  else return 0;
}
//                    Išvedimas lentele
void Lentele(zmogus *C, int n) {
  int i = 0;
  fprintf(F,"....................................\n");
  fprintf(F,":   Vardas  :  Pavarde  : gim. m. : ugis :\n");
  fprintf(F,"....................................\n");
  while(i < n) {
    fprintf(F, ":%10s :%10s :   %4d  : %3.2f :\n",
      C[i].vardas, C[i].pavarde, C[i].gim_met, C[i].ugis);
    i++; }
  fprintf(F,"....................................\n");
  fclose(F);
}
```

3.3 pratimas. Klaviatūra įvedami duomenys surašomi į tipizuotą failą. Po to jie skaitomi iš to failo ir parodomi ekrane.

```
#include <stdio.h>
#include <stdlib.h>
#include <iomanip.h>
#include <conio.h>

FILE *failas;
const char Rezultatai[]="Prat33.rez";
struct telefonas { char vardas [10];
                   char pavarde[10];
                   long tel;         } T;

int Atidaro ( const char *, char *);
void Raso();
void Skaito();
//                    Pagrindinė programa
void main(void ) {
  if (Atidaro(Rezultatai, "w")) exit(0);
  Raso();
  fclose(failas);
  if (Atidaro(Rezultatai, "r")) exit (0);
  Skaito();
```

```
  fclose(failas);
  cout << "Pabaiga\n\a";
}
//                Įvedamus duomenis surašo į failą
void Raso() {
  char buf[50] = "";
  printf("Programos nutraukimas - eilute 'q'\n");
  while(buf[0] != 'q') {
    printf("Nurodykite varda, pavarde ir telefono numeri:\n");
    gets(buf);
    if (buf[0] != 'q') {
      sscanf(buf, "%s%s%ld", T.vardas, T.pavarde, &T.tel);
      fwrite(&T, sizeof(struct telefonas), 1, failas); } }
}
//                      Failo atidarymas
int Atidaro(const char *Vardas, char *Forma) {
  if ((failas = fopen(Vardas, Forma)) == NULL)
    { cout << "Failas neatidarytas"; return 1; }
  else return 0;
}
//                      Skaito iš failo
void Skaito() {
  printf(".....................................\n");
  printf(":   Vardas   :  Pavarde  : telefonas  :\n");
  printf(".....................................\n");
  while(fread(&T, sizeof(struct telefonas), 1, failas))
    printf( ":%10s  :%10s : %10ld :\n",
                          T.vardas, T.pavarde, T.tel);
  printf(".....................................\n");
}
```

✓ Pertvarkykite programą, kad pagrindinėje funkcijoje pagal vartotojo pageidavimus būtų galima pasirinkti tokius veiksmus: duomenų failo turinio išvedimas ekrane, naujo failo formavimas arba esančio failo papildymas. Programos šakojimui vartokite operatorių `case`.

✓ Papildykite programą failo rikiavimo pagal pavardes alfabeto tvarka funkcija. Rikiuojamo failo duomenys iš pradžių turi būti perrašomi į įrašų masyvą ir tik po to rikiuojami. Lyginant eilutes turi būti vartojamos ne santykio operacijos, bet bibliotekos `string.h` funkcijos arba makrokomandos: `stricmp`, `strnicmp`, `strncmpi`, `strcmpi`.

Kuriant sudėtingesnes programas tikslinga žinoti keletą įdomesnių funkcijų savybių. Kalboje C++ realizuota labai efektyvi funkcijų savybė – polimorfizmas, kuri dar vadinama daugiavariantiškumu arba funkcijų persidengimu (overloading). Senose C++ kalbos versijose polimorfinių funkcijų aprašai turi būti pažymimi baziniu žodžiu `overload`. Kalbos realizacijose Borland C++ tai daryti nebūtina.

3.4 pratimas. Demonstruojama išvedimo ekrane polimorfinė funkcija `print`, kurios darbas priklauso nuo kreipinyje užrašyto duomenų tipo.

```
#include <stdio.h>
#include <conio.h>

// overload print;            // Bordland C galima praleisti.
inline void print(int    x) {printf("%d\n"   , x); }
inline void print(double x) {printf("%5.2f\n", x); }
inline void print(char  *x) {printf("%s\n",    x); }

void main(void ) {
  print(3.14);                // Slankaus kablelio skaičius
  print(15);                  // Sveikas skaičius
  print("Simboliu eilute");   // Simbolių seka
}
```

Čia baziniu žodžiu `inline` pažymėtos įterpiamos funkcijos. Įterpiamos funkcijos yra makrokomandų analogai – pirminis procesorius įrašo funkcijos tekstą kiekvieno kreipinio į ją vietoje. Jei funkcijų tekstai trumpi, toks funkcijų tekstų įterpimas padidina programų darbo greitį.

3.5 pratimas. Dar viena labai naudinga funkcijų savybė – argumentų parinkimas pagal nutylėjimą.

```
#include <stdio.h>
#include <conio.h>

struct upe { char *vardas;
             char *salis; }  x, y, z, k;
//                    Funkcija Rodo
void Rodo(struct upe *d, char *a = NULL, char *b = NULL) {
  d->vardas = a;
  d->salis = b; }
//                  Pagrindinė programa
void main(void) {
  Rodo(&x);
```

```
  Rodo(&y, "Nemunas");
  Rodo(&z, "Merkys", "Lietuva");
  Rodo(&k, "Nilas");
  printf("%10s  %10s \n", x.vardas, x.salis);
  printf("%10s  %10s \n", y.vardas, y.salis);
  printf("%10s  %10s \n", z.vardas, z.salis);
  printf("%10s  %10s \n", k.vardas, k.salis);
}
```

Įvykdžius programą ekrane matysime:

```
   (null)      (null)
Nemunas        (null)
 Merkys      Lietuva
  Nilas        (null)
```

4 skyrius. Klasės apibrėžimas. Objektas

C++ kalboje struktūra, jungianti savyje kintamuosius, skirtus duomenims saugoti, ir funkcijas, kurios naudoja tik tuos kintamuosius, vadinama **klase**.

Kintamieji vadinami **duomenimis**, o funkcijos – **metodais**. Objektiniame programavime priimta, kad duomenimis tiesiogiai gali naudotis tik tos klasės metodai.

Klasės aprašo struktūra:

```
class <Klasės Vardas> { <Duomenų elementai>
  public: <Metodai>
                        };
```

Klasėje duomenys ir metodai gali būti rašomi bet kokia seka. Klasės elementai (duomenys ir metodai) gali turėti požymius. Požymis klasėje galioja tol, kol bus sutiktas kito požymio užrašas. Jeigu požymio užrašo nėra, tuomet pagal nutylėjimą bus priimtas `private` visiems elementams iki pirmojo požymio užrašo, jeigu jis bus. Yra tokie požymiai:

- **`private`** (lokalusis). Elementai prieinami tik klasės viduje.
- **`public`** (globalusis). Klasės elementai prieinami jos išorėje.
- **`protected`** (apsaugotasis). Klasės elementai prienami klasėje, kuri paveldi duotąją klasę. Čia jie galioja `private` teisėmis.

Programavimo technologijos požiūriu reikėtų laikytis tam tikros tvarkos. Rekomenduojama pradžioje surašyti duomenis, po to metodus. Metodų sąraše taip pat reikalinga tvarka. Pirmuoju sąraše turėtų būti klasės **konstruktorius**. Tai metodas, skirtas klasės objekto pradinių duomenų reikšmėms nurodyti. Jo vardas privalo sutapti su klasės pavadinimu. Konstruktorius neturi grąžinamos reikšmės. Toliau metodų sąraše turi būti darbo su duomenimis metodai. Gale rašomas **destruktorius**, jeigu toks yra. Tai metodas, naikinantis objektą. Destruktoriaus vardas turi sutapti su klasės vardu, kurio pradžioje parašomas simbolis ‘~’. Pavyzdžiui:

```
class Katinas { …
  public
    Katinas(…) { … }   // Konstruktorius
    …
   ~Katinas()  { … }   // Destruktorius
```

Konstruktorius ir destruktorius privalo būti `public`.

Konstruktorius, kuris neturi parametrų, iškviečiamas darbui pagal nutylėjimą. Galima parašyti:

```
Katinas A;              // vietoje Katinas A();
```

Destruktorius skirtas tos klasės objektui naikinti kompiuterio atmintyje. Jeigu jo nėra, objektas naikinamas baigus vykdyti programą. Programos darbo eigoje kreipinys į destruktorių naikina objektą. Jeigu objekto duomenų laukai gauna atmintį dinamiškai, tuomet būtina destruktoriuje parašyti veiksmus, kuriais atsisakoma atminties, skirtos duomenų laukams. Destruktoriai, kaip ir konstruktoriai, negrąžina jokios reikšmės. Jeigu klasė turi destruktorių, bet nėra kreipinio į jį, tuomet, pagrindinei funkcijai baigus darbą, jis yra vykdomas pagal nutylėjimą.

Metodai klasėje gali būti pilnai aprašyti. Tokius metodų aprašus tikslinga turėti, jeigu jų tekstas yra trumpas. Kitų metodų aprašai iškeliami už klasės ribų. Tuomet klasėje rašomas tik metodo prototipas. Metodo aprašo klasės išorėje struktūra:

```
<Grąžinamos reikšmės tipas> <Klasės Vardas>::
                <Metodo vardas> (<Parametrų sąrašas>)
  { <Programos tekstas> }
```

Klasės tipo kintamieji vadinami **objektais**. Jų aprašymas analogiškas kitų tipų kintamųjų aprašams. Sukuriami objektai gali būti statiniai, dinaminiai. Galima turėti objektų masyvus. Objektai gali būti dinaminių sąrašų elementais.

4.1 pratimas. Sukuriama klasė, aprašanti vieno studento savybę – svorį. Klasėje yra du metodai: duomenų skaitymo klaviatūra ir duomenų rodymo ekrane. Pagrindinėje funkcijoje sukuriami du objektai. Duomenys įvedami panaudojant metodą **`Skaito`**, o parodomi ekrane panaudojant metodą **`Rodo`**. Metodai klasėje aprašomi kaip `public`, nes juos naudojame klasės išorėje.

```
#include <iostream.h>          // Įvedimo/išvedimo priemonės
#include <conio.h>             // Ryšio su ekranu priemonės
```

```
//                    Klasės aprašymas
class Studentas {
  char  *pav;
  float svoris;
public:
    void Skaito() {             // Metodas
      cout << "Iveskite pavarde: \n";
      cin >> pav;
      cout << "Iveskite svori:\n";
      cin >> svoris;  };
    void Rodo() {               // Metodas
      cout << pav << "  " << svoris << endl; };
};
//                    Pagrindinė programa
void main() {
  Studentas A, B;               // Sukuriami objektai
  A.Skaito();
  B.Skaito();
  B.Rodo();
  A.Rodo();
}
```

4.2 pratimas. Sukuriama klasė **`studentas`**, kuri turi du duomenų laukus, skirtus saugoti pavardei ir svoriui. Konstruktorius **`studentas`** suteikia pradines reikšmes duomenų laukams. Reikšmės nurodomos objekto sukūrimo metu. Konstruktoriaus ir metodo **`Rodo`** aprašai iškelti už klasės ribų. Metodas **`Rodo`** išveda ekrane studento pavardę ir svorį. Įterpiami metodai **`RodoVarda`** ir **`RodoSvori`** skirti gauti informaciją apie duomenų laukuose saugomą pavardę ir svorį. Metodas **`Duomenys`** skirtas naujų duomenų perdavimui į objektą. Vidinių metodų informacijai apdoroti klasėje nėra. Klasės objektai skiriami tik duomenų registracijai.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <conio.h>
//                    Klasės aprašymas
class student {
  char *vardas;
  float svoris;
public:
```

```
    student(char *, float);                 // Konstruktorius
    void  Rodo();                           // Metodas
    char *RodoVarda() { return vardas; }    // Metodas
    float RodoSvori() { return svoris; }    // Metodas
    void  Duomenys(char *a, float b)        // Metodas
      { vardas = a;   svoris = b;  }
};
//                    Pagrindinė programa
void main() {
  clrscr();
  student X("Petraitis", 13.5);  // Objektas X
  student Y(" ", 0);             // Objektas Y

  X.Rodo();                      // Objekto X duomenys
  cout << "Objekto student X duomenys \n";
  cout << X.RodoVarda() << " : "<< X.RodoSvori();
  cout << endl;
                  // Duomenų persiuntimas kitam objektui, t.y. Y = X
  Y.Duomenys(X.RodoVarda(), X.RodoSvori());
  Y.Rodo();                      // Objekto Y duomenys
}
//                    Metodų aprašai
student::student(char *a, float b)
  { vardas = a;  svoris = b;  }
void student::Rodo()
  { cout << vardas << " : " << svoris << endl; }
```

Įvykdžius programą ekrane matysime:

```
Petraitis : 13.5
Objekto student X duomenys
Petraitis : 13.5
Petraitis : 13.5
```

4.3 pratimas. Sukuriama klasė darbui su skaičiais masyve. Numatyti veiksmai: papildyti sąrašą nauja reikšme, pašalinti nurodytą reikšmę ir peržiūrėti ekrane turimus sąrašo skaičius. Klasę nagrinėti ir panaudoti paprasčiau, kai metodų aprašai pateikiami atskirai. Metodų aprašus siūloma rašyti tuoj po klasių (arba kiekvienos klasės) aprašų.

```
#include <iostream.h>
#include <conio.h>
const dydis = 50;
```

```
//                          Klasės aprašymas
class sar {
  int A[dydis];
  int ilg;
public:
    void sar();                  // Konstruktorius
    void add(int);               // Sąrašo papildymo metodas
    void del(int);               // Reikšmės šalinimo metodas
    void get();                  // Sąrašo rodymo ekrane metodas
};
//                          Metodų aprašai
void sar::sar() { ilg = 0; }     // Konstruktorius
void sar::add(int naujas) {      // Papildymo metodas
  if (ilg == dydis) {     // Sąrašas papildomas, jeigu sąraše dar nebuvo
    cout << "sarasas pilnas";
    return; }
  for(int i=0; i<ilg; i++)
    if (A[i] == naujas) return;
  A[ilg++] = naujas;  }
void sar::del(int elem) {        // Šalinimo metodas
  for(int i=0; i<ilg; i++)
    if (A[i] == elem) {
      for(int j=i; j<ilg-1; j++)  A[j] = A[j+1];
      ilg--;  }  }
void sar::get() {                // Išvedimo ekrane metodas
  for(int i=0; i<ilg; i++)  cout << A[i] << " ";
  cout << endl;
}
//                          Pagrindinė programa
void main() {
  clrscr();
  sar Rasa;                      // Objekto aprašas
  Rasa.sar();                    // Objektas paruošiamas
  Rasa.add(10);                  // Sąrašo papildymai
  Rasa.add(20);
  Rasa.add(30);
  Rasa.get();                    // Ekrane matome: 10  20  30
  Rasa.del(20);                  // Šaliname
  Rasa.get();                    // Ekrane matome: 10  30
}
```

5 skyrius. Įvedimo, išvedimo srautai

5.1. Išvedimas ekrane ir įvedimas klaviatūra

Įvedimo srautų klasė `istream` ir išvedimo srautų klasė `ostream` aprašytos faile `iostream.h`. Jų objektai `cin` ir `cout` palaiko klasėse numatytus veiksmus.

Anksčiau buvo nagrinėjami operatoriai >> ir <<.

Išvedamų skaičių formatui valdyti buvo panaudotos faile `iomanip`.h esančios priemonės `setw(int)` ir `setprecission(int)`.

Klasės `ostream` metodas `width(int)` skirtas nurodyti išvedamo sraute duomenų laukams. Galioja artimiausiam dydžiui. Tai `setw` analogas.

5.1 pratimas. Parodomas duomenų lauko valdymo metodas.

```
#include <iostream.h>
#include <conio.h>

main() {
  for(int i=2; i<6; i++) {
    cout.width(3);
    cout << "i=";
    cout.width(i);
    cout << i << endl; }
}
```

Įvykdžius programą ekrane matysime:

```
 i= 2
 i=  3
 i=   4
 i=    5
```

Išvedant duomenis nurodytu formatu nepanaudotos pozicijos užpildomos tarpo simboliais. Metodas `fill(int)` leidžia pakeisti užpildymo simbolį norimu. Pakeitimas galioja iki naujo nurodymo.

5.2 pratimas. Demonstruojamas duomenų laukų užpildymas simboliais.

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>

main() {
  cout.fill('.');
  for(int i=2; i<6; i++) {
    cout << "i=";
    cout.width(i);
    cout << i << endl; }
}
```

Įvykdžius programą ekrane matysime:

```
i=.2
i=..3
i=...4
i=....5
```

5.3 pratimas. Metodas `put` skirtas vienam simboliui išvesti ekrane.

```
#include <iostream.h>
#include <conio.h>

main() {
  char zodis[] = "Katinas";
  for(int i=0; zodis[i]; i++)
    cout.put(zodis[i]);
  cout << endl;
}
```

Įvykdžius programą ekrane matysime:

```
Katinas
```

5.4 pratimas. Metodas `put` programose dažnai vartojamas kartu su funkcija `int toupper(int)`, kuri mažąją raidę keičia didžiąja. Kiti simboliai nekinta.

```
#include <iostream.h>
#include <conio.h>
#include <ctype.h>
```

```
main() {
  char sim;
  char zodis[] = "Katinas ir Pele";
  for(int i=0; zodis[i]; i++) {
    sim = toupper(zodis[i]);
    cout.put(sim); }
  cout << endl << zodis << endl;
}
```

Įvykdžius programą ekrane matysime:

```
KATINAS IR PELE
Katinas ir Pele
```

5.5 pratimas. Įvedimo klasėje metodas `get` skirtas vieno simbolio įvedimui klaviatūra.

```
#include <iostream.h>
#include <conio.h>
#include <ctype.h>

main() {
  char sim;
  cout << "Ar dar dirbsite?( T/N ): ";
  do {
    sim = cin.get();
    sim = toupper(sim); }
  while((sim != 'T') && (sim != 'N'));
  cout << "Jus pasirinkote : " << sim << endl;
}
```

Įvykdžius programą ekrane matysime:

```
Ar dar dirbsite?( T/N ): t
Jus pasirinkote : T
```

5.6 pratimas. Įvedimo klasėje metodas `getline` skirtas vienos eilutės įvedimui klaviatūra. Matome, kad paprastai įvedama eilutė iki pirmojo tarpo simbolio arba iki galo, jeigu tarpo simbolio nebuvo. Panaudoję `getline` metodą galima įvesti norimą simbolių skaičių: nurodomas skaičius. Jeigu tiek simbolių nebuvo, įvedami visi simboliai iki eilutės galo. Įvedama eilutė visuomet papildoma nuliniu simboliu (eilutės galas).

```
#include <iostream.h>
#include <conio.h>
#include <ctype.h>

main() {
  char A[30];
  int n;
  cout << "Iveskite eilute:\n";
  cin >> A;                              //  Pirmasis eilutės žodis
  cout << "Eilute: *" << A << "*\n";
  cin.getline(A, 12);                    //Vienuolika simbolių ir  '\0'
  cout << "Eilute: *" << A << "*\n";
  cin.getline(A, sizeof(A));             //  Iki galo arba tiek, kiek telpa
  cout << "Eilute: *" << A << "*\n";
  n = cin.gcount();                      //  Eilutės ilgis su nuliniu simboliu
  cout << "n= " << n << endl;
}
```

Įvykdžius programą ekrane matysime:

```
Iveskite eilute:
Rainas Katinas ir Pilka Pele
Eilute: *Rainas*
Eilute: * Katinas ir*
Eilute: * Pilka Pele*
n=12
```

5.7 pratimas. Įvedimo klasėje metodas `getline` gali turėti trečią parametrą, nurodantį simbolį, kuriuo baigiamas įvedimas. Pavyzdyje parašyta raidė ‘Z’. Jeigu įvedamoje eilutėje jos nebus, tuomet bus įvedami visi eilutės simboliai arba tik tiek, kiek nurodyta, jeigu eilutėje jų yra daugiau.

```
#include <iostream.h>
#include <conio.h>
#include <ctype.h>

main() {
  char A[30];
  cout << "Iveskite eilute:\n";
  cin.getline(A, sizeof(A), 'Z');
  cout << "Eilute: *" << A << "*\n";
}
```

5.2. Įvedimas iš failo ir išvedimas į failą

Įvedimo srautu iš failo klasė `ifstream` ir išvedimo srautu į failą klasė `ofstream` aprašytos faile `fstream.h`. Galima sukurti objektus, kurie nukreipia įvedimo\išvedimo srautus iš\į failus. Jiems atitinkamai galioja operatoriai >> ir <<. Konstruktoriaus parametru yra failo vardas. Čia yra metodai, kuriuos turi jau nagrinėtos srautų klasės. Failo uždarymui yra metodas **`close().`**

5.8 pratimas. Duomenų faile "`Duom58.txt`" yra skaičius 15. Rezultatų faile "`Rez58.txt`" rasime skaičių 15.

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>

main() {
  ofstream R("Rez58.txt");
  ifstream D("Duom58.txt");
  int A;
  D >> A;
  cout << " A= " << A << endl;
  R << A;
}
```

Įvykdžius programą ekrane matysime:

```
A= 15
```

Rezultatų faile bus:

```
15
```

5.9 pratimas. Duomenų failo "`Duom59.txt`" perrašymas eilutėmis į naują failą "`Rez59.txt`".Taip programiškai gaunama failo kopija. Metodas **`eof`** skirtas failo pabaigai nustatyti: grąžina reikšmę `0`, kai failo pabaiga dar nepasiekta, ir `1`, kai jau turime failo pabaigą.

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>

main() {
  ofstream R("Rez59.txt");
```

```
  ifstream D("Duom59.txt");
  char A[80];
  while(!D.eof()) {
    D.getline(A, sizeof(A));
    cout << A << endl;
    R << A << endl;  }
}
```

Įvykdžius programą duomenų ir rezultatų failuose matysime:

```
Batuotas Katinas
pagavo gauruota peliuka.
Tupi sarka kamine.
```

5.10 pratimas. Duomenų failo "Duom510.txt" perrašymas žodžiais į naują failą "Rez510.txt". Ankstesnėje programoje pakeitus skaitymo iš failo metodą operatoriumi >>, gausime rezultatų faile atskirus duoto teksto žodžius.

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>

main() {
  ofstream R("Rez510.txt");
  ifstream D("Duom510.txt");
  char A[80];
  while(!D.eof()) {
    D >> A;
    cout << A << endl;
    R << A << endl;  }
}
```

Įvykdžius programą duomenų faile, rezultatų faile ir ekrane matysime:

```
Batuotas
Katinas
pagavo
gauruota
peliuka.
Tupi
sarka
kamine.
```

5.11 pratimas. Duomenų failo perrašymas simboliais į naują failą "Rez511.txt" ir ekraną. Ankstesnėje **5.9 pratimo** programoje pakeitus skaitymo iš failo metodą į get, gausime rezultatų faile ir ekrane duomenų failo kopiją.

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>

main() {
  ofstream R("Rez511.txt");
  ifstream D("Duom511.txt");
  char sim;
  while(!D.eof()) {
    sim = D.get();
    cout << sim;
    R << sim;  }
  D.close();
  R.close();
}
```

Dirbant su duomenų failais reikia patikrinti ar failas buvo surastas ir sėkmingai atidarytas. Kiekvienu skaitymo iš failo žingsniu būtina patikrinti, ar veiksmas buvo sėkmingai atliktas. Tam skirtas metodas **fail(),** kurio rezultatas yra 0, kai klaidų nebuvo, ir 1, kuomet įvyko trikis. Klaidų pranešimams ekrane skirtas išvedimo srautu objektas **cerr**. Metodas **fail()** naudojamas išvedime, norint įsitikinti, ar veiksmas buvo sėkmingas (buvo vietos naujam įrašui).

5.12 pratimas. Duomenų failo "Duom512.txt".skaičiai perrašomi į naują failą "Rez512.txt".

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>

main(){
  ofstream R("Rez512.txt");
  ifstream D("Duom512.txt");
  int a;
  if (D.fail()) cerr << "Neatidarytas duomenu failas";
  else {
    while((!D.eof()) && (!D.fail())) {
```

```
      D >> a;
      if(!D.fail()) {
        R << a << endl;
        cout << a << endl; }
    } }
  D.close();
  R.close();
}
```

"Duom.dat"	"Rez13.rez"
15 12 45 -56 3 15 89	15 12 45 -56 3 15 89

5.13 pratimas. Šiose klasėse yra `write` ir `read` metodai, skirti duomenų mainams su failais baitų lygyje. Programa užrašo faile duotą struktūrą, o po to perskaito ir parodo ekrane. Metoduose užrašas (`char *`) nusako kompiliatoriui, kad bus naudojama rodyklė į skirtingus tipus.

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>

struct Studentas{ char  pav[20];
                  int   metai;
                  float ugis; };

main() {
  Studentas A;
  Studentas S = {"Petriukas ", 21, 187.5};
  ofstream R("Rez513.txt");
  R.write((char *) &S, sizeof(Studentas));
  R.close();

  ifstream D("Rez513.txt");
  D.read((char *) &A, sizeof(Studentas));
  cout << A.pav << A.metai << "  " << A.ugis << endl;
  D.close();
}
```

5.14 pratimas. Analogiškai galima sukurti baitinį failą struktūriniams duomenims iš tekstinio failo saugoti. Programa demonstruoja duomenų failo "`Duom514.txt`" perrašymą į failą "`Rez514.txt`" ir po to skaitymą iš jo ir duomenų rašymą ekrane.

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>

struct Studentas{ char  pav[20];
                  int   metai;
                  float ugis; };

main() {
  char sim;
  Studentas A;
  ofstream R("Rez514.txt");
  ifstream D("Duom514.txt");
  while(!D.eof()) {                    // Skaitomi duomenys
    D.getline(A.pav, sizeof(A.pav));
    D >> A.metai >> A.ugis;
    sim = ' ';                         // Eilutės pabaiga
    while((!D.eof()) && (sim != '\n'))
      { sim = D.get();  cout << sim; }
                                       // Duomenys ekrane
    cout << A.pav << "  " << A.metai << "  " << A.ugis;
    cout << endl;
                                       // Duomenys faile
    R.write((char *) &A, sizeof(Studentas));  }
  D.close();
  R.close();
                          // Skaitomi failo duomenys ir parodomi ekrane
  ifstream C("Rez15.txt");
   while(!C.eof()) {
     C.read((char *) &A, sizeof(Studentas));
     if (!C.fail())
       cout << A.pav << A.metai << "  " << A.ugis;
       cout << endl; }
  C.close();
}
```

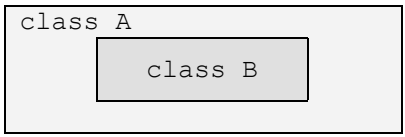
Duomenų failas:

```
Katinas Batuotas      45  12.5
Rapolas Didysis       15  159.98
Barbora Bulvyte       25  199.45
```

6 skyrius. Klasių savybės. Objektiškai orientuotas programavimas

6.1. Paveldėjimas

Objektiniame programavime viena svarbiausių savybių, apibrėžiančių objektų sąveiką, yra paveldėjimas.



Klasė `B`, kurią paveldi klasė `A`, vadinama **protėviu** (bazinė klasė). Klasė `A`, kuri šalia savo duomenų ir metodų paveldi kitą klasę `B`, vadinama **palikuonimi** (išvestinė klasė). Protėvio duomenys ir metodai paveldimi kaip `private`, t.y. juos gali naudoti tik palikuonio metodai. Jeigu norima tiesiogiai kreiptis iš išorės į protėvio metodus, reikia nurodyti paveldėjimo atributą `public`.

6.1 pratimas. Sukuriama klasė **`Langas`**, skirta tekstinio ekraninio lango parametrams saugoti ir langui ekrane formuoti. Sukuriama klasė **`Trys`**, skirta duomenims saugoti, keisti ir demonstruoti ekraniniame lange. Ši klasė paveldi klasę **`Langas`**. Jos metodai naudojami tik savo klasės viduje.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
//                    Klasė Langas
class Langas { int x1, y1, x2, y2;   // Lango koordinatės
               int fonas;            // ir fono spalva
  public:
    Langas(int, int, int, int, int);
    void Ekranas();
};
```

```
//                    Klasės Langas metodai
Langas::Langas(int xv, int yv, int xa, int ya,
                                                int spalva)
{ x1 = xv; y1 = yv; x2 = xa; y2 = ya; fonas = spalva;}

void Langas::Ekranas() {
  window(x1, y1, x2, y2);
  textbackground(fonas);
  clrscr();
}
//                      Klasė Trys
class Trys: Langas { int   a;
                     float b;
                     char  c;
public:
  trys(int, float, char, int, int, int, int, int);
  void add (int   k) { a += k; }      // Didina a
  void addf(float k) { b += k; }      // Didina b
  void addc(char  k) { c += k; }      // Didina c
  void Rodo();                        // Rodo ekrane
};
//                   Klasės Trys metodai
Trys::trys(int r1, float r2, char r3,
           int xv, int yv, int xa, int ya, int spalva):
           Langas(xv, yv, xa, ya, spalva) {
  a = r1;  b = r2;  c = r3;
  Ekranas();
}

void Trys::Rodo() {
  cprintf("Grazu:");
  cprintf("  %5i %5.2f %3c \r\n", a, b, c);
}
//                   Pagrindinė programa
main() {
  int Spalva = 2;
  window(1, 1, 80, 25);
  textbackground(BLACK);
  clrscr();

  Trys A(5, 2.5, 'b', 10, 10, 50, 15, BROWN);

  textcolor(Spalva++);   A.Rodo();   A.add (10  );
  textcolor(Spalva++);   A.Rodo();   A.addf(25.3);
```

```
  textcolor(Spalva++);   A.Rodo();   A.addc(3   );
  textcolor(Spalva++);   A.Rodo();
  getch();
}
```

Įvykdžius programą ekrane rudo fono lange matysime:

```
Grazu:   5     2.50      b
Grazu:  15     2.50      b
Grazu:  15    27.80      b
Grazu:  15    27.80      e
```

Žalia spalva
Žydra spalva
Raudona spalva
Violetine spalva

6.2 pratimas. Sukuriama klasė **`Pirmas`**, kurią paveldi antroji klasė **`Antras`**. Pagrindinėje funkcijoje sukuriami kiekvienos klasės objektai. Demonstruojamas kreipinys į protėvio klasės metodą. Kad galima būtų taip kreiptis, būtina nurodyti paveldėjimo atributą `public`. Konstruktoriuje **`Antras`** kreipinys į konstruktorių **`Pirmas`** rašomas tuoj po antraštės. Jis paveldimam laukui `x` suteikia reikšmę `5`.

```
#include <iostream.h>       // Paveldimo metodo panaudojimas
#include <conio.h>          // pagrindinėje funkcijoje
//                      Klasė Pirmas
class Pirmas { int x;
public:
  Pirmas(int a) { x = a; }
  void Rodo()
    { cprintf("Pirmas::rodo() %5d \n\r", x); }
};
//                      Klasė Antras
class Antras: public Pirmas { int y;
public:
  Antras(int b):Pirmas(5) { y = b; }
  void Rodo(){
  cprintf("Antras::rodo() %5d \n\r", y); }
};
//                    Pagrindinė programa
void main() {
  window(1, 1, 80, 25);
  textbackground(BLACK);
  clrscr();
  window(10, 10, 40, 18);
  textbackground(GREEN);
  clrscr();
```

```
  window(13, 12, 37, 16);
  textcolor( BLACK );

  Pirmas A(10);
  A.Rodo();
  Antras B(25);
  B.Rodo();
  B.Pirmas::Rodo();
  getch();
}
```

Įvykdžius programą ekrane žalios spalvos lange matysime:

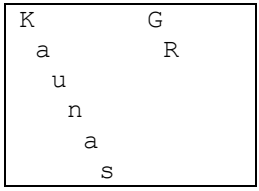
```
Pirmas::rodo()      10
Antras::rodo()      25
Pirmas::rodo()       5
```

6.3 pratimas. Demonstruojamas hierarchinis klasių paveldėjimas. Sukuriamos trys klasės: žymeklio valdymo **`Vieta`**, raidės rašymo ekrane nurodytoje vietoje nurodyta spalva **`Raide`** ir žodžio rašymo ekrane po vieną raidę **`Zodis`**. Klasė **`Vieta`** žymeklio nevaldo. Ji skirta žymeklio koordinatėms saugoti ir keisti.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
//                    Klasė Vieta
class Vieta { int x, y;
public:
  Vieta(int Xs, int Ys) { x = Xs; y = Ys; }
  int KoksX() { return x; }
  int KoksY() { return y; }
  void Kitas(int Xs, int Ys) { x = Xs; y = Ys; }
};
//                    Klasė Raidė
class Raide: Vieta { int spalva;
                     char sim;
public:
  Raide(int Xp, int Yp,int Sp, char Simb):
  Vieta(Xp, Yp) { spalva = Sp; sim = Simb; }
  void Kita(int a, int b, int c, char s) {
    Kitas(KoksX()+a, KoksY()+b);
    spalva +=c ;
```

```
    sim =s;
  }
  void Rodo() {
    textcolor(spalva);
    gotoxy(KoksX(), KoksY());
    cprintf("%c", sim );
} };
//                    Klasė Zodis
class Zodis: Raide { char Zd[];
public:
  Zodis(char A[]):Raide(1, 1, 1, 'A')
    { strcpy (Zd, A); }
  void Spausd();
};
void Zodis::Spausd() {
  int i = 0;
  while(Zd[i] != '\0') {
    Kita(1, 1, 1, Zd[i]);
    Rodo();
    i++; }
}
//                    Pagrindinė programa
void main(){
  window(1, 1, 80, 25);
  textbackground(BLACK);
  clrscr();

  Raide A(10, 2, RED, 'G');
  A.Rodo();
  A.Kita(2, 1, 1, 'R');
  A.Rodo();

  Zodis B("Kaunas\0");
  B.Spausd();
  getch();
}
```

Įvykdžius programą ekrane skirtingomis spalvomis matysime:



6.4 pratimas. Demonstruojamas paveldimo metodo panaudojimas. Klasė **Zodis** turi metodą **Rodo**. Klasė **Raide** taip pat turi metodą **Rodo**, kurią paveldi **Zodis**.

Sukurtas objektas `Zodis B;`

`B.Rodo();` kreipinys į savo klasės objektą, o

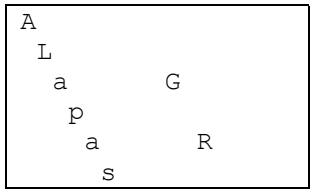
`B.Raide::Rodo();` kreipinys į paveldėtos klasės objektą.

Klasė **Zodis** turi paveldėti klasę **Raide** su nuoroda `public`.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
//                      Klasė Vieta
class Vieta { int x, y;
public:
  Vieta(int Xs, int Ys) { x = Xs; y = Ys; }
  int KoksX() { return x; }
  int KoksY() { return y; }
  void Kitas(int Xs, int Ys) { x = Xs; y = Ys; }
};
//                      Klasė Raide
class Raide: Vieta { int  spalva;
                     char sim;
public:
  Raide(int Xp, int Yp, int Sp, char Simb):  Vieta(Xp, Yp)
    { spalva = Sp;  sim = Simb; }
  void Kita(int a, int b, int c, char s) {
    Kitas(KoksX()+a, KoksY()+b);
    spalva +=c ;
    sim =s;
  }
  void Rodo() {
    textcolor(spalva);
    gotoxy(KoksX(), KoksY());
    cprintf("%c", sim);
} };
//                      Klasė Zodis
class Zodis: public Raide { char *Zd;
public:
  Zodis(char *A):Raide(1, 1, 1, 'A')
    { Zd = A ; }
  void Rodo();
};
```

```
//                      Zodis metodai
void Zodis::Rodo() {              // Metodo aprašymas
  int i =0;
  while(*(Zd+i) != '\0') {
    Kita(1, 1, 1, *(Zd+i));
    Raide::Rodo();
    i++; }
}
//                    Pagrindinė programa
void main() {
  char *Vardas ="Lapas";
  window(1, 1, 80, 25);
  textbackground(BLACK);
  clrscr();
  Raide A(10, 3, RED, 'G');
  A.Rodo();                       // Ekrane raidė G
  A.Kita(2, 2, 1, 'R' );
  A.Rodo();                       // Ekrane raidė R
  Zodis B(Vardas);
  B.Raide::Rodo();                // Ekrane raidė A
  B.Rodo();                       // Ekrane žodžio Lapas raidės
  getch();
}
```

Įvykdžius programą raides ekrane matysime skirtingomis spalvomis:



6.2. Operatorių perdengimas

Programavimo kalbose naudojami operatoriai pasižymi polimorfizmu (daugiavariantiškumu). Kaip pavyzdys gali būti operatorius **+**, kuris taikomas tiek `int`, tiek `float` tipo kintamiesiems, nors sudėties veiksmai atliekami nevienodai.

Klasėse galima naujai apibrėžti operatorius, t.y. pakeisti jų veikimą. Toks naujas operatorių perorientavimas yra vadinamas **perdengimu**

(overloading). Operatorių aprašai nuo funkcijų skiriasi vartojamu žodžiu `operator`:

`operator` <Operatoriaus simbolis>`;`

Perdengimas draudžiamas operatoriams: `.` `.*` `::` `?:`

6.5 pratimas. Demonstruojamas dviejų operatorių **+** ir **–** perkrovimas klasės **`Katinas`** objektams. Operatoriumi **+** bus sudedamos dvi eilutės, kurių viena yra objekte, o kita nurodoma parametru. Rezultate objekto eilutė pailgėja. Kitas operatorius **–** skiriamas nurodyto simbolio visų pakartojimų objekto eilutėje šalinimui.

```
#include <iostream.h>                    // Operatorių perkrovimas
#include <conio.h>
#include <string.h>
const N = 40;
//                    Klasė Katinas
class Katinas {
public:
  Katinas(char *);                       // Konstruktorius
  void operator + (char *);
  void operator - (char  );
  void Rodo();
 ~Katinas();                             // Destruktorius
private:
  char *Vardas;
};
//                    Katinas metodai
Katinas::Katinas(char *Vardas) {
  Katinas::Vardas = new char[N];
  strcpy(Katinas::Vardas, Vardas); }

void Katinas::Rodo()
  { cout << Vardas << endl; }

void Katinas::operator + (char * A)
  { strcat(Vardas, A); }

void Katinas::operator - (char C) {
  for(int i=0; *(Vardas+i) != '\0'; ) {
    if(*(Vardas + i) == C)
      for(int j=i; j<(strlen(Vardas)-1); j++)
        *(Vardas + j) = *(Vardas + j+1);
```

```
    i++;
} }

Katinas::~Katinas()
  { delete Vardas; }
//                    Pagrindinė programa
void main(void) {
  Katinas A ("Batuotas ir piktas");
  A.Rodo();
  A + " Rudas! ";
  A.Rodo();
  A - 'a';
  A.Rodo();
  getch();
}
```

Įvykdžius programą ekrane matysime:

```
Batuotas ir piktas
Batuotas ir piktas Rudas!
Btuots ir pikts Ruds!
```

6.3. Persidengiantys metodai

Patogi programavimo priemonė yra persidengiantys metodai. Gali būti persidengiantys konstruktoriai. Persidengiančios funkcijos turi tą patį vardą, tačiau gali turėti skirtingus parametrų sąrašus. vartojama funkcija atpažįstama pagal kreipinyje surašytų argumentų skaičių ir jų tipą.

6.6 pratimas. Demonstruojama klasėje **`Lapas`** persidengiantys konstruktoriai, kurių vienas duomenų laukams suteikia pradines reikšmes, fiksuotas klasės aprašyme, o kitas – vartotojo nurodytas objekto sukūrimo metu. Yra keturi metodai vardu **`Suma`**. Pagal kreipinio argumentus yra atrenkamas, kurį reikia vykdyti. Kreipinyje

`A.Suma((float)35.25);`

nurodomas argumento tipas. To nereikėtų daryti, jeigu čia būtų rašomas kintamojo vardas.

```
#include <iostream.h>                    // Metodų perdengimas
#include <conio.h>
#include <string.h>
#include <stdio.h>
```

```
//                      Klasė Lapas
class Lapas { int x;
              float y;
              char  z;
public:
  Lapas(int, float, char );            // Konstruktorius
  Lapas() { x = 0; y = 0; z = 'z'; }   // Konstruktorius
  void Rodo(char *);
  void Suma(int   Sk) { x = x + Sk; }
  void Suma(float Sk) { y = y + Sk; }
  void Suma(char  Sk) { z = Sk;     }
  void Suma(int A, char B)
    { x = x + 2.0 * A;   z = B + 4; }
};
//                      Lapas metodai
Lapas::Lapas(int A, float B, char C)
  { x = A;  y = B;  z = C; }

void Lapas::Rodo(char *Eilute) {
  cout << Eilute << " : ";
  cout << " x= " << x << "  y= " << y << "   z= " << z;
  cout << endl;
}
//                    Pagrindinė programa
void main(void) {
  Lapas A;
  Lapas B(5, 3.4, 'C');
  A.Rodo("Objektas A-1");
  B.Rodo("Objektas B-1");

  A.Suma(5);
  B.Suma('K');
  A.Rodo("Objektas A-2");
  B.Rodo("Objektas B-2");

  A.Suma((float)35.25);
  B.Suma(5, 'A');
  A.Rodo("Objektas A-3");
  B.Rodo("Objektas B-3");
  getch();
}
```

Įvykdžius programą ekrane matysime:

Pradinis A Pradinis B A pakeistas x B pakeistas z A pakeistas y B pakeistas x ir z	Objektas A-1 : x= 0 y= 0 z= z Objektas B-1 : x= 5 y= 3.4 z= C Objektas A-2 : x= 5 y= 0 z= z Objektas B-2 : x= 5 y= 3.4 z= K Objektas A-3 : x= 5 y= 35.25 z= z Objektas B-3 : x= 15 y= 3.4 z= E

6.4. Virtualios funkcijos

Tegul duota protėvio klasė **Lapas** ir jos palikuonio klasė **KnygaPirma**. Paskelbsime

`Lapas *p` – rodyklė į klasės **Lapas** objektą.
`Lapas L1` – klasės **Lapas** objektas.
`KnygaPirma Kn1` – klasės **KnygaPirma** objektas.

C++ egzistuoja taisyklė: bet kuris kintamasis, aprašytas kaip rodyklė į protėvio klasės objektą, gali būti naudojamas kaip rodyklė į palikuonio klasę. Mūsų atveju teisinga

```
p = &L1;
p = &Kn1;
```

Su rodykle `p` galima pasiekti visus objekto **Kn1** elementus, kuriuos jis paveldėjo iš klasės **Lapas**. Tačiau negalima paimti nuosavų (`private`) klasės **KnygaPirma** elementų. Jeigu duota rodyklė į palikuonio klasės objektą, tai jos pagalba negalima pasinaudoti protėvio klasės elementais.

Be to, reikia įvertinti, kad naudojant protėvio klasės tipo rodyklę palikuonio klasės objektui adresuoti, jį didinant ar mažinant, adresas keisis protėvio klasės objekto dydžiu.

Virtualios funkcijos leidžia atlikti funkcijų perkrovimą (polimorfizmo savybė) programos vykdymo metu. Tai protėvio klasėje su specifikatoriumi **virtual** aprašytos funkcijos, kurios iš naujo aprašytos vienoje (ar keliose) iš palikuonių klasių. Funkcijų vardai, grąžinamos reikšmės tipai ir argumentai nesikeičia.

Tegul duota išraiška `p = &L1` ir virtuali funkcija **Rodo**. Tegul ši funkcija aprašyta su specifikatoriumi `virtual` protėvio klasėje **Lapas** ir be specifikatoriaus `virtual` palikuonio klasėje **KnygaPirma**.

`p->Rodo()` išrinks protėvio klasės funkciją. Jei `p = &Kn1`, tai `p->Rodo()` išrinks sukurtos klasės **`KnygaPirma`** funkciją.

Reikalavimai virtualioms funkcijoms:

- Virtualių funkcijų prototipai protėvio ir palikuonio klasėse turi sutapti. Priešingu atveju funkcija bus laikoma perkraunama, o ne virtualia.
- Virtuali funkcija turi būti klasės komponente (negali turėti specifikatoriaus `friend`). Destruktorius gali turėti specifikatorių `virtual`, konstruktoriui tas draudžiama.
- Jei funkcija paskelbta virtualia, tai šią savybę ji išsaugo visoms ją paveldėjusioms klasėms. Jei kurioje nors sukurtoje klasėje virtuali funkcija neaprašyta (praleista), tai naudojama protėvio klasės versija.
- Jeigu praleidus virtualią funkciją, sukurtoje klasėje negalima naudoti protėvio klasės funkcijos, tai ji paskelbiama

  `pure virtual function.`

  `virtual` Funkcijos_tipas Funkcijos_vardas (parametrai)=0;

  tada visose ją paveldėjusiose klasėse turės būti sava virtualios funkcijos versija.
  Jei kuri nors klasė turi nors vieną `pure` funkciją, tai ji vadinama **abstrakčia**. Ją galima naudoti tik kaip protėvio.

6.7 pratimas. Sukuriama klasė **`Lapas`**, kurioje metodas **`Rodo`** pažymimas virtualiu. Turime dvi palikuonio klases **`KnygaPirma`** ir **`KnygaAntra`**. Čia taip pat yra tokios funkcijos. Sukuriami objektai ir parodomas virtualių funkcijų vartojimas.

```
#include <iostream.h>                    // Virtualūs metodai
#include <conio.h>
//                      Klasė Lapas
class Lapas {
public:
  virtual void Rodo( void )
    { cout << " Klases Lapas versija \n"; }
};
//                    Klasė KnygaPirma
class KnygaPirma: public Lapas {
public:
  virtual void Rodo(void)
    { cout << " Klases KnygaPirma versija \n";  }
};
```

```
//                    Klasė KnygaAntra
class KnygaAntra: public Lapas {
public:
  virtual void Rodo(void)
    { cout << " Klases KnygaAntra versija \n";  }
};
//                    Pagrindinė programa
void main(void) {
  Lapas *p,  L1;
  KnygaPirma Kn1;
  KnygaAntra Kn2;
  p = &L1;
  p->Rodo();            // Vykdomas klasės Lapas metodas
  p = &Kn1;
  p->Rodo();            // Vykdomas klasės KnygaPirma metodas
  p = &Kn2;
  p->Rodo();            // Vykdomas klasės KnygaAntra metodas
  getch();
}
```

Įvykdžius programą ekrane matysime:

```
Klases Lapas versija
Klases KnygaPirma versija
Klases KnygaAntra versija
```

6.8 pratimas. Modifikuotas **6.7 pratimas**. Klasėje **KnygaAntra** nėra virtualaus metodo. Sukuriama nauja klasė **Katinas**, kuri yra klasės **KnygaPirma** palikuonis. Šioje klasėje yra metodas **Rodo**, kuris nėra virtualus.

```
#include <iostream.h>        // Virtualūs metodai
#include <conio.h>
//                    Klasė Lapas
class Lapas {
public :
  virtual void Rodo(void)
    { cout << "Klases Lapas versija \n";  }
};
//                    Klasė KnygaPirma
class KnygaPirma: public Lapas {
public:
  virtual void Rodo(void)
    { cout << "Klases KnygaPirma versija \n";  }
};
```

```
//                    Klasė KnygaAntra
class KnygaAntra: public Lapas {
public:
  void Matau(void)
    { cout << "Klases KnygaAntra Matau \n";  }
};
//                     Klasė Katinas
class Katinas: public KnygaPirma {
public:
  void Rodo()
    { cout << "Klases Katinas versija \n";  }
};
//                   Pagrindinė programa
void main(void) {
  Lapas *p,  L1;
  KnygaPirma Kn1;
  KnygaAntra Kn2;
  Katinas    Cat;
  p = &L1;
  p->Rodo();            // Vykdomas klasės Lapas metodas
  cout << "\n";
  p = &Kn1;
  p->Rodo();            // Vykdomas klasės KnygaPirma metodas
  p->Lapas::Rodo();     // Vykdomas klasės Lapas metodas
  Kn1.Rodo();           // Vykdomas klasės KnygaPirma metodas
  Kn1.Lapas::Rodo();    // Vykdomas klasės Lapas metodas
  cout << "\n";
  p = &Kn2;
  p->Rodo();            // Vykdomas klasės Lapas metodas
  Kn2.Matau();
//  p->Matau();           Kreipinys negalimas
  cout << "\n";
  p = &Cat;
  p->Rodo();            // Vykdomas klasės Katinas metodas
  Cat.Rodo();           // Vykdomas klasės Katinas metodas
  p->Lapas::Rodo();       // Vykdomas klasės Lapas metodas
//  p->KnygaPirma::Rodo();    Kreipinys negalimas
  Cat.Lapas::Rodo();      // Vykdomas klasės Lapas metodas
  Cat.KnygaPirma::Rodo(); // Klasės KnygaPirma metodas
  getch();
}
```

Įvykdžius programą ekrane matysime:

```
Klases Lapas versija

Klases KnygaPirma versija
Klases Lapas versija
Klases KnygaPirma versija
Klases Lapas versija

Klases Lapas versija
Klases KnygaAntra Matau

Klases Katinas versija
Klases Katinas versija
Klases Lapas versija
Klases Lapas versija
Klases KnygaPirma versija
```

6.9 pratimas. Klasėje **`Lapas`** aprašomas metodas **`Rodo`** abstraktus. Šios klasės palikuonyse yra savos funkcijos, kurios atliekamos vykdymo metu.

```
#include <iostream.h>          // Virtualūs metodai
#include <conio.h>
//                      Klasė Lapas
class Lapas {
public:
  virtual void Rodo(void) = 0;
};
//                   Klasė KnygaPirma
class KnygaPirma: public Lapas {
public:
  void Rodo(void)
    { cout << "Klasės KnygaPirma versija \n";  }
};
//                   Klasė KnygaAntra
class KnygaAntra: public Lapas {
public:
  void Rodo(void)
    { cout << "Klasės KnygaAntra versija \n";  }
};
//                     Klasė Katinas
class Katinas:public KnygaPirma {
public:
  void Rodo()
```

```
    { cout << "Klasės Katinas versija \n";  }
};
//                  Pagrindinė programa
void main(void) {
  Lapas *p;
  KnygaPirma Kn1;
  KnygaAntra Kn2;
  Katinas    Cat;

  p = &Kn1;
  p->Rodo();        // Vykdomas klasės KnygaPirma metodas
  p = &Kn2;
  p->Rodo();        //Vykdomas klasės Lapas KnygaAntra metodas
  p = &Cat;
  p->Rodo();        // Vykdomas klasės Katinas metodas
  getch();
}
```

Įvykdžius programą ekrane matysime:

```
Klasės KnygaPirma versija
Klasės KnygaAntra versija
Klasės Katinas versija
```

6.5. Draugiškos (friend) klasės, funkcijos

Yra galimybė klasėms tarpusavyje draugauti, t.y.klasė **A** gali leisti kitai klasei **B** naudotis jos priemonėmis. Tokiu atveju klasėje **A** yra skelbiama klasė **B** draugiška – aprašoma su atributu **friend**. Klasėje **B** rašomi metodai, kurie naudojasi klasės **A** priemonėmis kaip savomis. Draugiška klasė skelbiama `public` dalyje.

6.10 pratimas. Klasė **Knyga** skelbia draugiška klasę **Lentyna**. Kadangi tos klasės aprašas yra toliau, tai jos antraštė-prototipas rašoma prieš klasės **Knyga** aprašą. Klasėje **Lentyna** esančios priemonės naudoja **Knyga** duomenis.

```
#include <iostream.h>
#include <string.h>
#include <conio.h>

class Lentyna;                          // Klasės antraštė
```

```
//                        Klasė Knyga
class Knyga {
public:
  Knyga(char *, char *, char *);     // Konstruktorius
  void Rodo(void);                   // Išvedimas ekrane
  friend Lentyna;                    // Draugiška klasė
private:
  char pav[64];
  char autorius[64];
  char leidykla[64];
};
//                      Knyga metodai
Knyga::Knyga(char *pav, char *autorius, char *leidykla) {
strcpy(Knyga::pav, pav);
  strcpy(Knyga::autorius, autorius);
  strcpy(Knyga::leidykla, leidykla);
}

void Knyga::Rodo(void) {
  cout << "Pavadinimas: " << pav << endl;
  cout << "Autorius: " << autorius << endl;
  cout << "Leidykla: " << leidykla << endl;
}
//                      Klasė Lentyna
class Lentyna {
public:
  void Keisti(Knyga *, char *);
      char *Rasti(Knyga);
};
//                      Lentyna metodai
// Klasės Knyga tipo objekte pakeičia leidyklos pavadinimą
void Lentyna::Keisti(Knyga * Kn, char * Leid)
  { strcpy(Kn->leidykla, Leid ); }

// Pranešamas klasės Knyga tipo objekto leidyklos pavadinimas
char *Lentyna::Rasti(Knyga Kn) {
  static char vardas[64];
  strcpy(vardas, Kn.leidykla);
  return(vardas);
}
//                    Pagrindinė programa
void main(void) {
  Knyga K("Informatika I dalis",
```

```
          " J.Adomavicius ir kt.", "KTU");
  Lentyna L;
  K.Rodo();
  L.Keisti(&K, "Technologija");
  K.Rodo();
  getch();
}
```

Įvykdžius programą ekrane matysime:

```
Pavadinimas: Informatika I dalis
Autorius: J.Adomavicius ir kt.
Leidykla: KTU
Pavadinimas: Informatika I dalis
Autorius: J.Adomavicius ir kt.
Leidykla: Technologija
```

Funkcijai, nepriklausančiai jokiai klasei, gali būti suteiktas draugiškos titulas. Ta funkcija įgyja teisę naudotis klasės duomenimis ir metodais. Tose klasėse, kurios tą funkciją kviečia būti draugiška, rašomas funkcijos prototipas su atributu `friend`.

6.11 pratimas. Funkcija **`Lygu`** skelbiama draugiška dviejose klasėse: **`Point`** ir **`Circle`**. Funkcija palygina skirtingų klasių objektų naudojamas spalvas ir praneša ekrane, ar jos lygios, ar nelygios.

Klasė **`Vieta`** skirta saugoti žymeklio vietos ekrane koordinatėms. Konstruktorius **`Vieta`** apibrėžia sukurto objekto pradinę poziciją. Metodas **`set`** skirtas keisti koordinatėms, o **`getX`** ir **`getY`** koordinatėms spausdinti ekrane.

Klasė **`Point`** paveldi klasę **`Vieta`** ir turi duomenų lauką spalvos kodui saugoti. Konstruktorius formuoja taško vietos ekrane koordinates ir spalvą. Metodas **`PutPoint`** padeda tašką grafiniame ekrane. Klasėje skelbiama funkcija **`Lygu`**.

Klasė **`Circle`** paveldi klasę **`Vieta`** ir turi duomenų laukus apskritimo spalvos kodui bei spinduliui saugoti ir papildomą darbui. Konstruktorius formuoja apskritimo centro koordinačių, spindulio ir spalvos pradines reikšmes. Metodas **`PutCircle`** brėžia apskritimą grafiniame ekrane. Klasėje skelbiama funkcija **`Lygu.`**

```
#include <iostream.h>
#include <graphics.h>
#include <conio.h>
```

```
#include <stdio.h>
#include <stdlib.h>

class Circle;
//                    Klasė Vieta
class Vieta {
protected: int x, y;
public:
  Vieta(int Xp, int Yp)  { x = Xp;  y = Yp; }
  void set(int a, int b) { x = a;   y = b;  }
  int getX() { cout << "x= " << x << endl; return x; }
  int getY() { cout << "y= " << y << endl; return y; }
};
//                    Klasė Point
class Point: public Vieta { int Spalva;
public:
  Point(int Xp, int Yp, int Cp);
  void PutPoint() { putpixel(x, y, Spalva); }
  friend void Lygu(Point p, Circle c);
};
//                    Klasė Circle
class Circle: public Vieta{ int Spalva, T, R;
public:
  Circle(int Xp, int Yp, int Cp, int Rp);
  void PutCircle() {
    T = getcolor();
    setcolor(Spalva);
    circle(x, y, R);
    setcolor(T);   }
  friend void Lygu(Point p, Circle c);
};
//                       Metodai
Point::Point(int Xp, int Yp, int Cp): Vieta( Xp, Yp )
  { Spalva = Cp; }

Circle::Circle(int Xp, int Yp, int Cp, int Rp):
                                  Vieta(Xp, Yp)
  { Spalva = Cp;  R = Rp; }

void Lygu(Point p, Circle c) {
  if (p.Spalva == c.Spalva)
    cout << "Spalvos sutampa\n";
  else cout << " Skirtingos spalvos \n";
}
```

```
//                    Pagrindinė programa
void Grafika();

void main(){
  Grafika();          // Ekranas paruošiamas darbui grafiniame rėžime

  Point Taskas(200, 100, 3);
  Circle C1 (400, 200, 3, 100);
  Circle C2 (200, 200, 1, 50 );

  Taskas.PutPoint();          // Ekrane padedamas taškas
  C1.PutCircle();             // Ekrane brėžiamas apskritimas
  C2.PutCircle();             // Ekrane brėžiamas apskritimas

  Lygu(Taskas, C1);           // Pranešama, kad spalvos lygios
  Lygu(Taskas, C2);           // Pranešama, kad spalvos skirtingos

  getch();
  closegraph();
}
//                Grafinio ekrano paruošimas
void Grafika() {
  int A = DETECT, B;
  initgraph(&A, &B, "c:\\");
  if (graphresult() != grOk) {
    printf("Klaida: %i %i\n ", A, B);
    exit(1);}
}
```

Galima kitos klasės metodus skelbti draugiškais savo klasėje ir leisti jiems naudotis duomenų laukais. Tam reikia klasėje **`A`** parašyti su atributu `friend` klasės **`B`** metodų (funkcijų) prototipus.

6.12 pratimas. Turime klasę **`Kitoks`**, kurioje yra trys metodai, skirti darbui su kitos klasės duomenų laukais: keisti apskritimo vietai ekrane, spindulio reikšmei ir spalvai. Klasėje **`Circle`** tie metodai skelbiami draugiškais. Pagrindinėje funkcijoje sukuriami du klasės **`Circle`** objektai **`C1`** ir **`C2`**. Nubrėžiamas žydras apskritimas **`C1`** duomenimis. Sukurtas klasės Kito**k**s objektas **`CC`** pakeičia objekto **`C1`** visus duomenis. Nubrėžiamas raudonas apskritimas. Pakeičiamas **`C1`** spindulys ir nubrėžiamas raudonas mažesnio spindulio apskritimas

(gauname du raudonus koncentriškus apskritimus). Toliau tas pat padaroma su **C2** apskritimu: du koncentriški violetiniai apskritimai.

```
#include <iostream.h>
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

class Circle;
//                    Klasė Kitoks
class Kitoks {
public:
  void Dydis(Circle *, int);
  void Vieta(Circle *, int, int);
  void Spalva(Circle *, int);
};
//                    Klasė Circle
class Circle { int Spalva, T, R, x, y;
public:
  Circle(int Xp, int Yp, int Cp, int Rp);
  void PutCircle() {
    T = getcolor();
    setcolor(Spalva);
    circle(x, y, R);
    setcolor(T); }
  friend void Kitoks::Dydis(Circle *, int);
  friend void Kitoks::Vieta(Circle *, int, int);
  friend void Kitoks::Spalva(Circle *, int);
};
//                       Metodai
Circle::Circle(int Xp, int Yp, int Cp, int Rp)
  { Spalva = Cp; R = Rp; x = Xp; y = Yp;  }

void Kitoks::Dydis(Circle *CC, int D) { CC->R = D; }

void Kitoks::Vieta(Circle *CC, int Nx, int Ny)
  { CC->x = Nx;  CC->y = Ny; }

void Kitoks::Spalva(Circle *CC, int S)
  { CC->Spalva = S; };
//                  Pagrindinė programa
void Grafika();
```

```
void main(){
  Grafika();                // Grafinio ekrano paruošimas

  Circle C1(400, 200, 3, 100);  // Objektas C1
  Circle C2(300, 300, 5, 50 );  // Objektas C2

  C1.PutCircle();           // Brėžiamas C1 žydras apskritimas

  Kitoks CC;           // Objektas CC darbui su Circle tipo objektas

  CC.Vieta(&C1, 100, 100);// Keičiama C1 vieta
  CC.Spalva(&C1, RED);     // Keičiama C1 spalva
  C1.PutCircle();           // Brėžiamas C1 raudonas apskritimas

  CC.Dydis(&C1, 25);        // Keičiamas C1 spindulys
  C1.PutCircle();           // Brėžiamas C1 raudonas apskritimas

  C2.PutCircle();           // Brėžiamas C2 violetinis apskritimas
  CC.Dydis(&CC, 30);        // Keičiamas spindulys
  C2.PutCircle();           // Brėžiamas C2 violetinis apskritimas

  getch();
  closegraph();
}
//                Grafinio ekrano paruošimas
void Grafika() {
  int A = DETECT, B;
  initgraph(&A, &B, "c:\\");
  if(graphresult() != grOk) {
    printf("Klaida: %i %i\n ", A, B);
    exit(1); }
}
```

6.6. Dinaminiai objektai

Galima turėti rodykles į objektus, objektų sąrašus: masyvus, rodyklių, masyvus, sąrašines struktūras.

Atminties skyrimui patogu naudoti operatorių **new**:

<rodyklė**> = new <**tipas**>;**

Dinamiškai skirtos atminties atsisakoma operatoriumi `delete` arba funkcija `free`:

`delete <`rodyklė**`>;`**
`void free (<`rodyklė**`>);`**

6.13 pratimas. Turime klasę **`Lapas`**. Sukuriama rodyklė `p` į klasę **`Lapas`**. Sukuriamas dinaminis objektas, kurio adresas saugomas rodyklėje `p`.

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
//                      Klasė Lapas
class Lapas { char *L;
public :
  Lapas(char * T) { L = T;}
  void Kitas(char *K) { strcpy(L, K); }
  void Rodo(void) {
    if (L) cout << L << " lapas \n";
    else cout << "Neturiu lapo\n"; }
};
//                    Pagrindinė programa
void main(void) {
  Lapas *p;
  p = new Lapas("Klevo ");
  if (!p) { cout << "Truksta atminties\n"; exit( 0 ); }
  p->Rodo();
  p->Kitas("Liepa");
  p->Rodo();
  free(p);
}
```

Įvykdžius programą ekrane matysime:

```
Klevo  lapas
Liepa lapas
```

6.14 pratimas. Turime klasę **`Lapas`**. Sukuriame rodyklių masyvą **`p`**, kuriame saugosime rodykles į tris objektus klasės tipo **`Lapas`**. Suformuojamas rodyklių į objektus masyvas. Objektai saugo tuščias eilutes. Po to klaviatūra suvedami žodžiai ir patalpinami į objektų duomenų laukus **`L`**. Programos pabaigoje atspausdinami objektų saugomi žodžiai ir objektai pašalinami iš atminties.

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
//                      Klasė Lapas
class Lapas { char *L;
public :
  Lapas() { L = NULL; }
  void Kitas(char *K) {
    L = new char[10];
    strcpy(L, K);     }
  void Rodo(void) {
   if (L) cout << L << " lapas \n";
   else   cout << "Neturiu lapo\n"; }
};
//                    Pagrindinė programa
void main(void) {
  int i;
  Lapas *p[3];
  char T[10];
  for (i=0; i<3; i++) {              // Masyvo sudarymas
    p[i] = new Lapas();
    cout << "*** ";
    p[i]->Rodo();  }
  cout << "--------------\n";
  for (i=0; i<3; i++) {              // Duomenų įvedimas
    cout << "Iveskite: ";
    cin >> T;
    cout << endl;
    p[i]->Kitas(T);   }              // Spausdinimas
  for (i=0; i<3; i++)  p[i]->Rodo();
  for (i=0; i<3; i++)  free(p[i]);  // Naikinimas
  cout << "Pabaiga\n";
}
```

Įvykdžius programą ekrane matysime:

```
*** Neturiu lapo
*** Neturiu lapo
*** Neturiu lapo
--------------
Iveskite: Liepa

Iveskite: Klevas

Iveskite: Maumedis
```

```
Liepa lapas
Klevas lapas
Maumedis lapas
Pabaiga
```

6.15 pratimas. Turime klasę **`Lapas`**. Sukuriamas tiesinis dinaminis sąrašas, kurio elementais yra klasės **`Lapas`** tipo objektai. Parodomas sąrašo objektų užpildymas duomenimis, sąrašo spausdinimas (atvirkštinė seka, nes sąrašas buvo formuojamas į "pradžią") ir sąrašo sunaikinimas.

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
//                      Klasė Lapas
class Lapas { char *L;
public :
  Lapas() { L = NULL; }
  void Kitas(char *K) {
    L = new char[10];
    strcpy(L, K);     }
  void Rodo(void) {
    if (L) cout << L << " lapas \n";
    else   cout << "Neturiu lapo\n"; }
 ~Lapas() { free(L); }
};
//                         Sąrašas
struct sar { Lapas *Medis;
             sar   *sek;};
//                   Pagrindinė programa
sar *Naikinti(sar *);

void main(void) {
  sar *P = NULL, *R;
  int i;
  char T[10];
  for(i=1; i<=3; i++) {              // Sąrašo formavimas
    cout << "\nIveskite: ";
    cin>>T;
    R = new sar;
    R->sek = P;
    P = R;
    P->Medis = new Lapas();
```

```
    P->Medis->Kitas(T); }
  R = P;                                  // Sąrašo spausdinimas
  while(R) {
    R->Medis->Rodo();
    R = R->sek; }
  P = Naikinti(P);                        // Sąrašo naikinimas
  if (!P)  cout << "Sarasas tuscias\n";
  cout << "Pabaiga\n";
}
//                      Sąrašo naikinimas
sar *Naikinti(sar *P) {
  sar *R;
  while(P) {
    R = P;
    P = P->sek;
    R->Medis->~Lapas();
    free( R );  }
  return P;
}
```

Įvykdžius programą ekrane matysime:

```
Iveskite: Liepa

Iveskite: Klevas

Iveskite: Slyva
Slyva lapas
Klevas lapas
Liepa lapas
Sarasas tuscias
Pabaiga
```

7 skyrius. Tiesiniai vienkrypčiai sąrašai

7.1. Sąrašo struktūra

Sąrašais vadinami tokie duomenų rinkiniai, kurių elementus apibūdina ne tik jų reikšmės, bet ir tvarkymo taisyklės, ryšiai tarp elementų. Keičiant elementų tipus, ryšių tarp elementų būdus ir tvarkymo taisykles, galima sudaryti daugybę įvairių tipų sąrašų (eiles, stekus, dekus, medžius), todėl universalios sąrašo tipo struktūros sudarymas yra problematiškas. Paprasčiausi sąrašai gali būti realizuojami masyvuose, papildant juos tvarkymo procedūromis ir užpildymo charakteristika. Universalesnė sąrašų sudarymo priemonė yra dinaminės struktūros, kurios sudaromos iš elementų su nuorodomis. Tokiuose elementuose saugomos ne tik jų reikšmės, bet ir ryšio su kitais elementais aprašymai:

Dinaminės struktūros elemento reikšmė gali būti bet kokia statinė arba dinaminė struktūra, o ryšio dalyje yra rašomos rodyklės į kitus to paties sąrašo elementus. Kadangi dinaminių struktūrų elementų reikšmės ir nuorodos yra aprašomos skirtingų tipų duomenimis, sąrašų elementai realizuojami įrašais, kurių struktūra yra tokia:

```
struct <Vardas> { <Reikšmės laukai>;
                  <Ryšio dalies laukai>; }
```

Homogeninėse dinaminėse struktūrose, kuriose visi elementai yra vienodo tipo, aprašant ryšio dalies rodykles, reikia nurodyti pačios naujai aprašomos struktūros tipą. Kreipimasis struktūros aprašyme į save pačią vadinamas rekursija, o struktūros, kuriose yra tokie kreipiniai, vadinamos rekursyviomis. Rekursyvios struktūros elemento aprašo pavyzdys:

```
struct list {int data;
             struct list *next; }  // Rekursyvi nuoroda
```

Iš elementų, kurių ryšio dalyje yra viena rodyklė, galima sudaryti tik tiesiniais sąrašais vadinamas nuosekliai sujungtas grandines. Papildant

tiesinius sąrašus tvarkymo procedūromis ir išorinio ryšio priemonėmis, galima sudaryti tokias tipines jų modifikacijas:

- **tiesinius sąrašus**, kuriuose nėra apribojimų sąrašo elementų analizei, tvarkymui ir apdorojimui;
- **eiles**, kuriose nauji elementai prijungiami sąrašo gale, o skaitymui yra prieinamas tik pirmasis elementas (struktūra FIFO – first in, first out);
- **stekus**, kuriuose galima skaityti tik vėliausiai įrašytą elementą (struktūra LIFO – last in, first out);
- **dekus**, kuriuose elementai gali būti rašomi ir skaitomi tiek sąrašo pradžioje, tiek gale (struktūra DEQUE – double-ended que).

Tokios sąrašinės struktūros labai plačiai vartojamos ne tik taikomosiose, bet ir sisteminėse programose. Pavyzdžiui, stekuose yra saugomi pertraukiamų procesų parametrai, organizuojami lokalūs duomenys. Eilės yra populiari pagalbinių (buferinių) atminčių, kuriose kaupiami iš lėtų įrenginių gaunami arba į juos siunčiami duomenys, organizavimo priemonė.

Kiekvieno sąrašo tipo realizavimui skirta dinaminė struktūra turi turėti jos elementus aprašančią struktūrą, rodyklę arba rodykles į išoriniams ryšiams skirtus elementus ir tokias tvarkymo operacijas:

- naujo sąrašo sukūrimo;
- naujo elemento prijungimo;
- elemento pašalinimo;
- sąrašo skaitymo ir analizės;
- sąrašo tvarkymo.

Sudarant sąrašų tvarkymo procedūras, reikalingas papildomas susitarimas apie tai, kaip bus žymima sąrašo pabaiga ir kaip bus žymimas tuščias sąrašas. Yra priimta tiek sąrašo pabaigą, tiek tuščią sąrašą dinaminėse struktūrose žymėti nuline rodykle **`NULL`**.

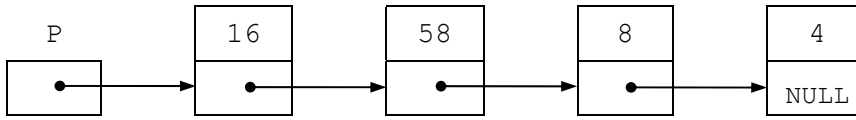
7.2. Tiesinis dinaminis sąrašas

Ryšys su sąrašu palaikomas naudojant rodyklę į jo pradžią. Rekomenduojama sąrašo elemento struktūra yra iš dviejų laukų: duomenims saugoti ir ryšio rodyklei užrašyti. Tokia struktūra supaprastina veiksmus su sąrašo elementais.

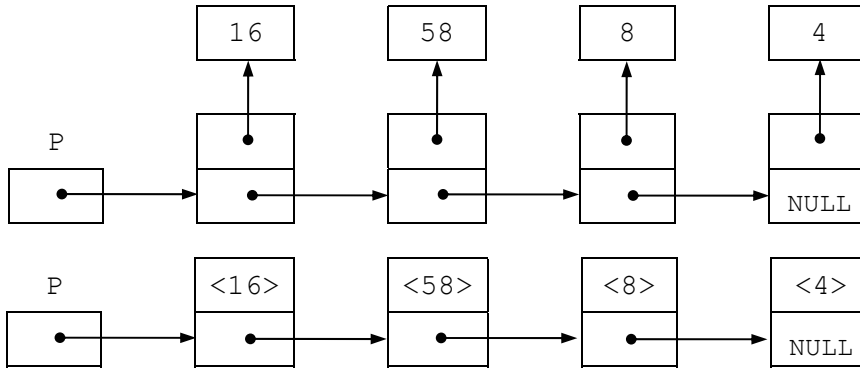
Sąrašo elemento struktūra gali būti aprašoma vienu arba dviem sakiniais:

```
struct List {                 struct List {
  int Sk;                       int Sk;
  struct List *next;            struct List *next; };
            } *First;         List  *First;
```

Tiesinio sąrašo grafinis vaizdas parodytas 7.1 pav. Čia elementas vaizduojamas stačiakampiu, padalintu į tiek dalių, kiek yra laukų jo struktūroje: duomenų ir adreso. Duomenų lauke saugomos reikšmės. Rodyklės tipo lauke saugomas adresas (nuoroda) į kitą sąrašo elementą. Nuoroda pavaizduota linija su rodykle gale. Tos linijos pradžia yra nuorodos lauko viduje. Rodyklė remiasi į elementą vaizduojantį stačiakampį bet kurioje vietoje. Jeigu nuoroda neegzistuoja, tuomet linija nėra brėžiama ir laukas lieka tuščias. Tai reiškia, kad adreso reikšmė neapibrėžta, "šiukšlė". Realiose programose tokia situacija yra neleistina.



7.1 pav. Tiesinio sąrašo vaizdavimas



7.2 pav. Nuorodos į reikšmę vaizdavimas

Būtina užrašyti tuščio adreso reikšmę – konstantą `NULL`. Grafiškai tai gali būti žymima įvairiais sutartiniais ženklais, kas leidžia konstruoti mažesnės apimties paveikslėlius. Ženklai naudojami, kuomet nagrinėjamos struktūros nėra siejamos su konkrečia realizacija (pvz., Turbo Paskalyje tuščio adreso konstanta yra `Nil`). Sąrašus suvokti lengviau, kai sutartinių pažymėjimų mažiau, todėl bus rašoma `NULL`. Be to paveikslėliuose bus

naudojamas tokio tipo pažymėjimas: ⟨reikšmė⟩. Tai reikš adresą į vietą, kur saugoma nurodyta reikšmė, pavyzdžiui skaičiaus 16 nuoroda bus užrašyta ⟨16⟩ (7.2 pav.).

Tiesinio sąrašo formavimo ir peržiūros iliustracijai panaudosime programą **Sąrašo formavimas.** Šioje programoje yra sukurta klasė DinSar, kurioje yra aprašytas kintamasis P (dinaminio sąrašo pradžios rodyklė), konstruktorius, destruktorius, bei metodai sąrašui formuoti ir spausdinti. Programa naudoja klasės DinSar objektą A. Klasėje yra keturi metodai:

- sąrašo formavimui parašytas metodas Formuoti. Jame yra imami sveiki skaičiai iš failo ir perduodami metodui **Elementas**, kuris sukuria dinaminį elementą su ta reikšme ir prijungia prie formuojamo sąrašo pradžios (steko formavimo būdas);
- sąrašo peržiūrai yra metodas **Spausdinti**, kuris sąrašo elementų reikšmes surašo ekrane viena eilute;
- klasės DinSar konstruktorius **DinSar** sukuria objektą A ir šio objekto sąrašo pradžios rodyklei P suteikia reikšmę NULL, o destruktorius **~DinSar** naikina iš atminties sąrašą ir objektą A.

Detaliau veiksmai bus aptariami atskiruose skyreliuose.

```
//            Sąrašo formavimas (stekas)
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
//                Struktūros tipo apibrėžimas
typedef struct list {int sk;
                     struct list *next; } sar;

//               Klasės Dinsar apibrėžimas
class DinSar {
  sar *P;                              // Sąrašo pradžia
  public:
  DinSar();                            // Konstruktorius
 ~DinSar();                           // Destruktorius
  void  Formuoti(FILE *);            // Formuoja sąrašą
  void  Elementas(int);              // Prijungia elementą
  void  Spausdinti();                  // Spausdina sąrašą
};
//                    Pagrindinė programa
void main() {
  FILE *D;
```

```
  DinSar A;                          // Klasės DinSar objektas
  D = fopen("Duom.dat", "r");
  if (D == NULL) {
    cout << "Failo 'Duom.dat' nepavyko atidaryti";
    exit (1); }
  A.Formuoti(D);                     // Formuojamas sąrašas
  fclose(D);
  A.Spausdinti();                    // Spausdinamas sąrašas
  A.~DinSar();
}
//                    Konstruktorius
DinSar::DinSar() { P = NULL; }
//                     Destruktorius
DinSar::~DinSar() {
  sar *D = P;
  while(P != NULL) {
  D = P;                          // Rodyklė į naikinamą elementą
  P = P->next;                    // Kito elemento adresas
  delete( D );  }                 // Elemento naikinimas
}
//                 Formuoja netiesioginį sąrašą
void DinSar::Formuoti(FILE *F) {
  int k;
  while(!feof(F)) {
    fscanf(F, "%i", &k);
    Elementas(k); }
}
//                    Prijungia elementą
void DinSar::Elementas(int Sk) {
  sar *R;
  R       = new sar;              // Naujo elemento sukūrimas
  R->sk   = Sk;                   // Užpildymas duomenimis
  R->next = P;                    // Prijungimas prie sąrašo
  P       = R;                    // Sąrašo pradžios pakeitimas
}
//                   Sąrašo spausdinimas
void DinSar::Spausdinti() {
  sar *D = P;
  while(D != NULL) {
    cout << D->sk << " ";         // Reikšmės spausdinimas
   D = D->next; }                 // Kito elemento adresas
  cout << "\n";
}
```

7.3. Tiesinio sąrašo formavimo iliustracija

Sąrašo formavimo esmę sudaro naujų elementų prijungimas prie sąrašo. Metodo **`Formuoti`** vaidmuo yra organizuoti nuoseklų duomenų skaitymą iš failo ir jų rašymą į dinaminį sąrašą. Metodui **`Elementas`** perduodama nauja reikšmė `k`. Metodas **`Elementas`** sukuria naują elementą, jį užpildo duomenimis ir prijungia formuojamo sąrašo pradžioje (steko principas), pakeisdamas sąrašo pradžios adresą. Sąrašo formavimo fragmentas:

```
typedef struct list { int sk;
                      struct list *next; } sar;
//                  Formuoja netiesioginį sąrašą
void DinSar::Formuoti(FILE *F) {
  int k;
  while(!feof(F)) {
    fscanf(F, "%i", &k);
    Elementas(k);  }
}
//                      Prijungia elementą
void DinSar::Elementas(int Sk) {
  sar *R;
  /*1*/  R       = new sar;
  /*2*/  R->sk   = Sk;
  /*3*/  R->next = P;
  /*4*/  P       = R;
}
```

7.3 pav. Rodyklės `P` reikšmė prieš ciklą

Metodo **`Formuoti`** darbo pradžioje formuojamas sąrašas yra tuščias, t.y. sąrašo pradžios rodyklės `P` reikšmė yra lygi `NULL`. Jeigu failas bus tuščias, tai tokia reikšmė ir pasiliks metodui baigus darbą. Ši situacija grafiškai parodyta 7.3 pav.

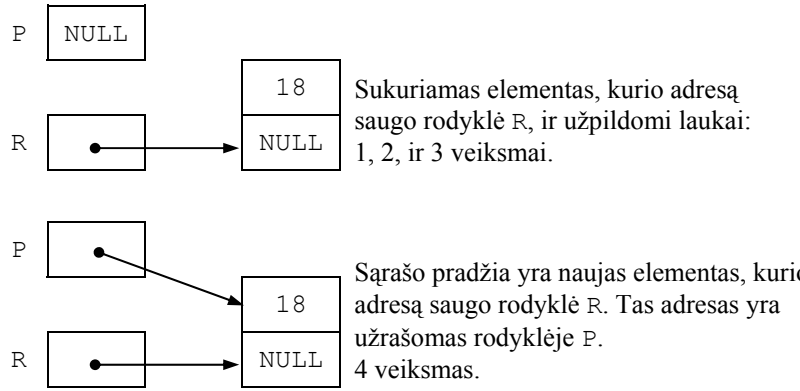
Tarkime duomenų faile "`Duom.dat`" yra tokia skaičių seka:

```
18 7 4 9
```

Pirmą kartą cikle (7.4 pav.) iš failo perskaitoma reikšmė `k = 18`. Ji perduodama metodui `Elementas`. Šis metodas sukuria naują elementą

su ta reikšme (1 ir 2 sakiniai), po to jis prijungiamas prie sąrašo pradžios (3 sakinys), o sąrašo pradžios rodyklė perkeliama prie to naujo elemento (4 sakinys), t.y. naujasis elementas tampa pirmuoju sąraše.

Antrą kartą cikle gauta nauja reikšmė `k = 7` tampa nauju elementu jau turimo sąrašo pradžioje (7.5 pav.).

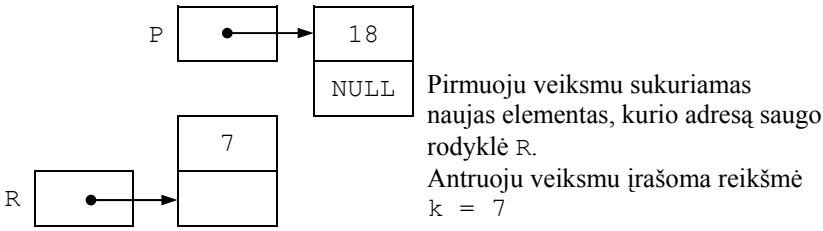


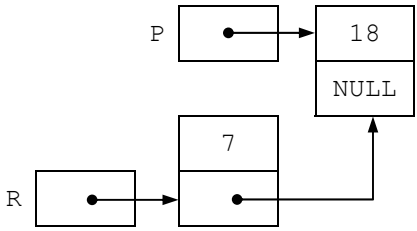
7.4 pav. Pirmasis elementas sąraše

Perkėlus visus faile esančius skaičius į sąrašą, sąrašo pradžios rodyklė `P` rodys sąrašo pradžią (saugo pirmojo sąrašo elemento adresą).

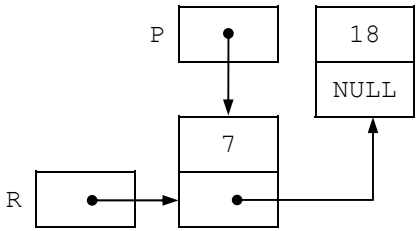
Suformuoto sąrašo vaizdas parodytas 7.6 pav. Matome, kad suformuotas sąrašas saugo failo duomenis priešinga tvarka, nes nauji elementai buvo prijungiami prie sąrašo pradžios.

Pastaba. Veiksmuose su sąrašo elementais sąrašo pradžios rodyklės `P` prarasti negalima, nes tuomet sąraše esantys duomenys bus neprieinami (prarasti).



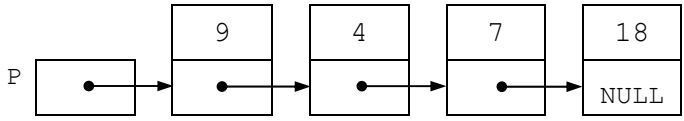


Trečiuoju veiksmu naujas elementas prijungiamas prie sąrašo pradžios.



Ketvirtuoju veiksmu sąrašo pradžios rodyklė perkeliama prie naujo elemento.

7.5 pav. Antrasis elementas sąraše



7.6 pav. Suformuotas sąrašas: faile buvo skaičiai **18 7 4 9**

7.4. Tiesinio sąrašo peržiūra

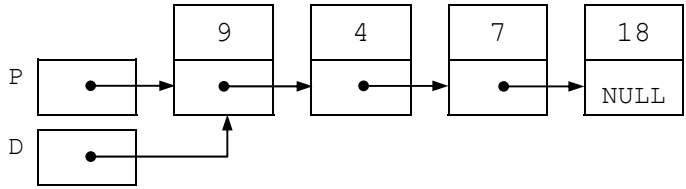
Vienas iš daugelio veiksmų su sąrašo elementais yra nuosekli jų peržiūra nuo sąrašo pradžios iki galo spausdinant duomenis. Tam panaudojama papildoma rodyklė, kurios pradinė reikšmė yra sąrašo pradžios rodyklės `P` saugomas adresas. Toliau nuosekliai darbinės rodyklės reikšmė keičiama kito elemento iš sąrašo reikšme tol, kol peržiūrimas visas sąrašas (7.7 pav.).

```
void DinSar::Spausdinti() {
  sar *D = P;
  while(D != NULL) {
    cout << D->sk << " ";       // Reikšmės spausdinimas
    D = D->next; }              // Kito elemento adresas
  cout << "\n";
```

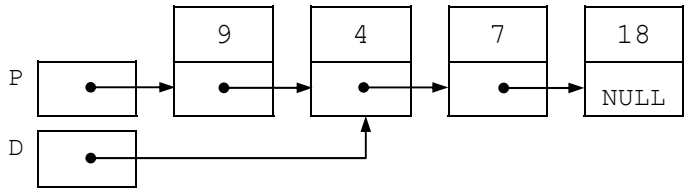
Turintiems programavimo įgūdžius siūlomas lakoniškesnis funkcijos užrašas:

```
void DinSar::Spausdinti() {
  sar *D = P;
  while(D) {
    cout << D->sk << " ";       // Reikšmės spausdinimas
    D = D->next; }              // Kito elemento adresas
  cout << "\n";
}
```

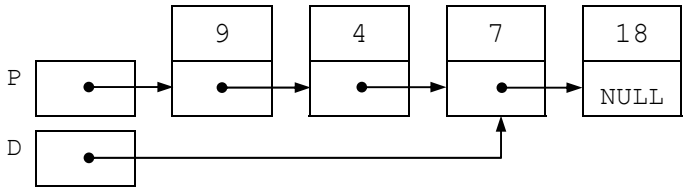
Kiekviena programa privalo tvarkyti naudojamą atmintį taip, kad nesusidarytų avarinių situacijų. Vienas iš tokių veiksmų yra nebereikalingo sąrašo pašalinimas iš atminties. Pavyzdžio programoje tam skirta destruktorius **`~DinSar`**. Čia naudojama papildoma rodyklė `D` šalinamo elemento adresui atsiminti. Sąrašas nuosekliai peržiūrimas naudojant jo rodyklę `P`, kuri turi sąrašo pradžios adresą. Pirmu veiksmu atsimenamas sąrašo pradžios elemento adresas (`D = P`). Po to sąrašo pradžia yra perkeliama prie tolimesnio elemento (`P = P->next;`), o "atjungtas" nuo sąrašo elementas `D` yra pašalinamas.



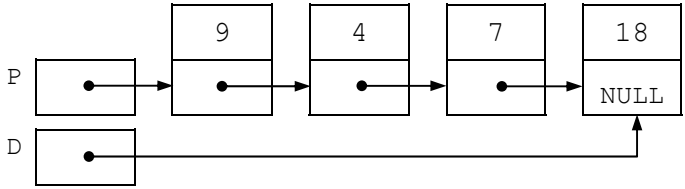
Darbinei rodyklei `D` suteikiamas sąrašo pradžios adresas: `D = P;`
Galima atspausdinti pirmojo elemento reikšmę **9**: `cout << D->sk;`



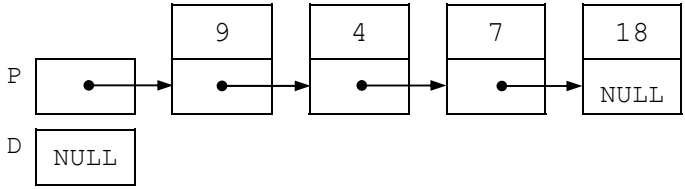
Pereinama prie kito elemento: `D = D->next;`
Galima atspausdinti elemento reikšmę **4**: `cout << D->sk;`



Pereinama prie kito elemento: `D = D->next;`
Galima atspausdinti elemento reikšmę **7**: `cout << D->sk;`



Pereinama prie kito elemento: `D = D->next;`
Galima atspausdinti elemento reikšmę **18**: `cout << D->sk;`



Pereinama prie kito elemento: `D = D->next;`
Papildoma rodyklė `D` gauna paskutinio elemento ryšio dalyje esantį adresą `NULL`. Ši reikšmė yra sąrašo peržiūros pabaigos požymis.

7.7 pav. Sąrašo elementų peržiūra

7.5. Tiesinio sąrašo papildymas

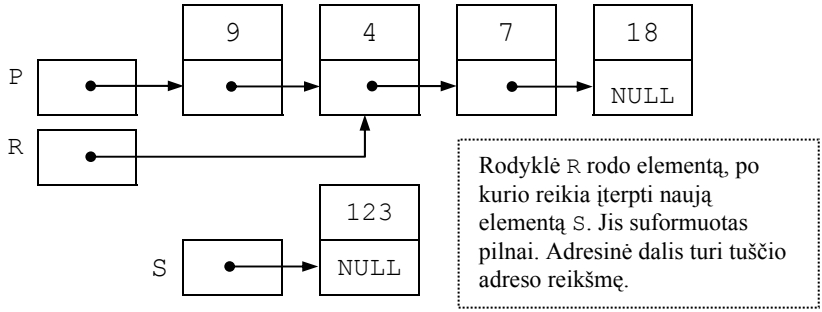
Sąrašo papildymas naujais elementais galimas:

- sąrašo pradžioje;
- sąrašo pabaigoje;
- sąrašo viduje.

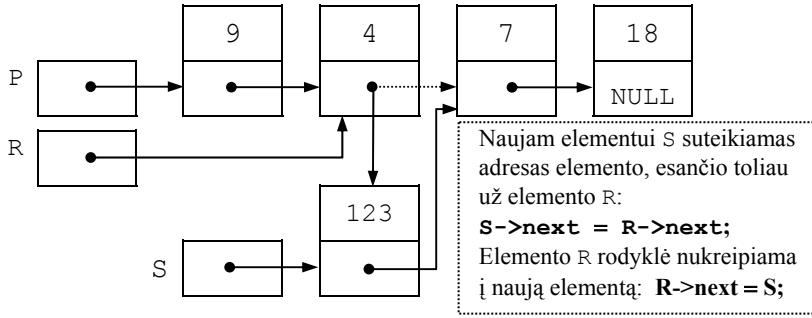
Naujo elemento prijungimas sąrašo pradžioje sutampa su sąrašo formavimo ciklo veiksmais.

Naujo elemento prijungimas sąrašo gale yra tapatus elemento įterpimui sąrašo viduje po nurodyto elemento. Veiksmų schema parodyta 7.8 pav. ir

7.9 pav. Patikrinkite, ar tikrai tinka tie patys veiksmai, kai `R` rodo paskutinį sąrašo elementą (papildymas gale).



7.8 pav. Naują elementą `S` reikia įterpti už elemento `R`



7.9 pav. Naujo elemento įterpimas

Atskiras sąrašo papildymo atvejis yra tada, kai sąrašas visiškai tuščias. Tuomet pakanka sąrašo pradžios rodyklei P suteikti naujo elemento adresą:

```
P = S;
```

Įterpimas prieš nurodytą elementą tokio tipo sąrašuose negalimas, nes elementai susieti rodyklėmis viena kryptimi. Galimas tik vienas atvejis: įterpti prieš pirmąjį, kas tolygu sąrašą papildyti pradžioje nauju elementu. Įterpimą "prieš" galima pakeisti įterpimu "po". Tam reikia surasti elementą, esantį prieš mus dominantį. Pavyzdžiui, reikia rasti elementą, esantį prieš didžiausią:

```
sar *R,                    // Didžiausios reikšmės elementas
    *R1,                   // Elementas, esantis prieš didžiausią
    *T,*T1;                // Papildomos rodyklės sąrašo peržiūrai
```

```
T1 = P;  T = P;
R1 = P;   R = P;
while(T != NULL) {
  if(T->sk > R->sk) { R1 = T1; R = T; }
  T1 = T;
  T = T->next; }
```

Peržiūrėjus sąrašą iki galo, rodyklė `R` rodys elementą su didžiausia reikšme, o rodyklė `R1` rodys elementą, esantį prieš didžiausią. Jos abi rodys pirmąjį elementą, kai jo reikšmė sąraše didžiausia. Tai reiškia, kad veiksmuose su sąrašais visuomet būtina nagrinėti visas galimas situacijas.

Įterpimą "prieš" galima atlikti paprasčiau. Jeigu darbo su sąrašu veiksmai neprieštarauja elementų reikšmių keitimui vietomis, tuomet siūloma tokia įterpimo veiksmų seka:

- Turime surastą elementą `R`, prieš kurį reikia įterpti elementą `S`;
- Sukeičiamos `R` ir `S` elementų reikšmės (čia `int k;`):

```
k = R->sk;
R->sk = S->sk;
S->sk = k;
```

- Įterpiamas elementas `S` po elemento `R`:

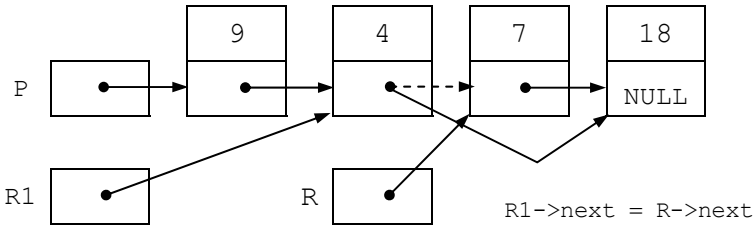
```
S->next = R->next;
R->next = S;
```

7.6. Tiesinio sąrašo elementų šalinimas

Tai priešingas veiksmas įterpimui, todėl veiksmai analogiški. Šalinant nurodytą elementą, būtina nagrinėti jo padėtį sąraše. Jeigu tai pradžios elementas, jis pašalinamas sąrašo pradžios rodyklę P perkeliant į sekantį elementą:

```
P = P->next;          // Šalinamas elementas R
delete(R);            // yra pirmas, t.y. R = P
```

Klasikinis elemento pašalinimo iš sąrašo atvejis yra, kai turime rodyklę į šalinamąjį elementą `R` ir į prieš jį esantį R1 (7.10 pav.).



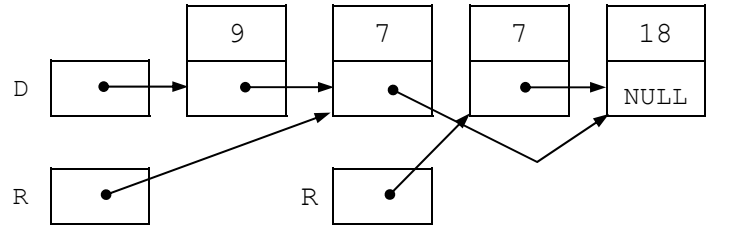
7.10 pav. Elemento pašalinimas

Šalinant elementus būtina atlaisvinti atmintį, kurią jie užima. Tam naudojama funkcija **`delete`**.

Labiau patyręs programuotojas ras lakoniškesnių priemonių veiksmams atlikti. Kaip vieną iš jų galima pasiūlyti nurodyto elemento `R` šalinimui (7.11 pav.):

```
*R = *R->next;
```

Čia yra tolesnio elemento turinio perkėlimas į nurodytą. Tuo sunaikinamas elemento R turinys. Sąraše turime du vienodus elementus, kurių tik vienas prieinamas.



7.11 pav. Sąrašas po elemento pašalinimo

Pastaba: Šio veiksmo negalima taikyti, kai `R` rodo paskutinį elementą, nes toliau jau nėra kitų elementų. Šiam atvejui būtina atskira veiksmų grupė. Tokios elementų šalinimo procedūros negalima taikyti sudėtingose sąrašinėse struktūrose, kur duomenų elementų adresai naudojami naujoms sąrašų sekoms formuoti. Duomenys negali keisti savo pradinio adreso.

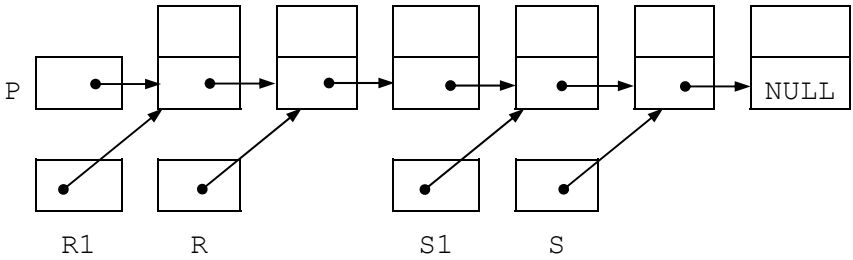
7.7. Tiesinio sąrašo elementų tvarkymas

Tipiškiausias sąrašo elementų tvarkymo būdas yra rikiavimas. Paprasčiausias atvejis yra, kai leidžiama keisti elementuose duomenis vietomis (analogija su masyvais). Toks būdas priimtinas, kai duomenų

struktūra elemente nesudėtinga arba duomenų apimtis nedidelė. Visais atvejais tikslinga elemente turėti tik vieną lauką duomenims nurodyti, o duomenų pilna hierarchinė struktūra yra aprašoma atskirai.

```
void DinSar::Rikiavimas() {
  sar *R = P, *R1;
  int k;
  while(R != NULL) {
    R1 = R->next;
    while(R1 != NULL) {
      if (R1->sk > R->sk) {
        k = R->sk;                    // Reikšmių sukeitimas
        R->sk = R1->sk;
        R1->sk = k; }
      R1 = R1->next; }
    R = R->next;     }
}
```

Programose dažna situacija, kai negalima keisti sąrašo elementų duomenis vietomis. Reikia sukeisti elementų seką sąraše, t.y. keičiamos nuorodos. Norint sukeisti vietomis sąraše du elementus, reikia turėti rodykles ne tik į keičiamus elementus, bet ir į prieš juos esančius. Čia reikės pasirūpinti keičiamų elementų kaimynais "prieš" ir "po" (7.12 pav.).



7.12 pav. `R` ir `S` rodo keičiamus vietomis elementus

Reikia sukeisti vietomis elementus `R` ir `S`. Tai galima padaryti taip:

- `R1` elemento kaimynu "po" padaromas elementas `S`:
  ```
  R1->next = S;
  ```
- `S1` elemento kaimynu "po" turi būti elementas `R`:
  ```
  S1->next = R;
  ```
- Atsimename elemento "po" `R` adresą:

```
pap = R->next;
```

- Elementas `R` turi rodyti į elementą "po" `S`:

```
R->next = S->next;
```

- Elementas `S` turi rodyti į elementą "po" `R`:

```
S->next = pap;
```

Reikia neužmiršti patikrinti keičiamų elementų sąraše padėtį (pradžioje, gale ar viduje). Gaunama sudėtinga veiksmų seka. Žymiai paprasčiau suformuoti naują tvarkingą sąrašą. Tai galima padaryti taip:

- rasti duotame sąraše elementą pagal duotą rikiavimo raktą;
- pašalinti surastą elementą iš duoto sąrašo;
- prijungti elementą prie naujo sąrašo.

```
void DinSar::Surikiavimas() {
  sar *T = P,                 // T  senojo sąrašo pradžia
  *D, *D1, *S, *S1;
  P = NULL;                   // Naujas sąrašas tuščias

  while(T) {
    D = T;                    // Gretimi du elementai sąrašo peržiūrai
    D1 = T;
    S = T;
    S1 = T;                   // Didžiausias  S  ir prieš jį esantis  S1
    while(D) {                // Didžiausio paieška
      if (D->sk > S->sk)
        { S = D; S1 = D1; }
      D1 = D;
      D = D->next; }
    if (S == T)  T = T->next;    // Šalinimas
    else  S1->next = S->next;
    S->next = P;                 // Prijungimas prie naujo sąrašo
    P = S;  }                    // Naujo sąrašo pradžia
}
```

Šiame pavyzdyje nebuvo kuriami nauji elementai. Jie buvo sujungiami nauja seka, pasinaudojant jau žinomomis operacijomis. Tą patį rezultatą galima gauti trumpesniu keliu. Siūlome panagrinėti "burbuliuko" būdu rikiuojančią funkciją:

```
void DinSar::RikiavBurb() {
  sar **T, *R;
```

```
  int flag = 1;
  while(flag) {
    flag = 0;
    T = &P;
    while((*T)->next) {
      if ((*T)->sk > (*T)->next->sk) {
        R = *T;
        *T = R->next;
        R->next = (*T)->next;
        (*T)->next = R;
        flag = 1; }
      T = &(*T)->next;  } }
}
```

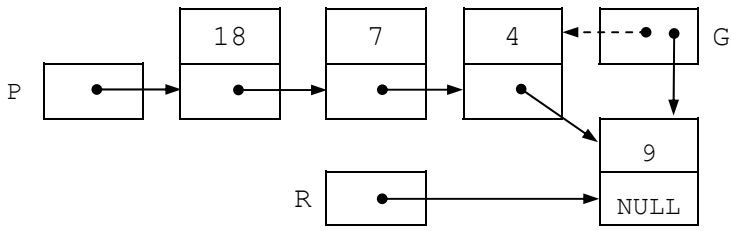
Tvarkymo ir kiti veiksmai vienkrypčiuose sąrašuose yra sudėtingi, reikalauja papildomų darbo sąnaudų. Žymiai yra patogesni dvikrypčiai sąrašai, kur kiekvienas elementas turi rodykles į kaimynus "prieš" ir "po". Realiai vienkrypčiai sąrašai naudojami su tam tikrais apribojimais: stekai, eilės, dekai.

7.8. Eilės formavimas

Visuose pateiktuose pavyzdžiuose sąrašas buvo formuojamas naujus elementus prijungiant prie pradžios, t.y. steko principu. Ne visuomet tai patogu. Galima sąrašą formuoti jų pateikimo eilės tvarka (7.13 pav.). Formavimo procesas skaldomas į du etapus:

- Pirmojo elemento sukūrimas. Rodyklės `P` ir `G` rodo pirmąjį elementą
- Kitų elementų prijungimas gale. Reikalinga sąrašo pabaigos rodyklė `G`. Ji rodo elementą, po kurio reikia padėti naują elementą.

Papildykite klasę tam skirtais metodais **`FormuotiEile`** ir **`ElementasEile`** ir juos išbandykite. Atkreipkite dėmesį į naujo elemento sukūrimo ir prijungimo skirtumus steko ir eilės atvejais.



7.13 pav. Eilės formavimas

```
void DinSar::FormuotiEile(FILE *F) {
  sar *D = NULL, *G;
  int k;

  if (!feof(F)) {                  // Pirmas elementas
    fscanf(F, "%i", &k);
    ElementasEile(&D, k);
    G = D; }                       // Galo rodyklė G
  while(!feof(F)) {                // Kiti elementai
    fscanf(F, "%i", &k);
    ElementasEile(&G, k); }        // Prijungimas gale
  P = D;
}
```

```
void DinSar::ElementasEile(sar **P, int Sk) {
  sar *R;
  R       = new sar;               // Suformuojamas elementas
  R->sk   = Sk;
  R->next = NULL;
  if (*P) {                        // Elementas prijungiamas gale
    (*P)->next = R;
    *P = R; }
  else *P = R;                     // arba jis dar pirmas sąraše
}
```

7.9. Stekų tvarkymas

Stekai yra paprasčiausi ir lengviausiai tvarkomi tiesiniai dinaminiai sąrašai. Juose nauji elementai įterpiami sąrašo pradžioje, o vartojimui prieinamas tik pirmas elementas, kuris po to šalinamas. Deklaruojant stekus, pakanka aprašyti jų elementų struktūras ir rodykles į stekų pradžias. Darbui su steku reikalingos dvi funkcijos: naujo elemento prijungimo prie sąrašo pradžios ir pirmojo elemento pašalinimo iš sąrašo (prieš tai tas elementas yra panaudojamas veiksmuose).

7.1 pratimas. Išsiaiškinkite pavyzdyje aprašytų simbolių steko tvarkymo funkcijų sudarymo principus ir patikrinkite demonstracinės programos darbą. Programoje simboliai yra atrenkami ir siunčiami į steką iš joje deklaruotos ir inicializuotos simbolių eilutės.

```
//                      Veiksmai su steku
#include <iostream.h>

struct stack { char d;            // Steko elementų tipas
               struct stack *next; };

class stekas {
  public:
    stack *first;                 // Steko realizacija (rodyklė)
    stekas() { first = NULL; }    // Konstruktorius
   ~stekas() {};                  // Destruktorius
    stack *Get(){ return first; };
    void Push(char);              // Steko papildymas
    void Ekranas(stack *);        // Steko peržiūra
    char Pop();                   // Steko skaitymas
};
//                    Pagrindinė programa
main() {                  // Tvarkoma eilutė ir rodyklė į ją
  char prad[] = "abcdefgh",  *p = prad;
  stack *d;                       // Darbinis  kintamasis
  stekas A;                       // Objekto aprašymas

                          // Tvarkomos eilutės parodymas  ekrane
  cout << "Siunciami i steka simboliai: " << p << endl;
  while(*p)  A.Push(*p++);        // Steko užpildymas
  cout << "Steko patikrinimas:\n";
  d = A.Get();
  A.Ekranas(d);           // Steko turinio  "abcdefgh"  parodymas
  cout << "Skaitymas po viena simboli:\n";
  while(A.first)  cout << A.Pop() << endl;
                                  // Steko išvalymo kontrolė
  if (!A.first) cout << "Tuscias stekas\n";
  A.~stekas();
}
//                    Steko papildymas
void stekas::Push(char e) {
  stack *newe;
  newe    = new stack;    // Atminties skyrimas
  newe->d = e;            // Naujo elemento reikšmė
  newe->next = first;     // Senos steko dalies prijungimas
       first = newe;      // Naujas steko adresas
}
```

```
//              Rekursyvus steko turinio patikrinimas
void stekas::Ekranas(stack *p) {
  if (p) {                  // Rekursijos nutraukimo sąlyga
    cout << p->d;           // Reikšmės parodymas ekrane
    Ekranas(p->next);} // Rekursyvus kreipinys į tolimesnį elementą
  else cout << endl;
}
//                Elemento pašalinimas iš steko
char stekas::Pop() {
  stack *old= first;        // Adreso išsaugojimas
  char e = first->d;        // Reikšmės išsaugojimas
  first = first->next;      // Pirmo elemento šalinimas
  delete old;               // Atminties išlaisvinimas
  return e;                 // Pašalinto elemento reikšmė
}
```

Rekursyvi funkcija **`Ekranas`** stekui yra perteklinė. Joje demonstruojama, kad rekursyvaus tipo dinaminių struktūrų elementų perrinkimą galima glaustai aprašyti rekursyviomis funkcijomis. Elementų perrinkimui taip pat galima vartoti funkcijos viduje deklaruojamą lokalią rodyklę. Naudojant šią perrinkimo techniką, galima sudaryti tokią funkcijos **`Ekranas`** modifikaciją:

```
//              Ciklinis steko turinio patikrinimas
void stekas::Ekranas1(stack *p) {
  stack *temp = p;          // Pagalbinė rodyklė
  while(temp) {             // Steko perrinkimo ciklas
    cout << temp->d;        // Reikšmės parodymas ekrane
    temp = temp->next; }    // Rodyklė į tolimesnį elementą
  cout << endl;
}
```

Standartinės stekų valdymo procedūros yra labai paprastos ir greitos, todėl stekai plačiai vartojami laikinam duomenų saugojimui tiek taikomosiose, tiek sisteminėse programose. Stekus galima realizuoti ne tik dinaminėmis struktūromis, bet ir kitais būdais: aparatūros pagalba, vartojant specialius indeksų registrus. Pavyzdžiui, steko registro pagalba tvarkomą steko atminties segmentą kompiliatoriai vartoja lokaliems kintamiesiems saugoti ir informaciniams ryšiams tarp funkcijų palaikyti.

7.10. Eilės ir dekai

Eilės (7.14 pav.) nuo stekų skiriasi tuo, kad į jas duomenys yra siunčiami ir iš jų atrenkami priešinguose sąrašo galuose. Pertvarkant steką į eilę, galima vieną jo tvarkymo procedūrą, pavyzdžiui **`Pop`**, palikti, o antrąją reikia perrašyti panaudojant rekursyvią sąrašo peržiūrą eilės pabaigos paieškai. Suradus pabaigą, eilė papildoma tokiu pat būdu, kaip ir paprastas sąrašas. Taip sudarytos funkcijos **`Add`** pavyzdys yra pateiktas 7.2 pratime.

7.2 pratimas. Čia pateikta eilės papildymo funkcija, kuri pritaikyta eilėms sudarytoms iš 7.1 pratime aprašytos struktūros `stack` elementų. Skaitytojui siūloma savarankiškai sudaryti ciklinį šios funkcijos variantą.

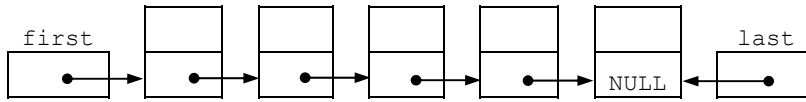
```
//                          Eilės papildymas
void stekas::Add(stack *p, char e) {
  if (!p) {                        // Jeigu eilė tuščia
    p = new stack;
    p->next = NULL;
    p->d = e;
    first = p; }                   // Eilėje atsiranda 1 elementas
  else
    if (!p->next) {                // Rekursijos pabaigos sąlyga
      p->next = new stack;         // Naujo elemento prijungimas
      p->next->next = NULL;
      p->next->d = e; }
    else Add(p->next, e);          // Rekursija
}
```

Norint iš eilės padaryti deką, reikia sudaryti dar vieną funkciją, kuri leistų pašalinti galinius eilės elementus. Rekursyvios galinio eilės elemento šalinimo funkcijos realizacija anksčiau aprašytai struktūrai `stack` atrodo taip:

```
//                       Šalinimas iš eilės galo
char stekas::Del(stack *p) {
  char s;                  // Šalinamasis elementas
  if (!p->next) {          // Jeigu eilėje tik 1 elementas
    s = p->d;
    delete first;          // Šalinamas pirmas eilės elementas
    first = NULL;
    return s; }
```

```
  else
    if (!p->next->next){
      s = p->next->d;
      delete p->next;   // Šalinamas priešpaskutinis eilės elementas
      p->next = NULL;
      return s; }
                        // Jeigu eilėje daugiau kaip vienas elementas
    else Del(p->next); // Rekursija
}
```

Eilės pabaigos paieškai vartoti rekursyvų arba ciklinį jos elementų perrinkimą yra neracionalu. Toks paieškos būdas gali būti pateisinamas tiktai trumpose eilėse. Tvarkant ilgas eiles, jų elementų perrinkimo patartina vengti. Tai galima padaryti, papildant eilę galinio elemento rodykle, kurios reikšmė būtų vartojama ir keičiama papildant eilę naujais elementais. Tokios eilės išoriniam aprašymui rekomenduojama sudaryti dviejų rodyklių struktūrą ir ją tvarkyti modifikuotomis steko tvarkymo funkcijomis. Kaip tai yra daroma, iliustruojama 7.3 pratime.



7.14 pav. Eilę nusako pradžios ir pabaigos rodyklės

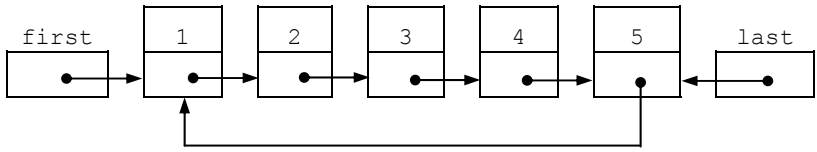
7.3 pratimas. Naudodami pateiktus simbolių eilės su dviem rodyklėm struktūros ir jos tvarkymo funkcijų aprašymus, sudarykite demonstracinę programą, kuri leistų patikrinti šių funkcijų darbą.

```
struct stack {                  // Sąrašo elementų tipas
  char d;
  struct stack *next; };

struct list {                   // Eilė su dviem išorinėm rodyklėm
  stack *first;                 // Pradžios rodyklė
  stack *last; };               // Pabaigos rodyklė
//                      Klasės aprašas
class eile {
  public:
    list que;                   // Eilės realizacija
    eile() {                    // Konstruktorius
      que.first = NULL;
      que.last  = NULL; };
```

```
   ~eile() {};                  // Destruktorius
    int Add(char);              // Eilės papildymas gale
    char Pop();                 // Pradinio elemento pašalinimas
};
//                Eilės tvarkymo funkcijų realizacijos
//                        Eilės papildymas gale
int eile::Add(char e) {
                  // F-jos Add reikšmė 1 praneša, kad papildyta sėkmingai
  stack *newe;
  if (!(newe = new stack))  return 0; // Ar yra atmintyje vietos?
  newe->d = e;                 // Naujas elementas
  newe->next = NULL;           // Pabaigos žymė
  if (!que.last) {             // Jeigu eilė tuščia
    que.last  = newe;          // Naujas pabaigos adresas
    que.first = newe; }        // Naujas pradžios adresas
  else {
    (que.last)->next = newe; // Elemento prijungimas
    que.last = newe;  }        // Naujas pabaigos adresas
  return 1;                    // Pranešimas apie sėkmingą pabaigą
}
//                    Pradinio elemento pašalinimas
char eile::Pop() {
  stack *old= que.first;           // Adreso išsaugojimas
  if (!(que.first)) return '\0'; // Pranešimas apie tuščią eilutę
  char e = (que.first)->d;         // Reikšmės išsaugojimas
  que.first = (que.first)->next; // Pirmo elemento naikinimas
  delete old;                      // Atminties išlaisvinimas
  return e;                        // Reikšmės grąžinimas
}
```

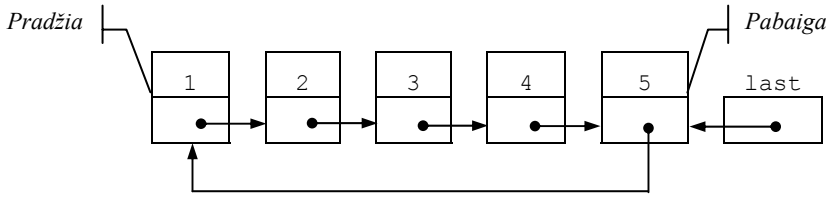
7.11. Žiediniai sąrašai

Tiesinio vienkrypčio sąrašo galiniame elemente įrašius pirmojo elemento adresą, gausime žiedinį sąrašą: **`last->next = first;`** (7.15 pav.).



7.15 pav. Žiedinė sąrašo struktūra

Toliau veiksmuose patogu turėti tik vieną išorinę rodyklę `last`, kuri saugos žiedo galinio (paskutinio) elemento adresą, o galinio elemento ryšio dalyje bus saugoma žiedo pradinio elemento adresas (7.16 pav.). Tą rodyklę (`last`) toliau vadinsime **žiedo rodykle**. *Pavadinimai 'pradžia' ir 'galas' žiede yra sąlyginiai*. Naudojant šiuos pavadinimus bus lengviau aiškintis veiksmus žiede.



7.16 pav. Žiedinis sąrašas

Žiediniai sąrašai gali būti vartojami eilių realizavimui sprendžiant tokius uždavinius, kuriuose reikalinga daugelį kartų peržiūrėti tą patį duomenų rinkinį. Žiedams yra taikomos tokios standartinės tvarkymo operacijos:

- tuščio žiedo inicializavimas;
- naujo elemento įterpimas žiedo pradžioje;
- papildymas nauju elementu pabaigoje;
- pradinio elemento pašalinimas;
- paieška ir pašalinimas.

Tuščio žiedo, kuris žymimas tuščia rodykle `NULL`, inicializavimas yra toks pats, kaip ir tiesinio sąrašo. Tuo tarpu, kitų operacijų realizavimas skiriasi. Pavyzdžiui, papildant sąrašą naujais elementais, tenka taikyti skirtingas procedūras tuščiam ir netuščiam sąrašui.

Paieškos ir pašalinimo operacijos metu turi būti randamas ir pašalinamas elementas su duotąja rakto reikšme. Paieška yra vykdoma nuosekliai perrenkant žiedo elementus. Šalinant reikia išskirti du atvejus: kai yra šalinamas elementas, į kurį rodo žiedo išorinė rodyklė, yra 'paskutinis' ir kai šalinamas elementas žiede yra vienintelis. Pirmuoju atveju, turi būti keičiama žiedo išorinės rodyklės reikšmė, o antruoju atveju, žiedo išorinės rodyklės reikšmei priskiriama `NULL`. Atvejį, kai reikia šalinti žiede esantį elementą, į kurį nerodo žiedo rodyklė, visuomet galima pakeisti į atvejį, kai šalinamas elementas, kurio adresą saugo žiedo rodyklė.

7.4 pratimas. Sudarykite demonstracinę programą, kuri leistų patikrinti veiksmus su žiediniu sąrašu. Naudokite klasę **`ziedas`** ir jos tvarkymo metodų aprašymus.

```
struct stack { char d;       // Žiedo elementų tipas
               struct stack *next; };

class ziedas {
public:
  stack *last;               // Žiedo realizacija (rodyklė)
  ziedas() { last = NULL; };// Konstruktorius
 ~ziedas() {};               // Destruktorius
  int Ins(char);             // Įterpimas žiedo pradžioje
  int Add(char);             // Įterpimas žiedo pabaigoje
  char Del();                // Pirmojo žiedo elemento pašalinimas
  char DelSerch(char);       // Elemento paieška ir šalinimas
  };
//              Žiedo tvarkymo funkcijų realizavimas
//                    Įterpimas žiedo 'pradžioje'
// Funkcijos reikšmė 1 praneša, kad papildyta sėkmingai
int ziedas::Ins(char e) {
  stack *newe;
  if (!(newe = new stack))    // Ar yra atmintyje vietos?
    return 0;
  newe->d = e;                // Naujas duomenų elementas
  if (last) {                 // Netuščio žiedo sąlyga
    newe->next = last->next; // Pirmo elem. adreso perdavimas
    last->next = newe;  }     // Naujas pradžios adresas
  else  last = newe->next = newe;   // Tuščio žiedo papildymas
  return 1;                   // Pranešimas apie sėkmingą pabaigą
}
//                    Žiedo papildymas 'pabaigoje'
// Funkcijos reikšmė 1 praneša, kad papildyta sėkmingai
int ziedas::Add( char e ){
  stack *newe;
  if (!(newe = new stack))    // Ar yra atmintyje vietos?
    return 0;
  newe->d = e;                // Naujas duomenų elementas
  if (last) {                 // Netuščio žiedo sąlyga
    newe->next = last->next; // Pirmo elem. adreso perdavimas
    last->next = newe;        // Elemento prijungimas pabaigoje
    last = newe;  }           // Naujas pradžios adresas
```

```
  else  last = newe->next = newe;   // Tuščio žiedo papildymas
  return 1;                         // Pranešimas apie sėkmingą pabaigą
}
//              'Pirmojo' žiedo elemento pašalinimas
char ziedas::Del() {
  stack *temp;                      // Pagalbiniai kintamieji
  char e;
  if (!last)  return '\0';          // Pranesimas apie klaida
  if (last == last->next) {         // Žiede vienas elementas
    temp = last;                    // Šalinamo elemento adresas
    last = NULL;                    // Nauja išorinė rodyklė
  }
  else {                            // Žiede yra keletas elementų
    temp = last->next;              // Šalinamo elemento adresas
    last->next = last->next->next;  // Šalinimas iš žiedo
  }
  e = temp->d;                      // Reikšmės išsaugojimas
  delete temp;                      // Atminties išlaisvinimas
  return e;                         // Reikšmės grąžinimas
}
//              Elemento paieška ir pašalinimas žiede
char ziedas::DelSerch(char e) {
  stack *temp = last;               // Pagalbiniai kintamieji
  stack *t;
  if (!last) return '\0';           // Pranešimas apie klaidą
  while (((temp->next) != last) &&
         ((temp->next->d) !=e))
    temp = temp->next;              // Elementų perrinkimas
  if ((temp->next->d) == e)         // Jeigu elem. buvo surastas
    if (last == temp->next)         // Galinis elementas
      if (temp == last) {           // Jeigu žiede 1 elementas
        last = NULL;                // Tuščias žiedas
        delete temp;  }
      else {
        temp->next = last->next;    // Elemento šalinimas
        delete last;
        last = temp;  }             // Nauja išorinė rodyklė
    else {
      t = temp->next;
      temp->next = temp->next->next;
      delete t;  }
}
```

7.12. Surikiuoto sąrašo papildymas žiede

Tiesinių sąrašų elementų rikiavimo tvarką galima įvertinti jų formavimo funkcijose ir jas sudaryti taip, kad, papildant sąrašą naujais elementais, jų rikiavimo tvarka išliktų. Tokios funkcijos yra sudaromos iš dviejų dalių. Pirmojoje dalyje yra ieškoma sąrašo vieta, kurioje reikia įterpti naują elementą, o antrojoje – aprašoma įterpimo procedūra. Jeigu simbolių sąrašo elementai yra saugomi alfabeto tvarka, naujas elementas turi būti rašomas prieš pirmąjį sąrašo elementą, kurio kodas yra didesnis už papildančio simbolio kodą arba, jei tokio elemento nėra, sąrašo gale. Surikiuoto sąrašo papildymo funkcijos realizacija žiede:

```
void ziedas::Terpti(char e) {
  stack *t, *temp;     // Pagalbiniai kintamieji
  if (!last) {         // Jeigu žiedas tuščias
    last = new stack;
    last->d = e;
    last->next = last; }
  else                 // Jeigu įterpiama prieš pirmą žiedo elementą
    if (last->next->d > e) {
      t = new stack;
      t->d = e;
      t->next = last->next;
      last->next = t; }
    else               // Jeigu įterpiama po paskutinio žiedo elemento
      if (last->d < e) {
        t = new stack;
        t->d = e;
        t->next = last->next;
        last->next = t;
        last = t; }
      else {           // Įterpimo vietos paieška ir įterpimas
        temp = last->next;
        while ((temp != last) && ((temp->next->d) < e))
          temp = temp->next;
        t = new stack;
        t->d = e;
        t->next = temp->next;
        temp->next = t;  }
}
```

7.13. Sąrašai su neapibrėžto tipo elementais

Sąrašai, kurių elementuose tiesiogiai saugomos jų reikšmės, nėra universalūs, nes jų tvarkymo funkcijų realizacijos priklauso nuo elementų tipo. Norint sudaryti universalius sąrašus su įvairių tipų elementams tinkančiomis tvarkymo funkcijomis, reikia sąraše saugoti ne pačias elementų reikšmes, o rodykles į jas. Geriausiai šiam tikslui tinka neapibrėžto (`void`) tipo rodyklės, kurios yra suderinamos su visais kitais rodyklių tipais:

```
struct stack {
  void *d;
  struct list *next;  };
```

Sąrašų su neapibrėžto tipo duomenų elementų rodyklėmis tvarkymo veiksmus reikia atskirti nuo elementams skiriamos atminties tvarkymo veiksmų ir veiksmų, kurie susiję su sąraše saugomų duomenų analize, jų interpretavimu. Dauguma sąrašų tvarkymo veiksmų yra universalūs ir juos galima aprašyti tokiomis funkcijomis, kurios tinka visiems rodyklių nurodomiems elementų tipams. Tuo tarpu, su duomenų interpretavimu susiję veiksmai paprastai būna specializuoti, priklauso nuo sąrašo elementų reikšmių tipo.

Rodyklių sąrašų formavimas yra labai galinga duomenų tvarkymo priemonė, kuri įvairioms duomenų saugojimo struktūroms gali suteikti naujas savybes. Pavyzdžiui, tokio sąrašo sudarymas gali pakeisti masyvo rikiavimą. Be to, vienam masyvui galima sudaryti kelis rodyklių sąrašus, aprašančius skirtingus jame saugomų duomenų perrinkimo būdus.

Aprašant neapibrėžto tipo rodyklių sąrašų nurodomų duomenų interpretavimo operacijas, reikia apibrėžti šių rodyklių interpretavimo būdą. Interpretavimo būdas yra apibrėžiamas tokia struktūra:

(<`Duomenų tipas`**>*) <**`Elemento duomenų rodyklė`**>**

7.5 pratimas. Patikrinkite pratime pateiktos programos darbą. Išsiaiškinkite joje aprašyto universalaus steko tvarkymo funkcijų sudarymo principus ir šių funkcijų pritaikymą simbolių stekui tvarkyti. Programoje taip pat iliustruojamas specializuotos funkcijos **`Show`** sudarymas, kuri gali perduoti į ekraną tik simbolių steko reikšmes.

```
//              Sarašas su neapibrėžto tipo elementais
#include <iostream.h>
char line[] = "Sula";            // Simbolių masyvas
```

```
struct stack {                    // Universalus sąrašo elementas
  void *d;                        // Duomenų rodyklė
  struct stack *next;  };
class stekas {
  public:
    stack *first;                 // Steko realizacija (rodyklė)
    stekas() { first = NULL; };// Konstruktorius
   ~stekas() {};                  // Destruktorius
    stack *Get() { return first; };
//              Universalios tvarkymo funkcijos
    void Push(void *);            // Papildymas
    void *Pop();                  // Elemento pašalinimas
    void Show(stack *);           // Steko turinio parodymas ekrane
};
//                    Pagrindinė programa
main() {
  stack *d;                     // Papildomas kintamasis
  stekas A;                     // Objekto aprašymas
  int i = 0;                    // Simbolių masyvo indeksas
  cout << "Tvarkoma eilute: " << line << endl;
  while(line[i])                // Simbolių masyvo pabaigos sąlyga
    A.Push(&line[i++]);         // Rodyklių steko sudarymas
  cout << "Steko rodomi simboliai: " << endl;
  d = A.Get();
  A.Show(d);
  cout << "Trinama rodykle i pirma simboli ";
  cout << *((char *) A.Pop()) << endl;
  cout << "Liko rodykles i siuos simbolius: \n";
  d = A.Get();
  A.Show( d );
}
//              Sąrašo tvarkymo funkcijų realizacijos
//                        Steko papildymas
void stekas::Push(void *e) {
  stack *newe;             // Papildomas kintamasis
  newe = new stack;        // Atminties skyrimas
  newe->d = e;             // Naujo elemento duomenų rodyklė
  newe->next = first;      // Senos steko dalies prijungimas
  first = newe;            // Naujas steko adresas
}
```

```
//                  Elemento pašalinimas iš steko
void * stekas::Pop() {
  stack *old = first;    // Adreso išsaugojimas
  void *e = first->d;    // Reikšmės rodyklės išsaugojimas
  first = first->next;   // Pirmo elemento šalinimas
  delete old;            // Atminties išlaisvinimas
  return e;
}
//                  Rekursyvus steko patikrinimas
void stekas::Show(stack *p) {
  if (p) {              // Interpretavimas ir išvedimas į ekraną
    cout << *(char *)(p->d);
    Show(p->next);  }   // Rekursyvus kreipinys į tolimesnį elementą
  else  cout << endl;
}
```

7.14. Sąrašų su neapibrėžto tipo elementais panaudojimo pavyzdžiai

1 pavyzdys. Tekstiniame faile "`Stud.dat`" turime studentų sąrašą: pavardė, vardas, pažymių vidurkis. Reikia suformuoti du studentų sąrašus: vieną - surikiuotą pagal abėcėlę ir antrą - surikiuotą pažymių vidurkio mažėjimo tvarka.

Šiai užduočiai atlikti yra daug būdų. Paprasčiausias yra suformuoti du nepriklausomus sąrašus, kurie surikiuojami pagal nurodytą raktą. Kitas gali būti toks: duomenys surašomi į masyvą ir suformuojami du rodyklių į duomenis sąrašai pagal nurodytą raktą. Vietoje masyvo galima panaudoti dinaminį sąrašą.

Siūlome vienu sąrašu saugoti duomenis ir rezultatus. Duomenys surašomi į sąrašą ir sutvarkomi pagal abėcėlę. Sąraše rezervuojame dar vieną lauką nuorodai į duomenis. Tas nuorodas surikiuojame pagal studentų mokymosi vidurkį.

Failo "`Stud.dat`" pavyzdys.

```
Lapinas   Baisus      4.5
Vilkas    Pilkas      6.85
Baisus    Katinas     3.6
Katinas   Didysis     9.5
```

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

typedef struct studentas { char  pavard[10];
                           char  vardas[10];
                           float vidur;  };

typedef struct sar { studentas  *st;
                     studentas  *vd;
                     sar  *next;  };
//                   Klasės aprašymas
class student {
  sar *P;                          // Sąrašo pradžia
  public:
    student() { P = NULL; };       // Konstruktorius
    void Formuoti(FILE *);         // Formuoja sąrašą
    void Spausdinti(int);          // Spausdina sąrašą
    void Tvarkymas(int);           // Tvarkymas
};
//                   Pagrindinė programa
void main() {
  FILE *D;
  student A;
  clrscr();
  D = fopen("stud.dat", "r");
  if (D == NULL) {
    cout << "Failo Stud.dat nepavyko atidaryti!";
    exit (1);  }
  A.Formuoti(D);
  fclose(D);
  A.Spausdinti(0);     // Nesurikiuotas
  A.Tvarkymas(1);      // Rikiavimas pagal abėcėlę
  A.Tvarkymas(0);      // Rikiavimas pagal vidurkį
  A.Spausdinti(1);     // Abėcėlinis sąrašas
  A.Spausdinti(0);     // Sąrašas pagal vidurkį
}

//                   Studentų sąrašo formavimas
void student::Formuoti(FILE *F) {
  sar *D = NULL;
```

```
  while (!feof(F)) {
    D = new sar;
    D->st = new studentas;
    D->vd = D->st;
    fscanf(F, "%s%s%f", D->st->pavard, D->st->vardas,
                                       &D->st->vidur);
    D->next = P;
    P = D;  }
}
```

Studentų sąrašo spausdinimas

```
void student::Spausdinti(int kaip) {
  sar *D = P;
  while(D) {
    if (kaip)  cout << D->st->pavard << " "
                    << D->st->vardas << " "
                    << D->st->vidur  << "\n";
    else  cout << D->vd->pavard << " "
               << D->vd->vardas << " "
               << D->vd->vidur << "\n";
    D = D->next;  }
  cout << "--------------------\n";
}
```

Studentų sąrašo rikiavimas

```
void student::Tvarkymas(int kaip) {
// kaip = 1 tvarko pagal abėcėlę st rodyklę
// kaip = 0 tvarko pagal vidurkį vd rodyklę
  sar *R = P, *R1;
  studentas *k;
  while(R != NULL ) {
    R1 = R->next;
    while(R1 != NULL) {
      if (kaip) {
        if (strcmp(R1->st->pavard, R->st->pavard) < 0)
          { k = R->st;  R->st = R1->st;  R1->st = k; }}
      else if (R1->vd->vidur > R->vd->vidur)
        { k = R->vd;  R->vd = R1->vd;  R1->vd = k; }
      R1 = R1->next; }
    R = R->next; }
}
```

---

2 pavyzdys. Tekstiniame faile "`Stud.dat`" turime studentų sąrašą: pavardė, vardas, pažymių vidurkis. Yra trys kompiuterių klasės, kurias prižiūri ir tvarko studentai. Reikia suformuoti studentų darbo klasėse sąrašą savaitei. Kasdieną paskiriami trys studentai sąrašo eilės tvarka.

Šiam uždaviniui spręsti patogiausias yra žiedinis sąrašas. Modifikuojame ankstesnio pavyzdžio programą.

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

typedef struct studentas { char  pavard[10];
                           char  vardas[10];
                           float vidur; };
typedef struct  sar { studentas  *st;
                      sar *next; };
//                    Klasės aprašymas
class student {
  sar *P;                          // Sąrašo pradžia
  public:
    student() { P = NULL; };       // Konstruktorius
    void Formuoti(FILE *);         // Formuoja sąrašą
    void Spausdinti();             // Spausdina sąrašą
    void Darbai(int, int);         // Sąrašai darbui
};
//                    Pagrindinė programa
void main() {
  FILE *D;
  student A;                 // Objekto aprašymas
  clrscr();
  D = fopen("stud.dat", "r");
  if (D == NULL) {
    cout << "Failo Stud.dat nepavyko atidaryti!";
    exit(1);  }
  A.Formuoti(D);
  cout << "---------------------\n";
  fclose(D);
  A.Spausdinti();
  cout << "---------------------\n";
  A.Darbai(3, 3);
}
//             Žiedinio studentų sąrašo formavimas
void student::Formuoti(FILE *F) {
  sar *D = NULL;
  while(!feof(F)) {
    D = new sar;
```

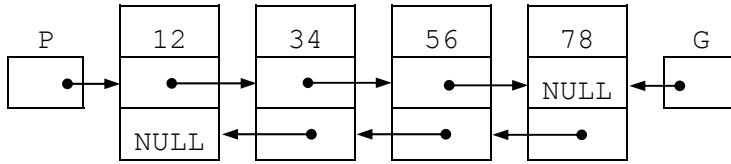
```
    D->st = new studentas;
    fscanf(F, "%s%s%f", D->st->pavard, D->st->vardas,
                                       &D->st->vidur);
    D->next = P;
    P = D;  }
  if (P) {                        // Sąrašo pabaigos paieška
    D = P;
    while(D->next)  D = D->next;
    D->next = P; }                // Žiedo sudarymas
}
//                  Studentų sąrašo spausdinimas
void student::Spausdinti() {
  sar *D;
  if (P) {                        // Pirmojo spausdinimas
    D = P;
    cout << D->st->pavard << " "
         << D->st->vardas << " "
         << D->st->vidur  << "\n";
    D = D->next;
    while(D != P) {               // Kitų spausdinimas
      cout << D->st->pavard << " "
           << D->st->vardas << " "
           << D->st->vidur  << "\n";
      D = D->next;  }}
}
//                        Darbų sąrašas
void student::Darbai(int Dien, int kiek) {
  sar *D = P;
  int i, k;
  if (!D) { cout << "Sąrašas tuščias\n";  exit; }
  for(i=1; i<=Dien; i++) {
    cout << "Diena: " << i << "\n";
    for(k=1; k<=kiek; k++) {
      cout << "  " << D->st->pavard << " "
                   << D->st->vardas << "\n";
      D = D->next;  }}
}
```

8 skyrius. Dvikrypčiai tiesiniai sąrašai

8.1. Sąrašo struktūra

Pagrindinis vienkrypčių sąrašų trūkumas yra tas, kad juose galimas tik nuoseklus elementų perrinkimas. Dėl to juose lėta duomenų paieška, neefektyvios su paieška susijusios tvarkymo operacijos. Siekiant išvengti šio tiesinių sąrašų trūkumo, yra sukurtos įvairios tiesinių sąrašų modifikacijos. Pavyzdžiui, realizuojant tiesiniame sąraše eilę, jis yra papildomas išorine pabaigos rodykle, kuri leidžia greitai papildyti sąrašą nauju elementu gale. Tą patį efektą galima pasiekti žiedinėje tiesinio sąrašo modifikacijoje. Tačiau abi šios modifikacijos nepalengvina paieškos pagal požymį ir galinio elemento šalinimo operacijų.

Paieškai ir elementų šalinimui geriau tinka dvikryptė tiesinių sąrašų modifikacija, kurios elementuose yra dvi rodyklės, rodančios pirmesnį ir tolimesnį gretimus elementus, ir viena arba dvi išorinės rodyklės (8.1 pav.). Tokiuose sąrašuose galima nuo bet kurio jų elemento pasiekti visus kitus elementus.



8.1 pav. Dvikrypčio sąrašo struktūra

Dvikrypčių sąrašų elementų struktūra:

```
Struct  sar {
  int  *S;            // Duomenų rodyklė
  sar *de;            // Rodyklė į tolimesnį elementą (dešinė)
  sar *ka;            // Rodyklė į artimesnį elementą (kairė)
            }
```

Dvikryptis sąrašas gali būti laikomas dviejų vienkrypčių sąrašų kompozicija, todėl jo tvarkymo funkcijos yra panašios į šių sąrašų tvarkymo funkcijas. Pagrindinis skirtumas yra tas, kad tvarkymo

operacijose reikia formuoti dvi rodykles. Be to, galima sudaryti universalias elementų įterpimo ir šalinimo operacijas, kurios tinka visiems sąrašo elementams, ne tiktai galiniams.

Pradžioje sąrašas tuščias. Todėl pirmojo elemento rodyklės `P` (pradžia) ir galinio elemento rodyklės `G` (galas) reikšmės turi būti nulinės – `NULL`. Tuščias sąrašas yra inicializuojamas deklaruojant nulines jo rodykles:

```
sar *P = NULL,   *G = NULL;
```

8.2. Sąrašo formavimas ir peržiūra

Dvikryptis sąrašas tik `P` arba `G` rodyklių atžvilgiu yra analogiškas vienkryčiui sąrašui. Formuojant sąrašą, kai nauji elementai prijungiami sąrašo pradžioje (rodyklė `P`), rodyklės `G` atžvilgiu nauji elementai bus jungiami sąrašo gale. Ir atvirkščiai. Darome išvadą, kad šio sąrašo formavimui tinka veiksmai, kurie vykdomi vienkrypčiame sąraše formuojant sąrašą steko ir eilės būdais.

```
//                      Dvikryptis sąrašas
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

typedef struct sar { int *S;          // Rodyklė į elementą
                     sar *de;         // Rodyklė į dešinę
                     sar *ka; };      // Rodyklė į kairę
//                    Klasės aprašymas
class DviKryptys {
  sar *P;                             // Sąrašo pradžia
  sar *G;                             // Sąrašo galas
  public:
    DviKryptys() { P = NULL; G = NULL; };   // Konstruktorius
   ~DviKryptys();                           // Destruktorius
    sar *GetP() { return P; };
    sar *GetG() { return G; };
    void Formuoti(FILE *);            // Formuoja sąrašą
    void Spausdinti();                // Spausdina sąrašą
    void SpausdintiAtv();             // Spausdina sąrašą atvirkščiai
};
```

```
//                    Pagrindinė programa
void main() {
  FILE *D;
  DviKryptys A;               // Klasės DviKryptys objektas
  clrscr();
  D = fopen("Duom.dat", "r");
  if (D == NULL) {
    cout << "Failo Duom.dat nepavyko atidaryti";
    exit(1); }
  A.Formuoti(D);
  fclose(D);

  A.Spausdinti();
  cout << "***********\n";
  A.SpausdintiAtv();
  A.~DviKryptys();
}
//              Sąrašo formavimas, rašant į pradžią
void DviKryptys::Formuoti(FILE *F) {
  sar *R;                      // Darbinis kintamasis
  if (!feof(F)) {              // Pirmojo elemento prijungimas
    G = new sar;
    P = G;
    P->S = new int;
    fscanf(F, "%i", P->S);
    P->ka = NULL;
    P->de = NULL;  }
  while (!feof(F)) {           // Likusių elementų prijungimas
    R = new sar;
    R->S = new int;
    fscanf(F, "%i", R->S);
    R->ka = NULL;
    R->de = P;
    P->ka = R;
    P = R;  }
}
//                Spausdina sąrašą nuo pradžios
void DviKryptys::Spausdinti() {
  sar *D = P;
  while(D != NULL) {
    cout << *D->S << "\n";
    D = D->de;  }
}
```

```
//                    Spausdina sąrašą nuo galo
void DviKryptys::SpausdintiAtv() {
  sar *D = G;
  while(D != NULL) {
    cout << *D->S << "\n";
    D = D->ka;  }
}
//                          Destruktorius
DviKryptys::~DviKryptys() {
  sar *D = P;
  while(P != NULL) {
    D = P;
    P = P->ka;
    delete(D->S);
    delete(D);  }
}
```

Rodyklė `P` rodo sąrašą, kuriame duomenys saugomi atvirkštine tvarka, o rodyklė `G` rodo sąrašą tiesiogine tvarka. Norint sukeisti rodyklių reikšmes sąrašo tvarkos požiūriu, reikia modifikuoti funkciją **`Formuoti`**. Siūlome tai padaryti patiems ir patikrinti.

Sąrašo peržiūrai tinka vienkrypčio sąrašo peržiūros algoritmas. Sąrašas nagrinėjamas panaudojant vieną rodyklę: `P` ir `de` lauko reikšmes arba `G` ir `ka` lauko reikšmes. Turime galimybę sąrašą peržiūrėti tiesiogine ir atvirkštine tvarka. Tai demonstruoja spausdinimui skirtos funkcijos.

8.3. Sąrašo papildymas

Galimos trys situacijos:

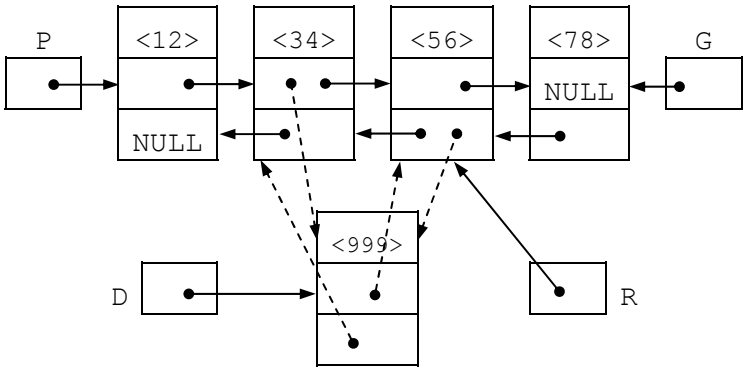
- naujo elemento prijungimas sąrašo pradžioje (`P` rodyklės atžvilgiu),
- sąrašo gale (`G` rodyklės atžvilgiu),
- bet kurioje sąrašo vietoje (rodyklės `P` ir `G` nekinta).

Pirmos dvi situacijos atitinka vieno elemento prijungimo veiksmams sąrašo formavimo procese, todėl atskirai jų neaptarsime. Kuriant funkciją, skirtą naujo elemento įterpimui, reikia patikrinti situaciją ir atlikti jai skirtus veiksmus (prieš pirmą elementą, po paskutinio elemento arba sąrašo viduje). Elemento įterpimas sąrašo viduje po elemento `R` yra tapatus veiksmams prieš elementą `R`.

```
D->de = R;          // Naujas elementas “pamato” savo naujus kaimynus
```

```
D->ka = R->ka;
(R->ka)->de = D;   // Elementas D įjungiamas į sąrašą
R->ka = D;
```

Įterpiant naują elementą D prieš rodyklės R nurodomą sąrašo elementą, reikia suformuoti keturias naujas, punktyrinėmis linijomis parodytas, rodykles (8.2 pav.), kurios turi pakeisti dvi elementų ryšius aprašančias rodykles. Pradžioje reikia sutvarkyti naujo elemento ryšius su būsimais kaimynais sąraše, o po to pakeisti sąrašo elementų ryšius į naująjį elementą.



8.2 pav. Naujo elemento D įterpimas prieš elementą R

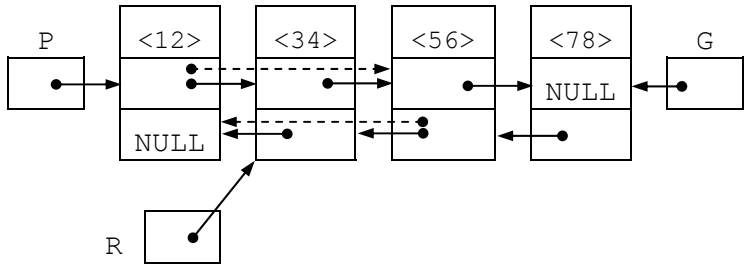
Universalios dvikrypčio sąrašo papildymo funkcijos **`Terpti`** realizacija ir panaudojimo pavyzdžiai:

```
void Terpti(sar *, int *);   // Metodo prototipas
  int x = 0,  y = 100;       // Įterpiamos reikšmės

  T = A.GetP();
  A.Terpti(T, &x);       // Įterpimas sąrašo pradžioje
  A.Terpti(T->de, &x); // Įterpimas prieš antrąjį elementą. Jis privalo būti
  T = A.GetG();
  Terpti(T, &y);         // Įterpimas prieš paskutinį elementą
//                Įterpimas "prieš" R elementą
void DviKryptys::Terpti(sar *R, int *sk) {
  sar *D;
  D = new sar;           // Naujas elementas
  D->S = sk;             // Rodyklė į reikšmę
```

```
  if (R == P) {          // Kai R rodo pirmą: padėti pradžioje
    D->de = P;
    D->ka = NULL;
    P->ka = D;
    P = D;  }
  else {                 // Padėti sąraše prieš nurodytą R
    D->de = R;
    D->ka = R->ka;
    (R->ka)->de = D;
    R->ka = D;  }
}
```

8.4. Elementų šalinimas

Universali elementų šalinimo funkcija yra paprastesnė, nes jai reikia suformuoti tik dvi naujas sąrašo rodykles. Šioje funkcijoje reikia atsižvelgti į atvejus, kai šalinamas pradinis elementas, paskutinis elementas ir kai šalinamas elementas iš sąrašo vidaus (8.3 pav.). Funkcija grąžina rodyklę į pašalinto elemento duomenų saugojimo vietą (arba `NULL`). Tai reikalinga, nes yra šalinama nuoroda sąraše į duomenis, bet ne patys duomenys. Kaip elgtis su duomenimis sprendžia duomenų apdorojimo paprogramės. Šalinimo funkcija skirta nurodyto elemento pašalinimui. Tai reiškia, kad prieš tai buvo atlikta to elemento paieška. Jeigu elementas nebuvo surastas, tai ir nebus panaudota šalinimo funkcija.



8.3 pav. Elemento `R` pašalinimas

Elemento šalinimo funkcijoje atsimenama ir grąžinama rodyklė į duomenis, į kuriuos buvo nuoroda šalinamame elemente. Šis veiksmas šiai funkcijai yra perteklinis (nebūtinas), nes atliekant šalinamo elemento paiešką yra nagrinėjami duomenys, o tai reiškia, kad juos jau turime.

```
R->de->ka = R->ka;
R->ka->de = R->de;
```

Šalinimo funkcijos **`Mesti`** pavyzdyje nagrinėjamos visos trys situacijos, bet nenagrinėjama situacija, kai sąrašas tuščias ir ar šalinamo elemento rodyklė netuščia. To daryti nebūtina, nes programos dalyje, kurioje nagrinėjami pašalinimo iš sąrašo kandidatai, būtina patikrinti, ar toks elementas egzistuoja. Siūlome papildyti pavyzdžio programą šia funkcija ir patikrinti jos darbą.

```
int *Mesti(sar *);          // Metodo prototipas

  T = A.GetP();
  A.Mesti(T);                    // Šalinamas pirmas elementas
  T = A.GetG();
  A.Mesti(T);                    // Šalinamas paskutinis elementas
  T = A.GetP();
  A.Mesti(T->de);                // Šalinamas antras elementas
//                    Šalinti R elementą
int *DviKryptys::Mesti(sar *R) {
  int *sk = R->S;
  if (R == P) {                  // Jeigu sąrašo pradžioje
    P = P->de;
    if (P != NULL) P->ka = NULL; }
  else
    if (R == G) {                // Jeigu sąrašo pabaigoje
      G = G->ka;
      if (G != NULL) G->de = NULL; }
    else {                       // Jeigu sąrašo viduje
      R->de->ka = R->ka;
      R->ka->de = R->de; }
  delete(R);
  return sk;
}
```

8.5. Tvarkymo veiksmai

Sąrašo elementų tvarkymo veiksmams priskiriamos paieškos, atrankos ir rikiavimo operacijos.

Paieškos veiksmams negalima sudaryti universalių paprogramių, nes tai susieta su konkrečiu uždaviniu, duomenimų struktūra ir konkrečiu paieškos raktu. Paieškoje galima skirti du būdus:

- Sąraše duomenys rakto atžvilgiu yra nesutvarkyti. Paieška atliekama nuosekliai peržiūrint visus sąrašo elementus. Paieška nutraukiama, kai

surandamas reikalingas elementas. Tik visą sąrašą peržiūrėjus galima sužinoti, kad neradome reikalingo elemento.

- Sąrašo duomenys rakto atžvilgiu yra sutvarkyti. Paieška vykdoma nuosekliai peržiūrint elementus. Paieška nutraukiama suradus tinkamą elementą arba susidarius situacijai, kai tolesnis sąrašo nagrinėjimas yra beprasmis: jau tikrai toliau sąraše negali būti ieškomas elementas. Pavyzdžiui, telefonų knygoje, ieškant abonento pagal pavardę, nebūtina perskaityti viso sąrašo iki galo.

Duomenų atrankos veiksmai priskirtini naujo sąrašo formavimui, nes yra sudaromas naujas nuorodų į jau turimus duomenis sąrašas. Pavyzdžiui, turime studentų sąrašą. Galima sudaryti nuorodų sąrašą tik į pirmo kurso studentus.

Rikiavimo pagal duotą raktą funkcijos yra analogiškos funkcijoms, skirtoms darbui su vienkrypčiu sąrašu. Čia reikia nepamiršti, kad sąrašas dvikryptis ir reikia koreguoti dvi rodykles elementui. Šio tipo sąrašuose veiksmų su rodyklėmis padvigubėja, tačiau paieškai skirtų rodyklių sumažėja. Jeigu rikiavime leidžiama keisti vietomis duomenis, tuomet naudojamos tik vienos krypties rodyklės (pradžios arba pabaigos). Jeigu reikia tvarkyti nuorodas, tuomet veiksmai tampa komplikuoti. Problema išsprendžiama, kuomet sąraše saugomi ne duomenys, o nuorodos į juos. Rikiavimo metu keičiame vietomis tas nuorodas. Duomenys savo padėties nekeičia. Tai leidžia formuoti atskirus nuorodų, tvarkingus pagal įvairius požymius, sąrašus. Pavyzdžiui, turime telefonų abonentų sąrašą. Galima sudaryti nuorodų sąrašą pagal pavardes, abėcėlės tvarka. Galima sudaryti sąrašą, kuriame būtų nuorodos į tą patį abonentų sąrašą, tik telefonų numerių didėjimo seka.

Dvikryptis sąrašas gali būti naudojamas su tam tikrais apribojimais. Tai dvigubo steko bei eilės tipo sąrašai. Vartotinas dvigubo žiedo sąrašas, nes jame įterpimo bei šalinimo veiksmai nesudėtingi, be to paprastas sąrašo skenavimas pirmyn-atgal.

Dirbant su sąrašinėmis struktūromis, visuomet būtina tikrinti nagrinėjamo elemento padėtį sąraše: kraštinis ar viduje. To galima išvengti, jeigu sąrašo kraštinius elementus turėsime fiktyvius, t.y. jie nebus susieti su duomenimis, atliks buferinių elementų funkciją. Tuščias sąrašas bus tuo atveju, kai bus tik tie elementai. Vienkrypčiame sąraše paprastai toks elementas formuojamas pradžioje, tačiau eilės atveju jį naudinga turėti ir gale. Dvikryptis sąrašas paprastai turi du kraštinius “tuščius” elementus,

nors pakanka ir vieno. Tie elementai gali būti pastovūs arba formuojami laikinai, kol bus atlikti atitinkami veiksmai, – po to jie šalinami.

8.6. Panaudojimo pavyzdys

Tekstiniame faile **`Stud.dat`** turime studentų sąrašą: pavardė, vardas, mokymosi vidurkis. Reikia pašalinti nepažangius studentus.

Formuosime dvikryptį sąrašą. Elementų šalinimo veiksmuose sąrašą papildysime pradžioje ir gale "tuščiais" elementais. Gausime situaciją, kai visi duomenų elementai yra sąrašo viduje pradžios ir galo atžvilgiu. Elementų šalinimas supaprastėja.

```
//                    Panaudojimo pavyzdys
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

typedef struct studentas { char pavard[10];
                           char vardas[10];
                           float vidur;  };

typedef struct sar { studentas *st;
                     sar *de;
                     sar *ka;  };

//              Klasės Studentai apibrėžimas
class Studentai {
  sar *P;                                  // Sąrašo pradžia
  sar *G;                                  // Sąrašo galas
  public:                                  // Konstruktorius
    Studentai() { P = NULL;  G = NULL; }
    void Formuoti(FILE *);                 // Formuoja sąrašą
    void Spausdinti();                     // Spausdina sąrašą
    void Tvarkymas(float);                 // Tvarkymas
};
//                    Pagrindinė programa
void main() {
  FILE *D;
  Studentai A;                      // Klasės Studentai objektas
  clrscr();
```

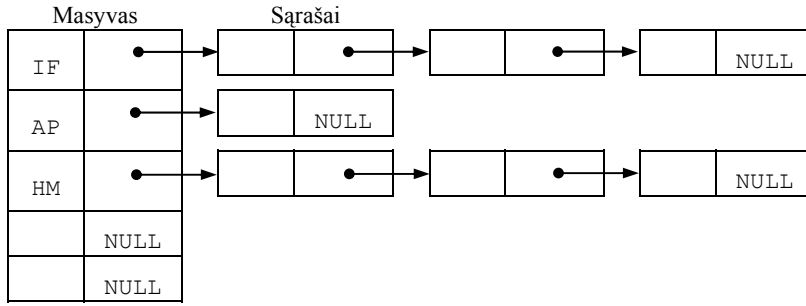
```
  D = fopen("stud.dat", "r");
  if (D == NULL) {
    cout << "Failo Stud.dat  nepavyko atidaryti ";
    exit(1);  }
  A.Formuoti(D);
  fclose(D);
  A.Spausdinti();
  cout << "---------------------\n";
  A.Tvarkymas(4.5);
  A.Spausdinti();
}
//                  Studentų sąrašo formavimas
void Studentai::Formuoti(FILE *F) {
  sar *R;
  if (!feof(F)) {
    R = new sar;
    R->st = new studentas;
    fscanf(F, "%s%s%f", R->st->pavard, R->st->vardas,
                                       &R->st->vidur );
    R->de = NULL;
    R->ka = NULL;
    P = R;
    G = R;  }
  while(!feof(F)) {
    R = new sar;
    R->st = new studentas;
    fscanf(F, "%s%s%f", R->st->pavard, R->st->vardas,
                                       &R->st->vidur);
    R->ka = G;  G->de = R;  R->de = NULL;  G = R;
  }
}
//                  Sąrašo spausdinimas
void Studentai::Spausdinti(){
  sar *D = P;
  while(D != NULL) {
    cout << D->st->pavard << " "
         << D->st->vardas << " "
         << D->st->vidur  << "\n";
    D = D->de;     }
}
//                  Šalinimas iš sąrašo
void Studentai::Tvarkymas(float vd) {
  sar *R, *D;
  R = new sar;                  // "Tuščias" elementas sąrašo pradžioje
```

```
  R->ka = NULL;
  R->de = P;
  R->st = NULL;
  P->ka = R;
  P = R;
  R = new sar;                  // "Tuščias" elementas sąrašo gale
  R->de = NULL;
  R->ka = G;
  R->st = NULL;
  G->de = R;
  G = R;
  R = P->de;
  while(R->de != NULL) {        // Šalinimo ciklas
    if (R->st->vidur < vd) {
      R->de->ka = R->ka;
      R->ka->de = R->de;
      D = R;
      R = R->ka;
      delete(D->st);    // Šalinimas rodyklės į studentą
      delete(D);  }     // Šalinimas sąrašo elementas
    R = R->de; }
  R = P;                // "Tuščio" elemento šalinimas iš sąrašo pradžios
  P = R->de;
  P->ka = NULL;
  delete(R);
                        // "Tuščio" elemento šalinimas iš sąrašo galo
  R = G;
  G = R->ka;
  if(G)  G->de = NULL;
  else   P = NULL;      // Sąrašas tuščias
  delete(R);
}
```

9 skyrius. Tiesinių sąrašų rinkiniai

9.1. Sąrašų struktūra

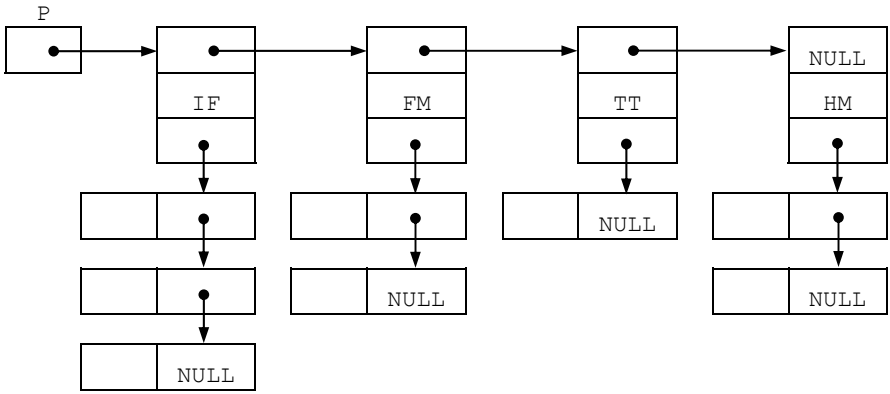
Apdorojant duomenis, ne visuomet patogu turėti vieną duomenų sąrašą, ypač kai tų duomenų daug. Patogu turėti keletą atskirų tiesinių sąrašų. Pavyzdžiui, turime universiteto kiekvieno fakulteto studentų sąrašus. Galima sudaryti tiek sąrašų, kiek yra fakultetų. Tačiau prireikus papildyti duomenis naujo fakulteto sąrašu, reikia modifikuoti programą. Tai nepatogu. Galima suformuoti rodyklių į fakultetus masyvą, paliekant rezervą naujiems sąrašams registruoti. Tokio sąrašo schema parodyta 9.1 pav.



9.1 pav. Daugiasąraišinė struktūra, organizuojama masyvu.
Sąrašų raktiniai žodžiai ir rodyklės į sąrašus saugomos masyve

Lankstesnė struktūra gaunama, kai vietoje masyvo yra formuojamas nuoseklus sąrašas, vienkryptis arba dvikryptis (9.2 pav.). Tokio sąrašo elementas turi rodyklę į duomenų tiesinį sąrašą ir sąrašo pavadinimo lauką (arba rodyklę į tą pavadinimą). Gauname **šakotą** sąrašą, kurio struktūra universali duomenų atžvilgiu: nereikia modifikuoti programą papildant nauju duomenų sąrašu. Tinka visi veiksmai, aptarti tiesiniams sąrašams.

Priklausomai nuo pradinių duomenų funkcinės tarpusavio priklausomybės ir planuojamų apdorojimo veiksmų yra projektuojama sąraišinė struktūra. Ją tikslinga parinkti tokią, kad veiksmai būtų kuo paprastesni. Kurdami šakotą struktūrą mes iš anksto numatome tam tikrą duomenų suskirstymą, klasifikavimą. Gerai suprojektuota struktūra supaprastina veiksmus.



9.2 pav. Tiesinių sąrašų rinkinys

9.2. Sąrašo panaudojimo pavyzdys

Tekstiniame duomenų faile yra studentų sąrašas: pavardė, vardas, fakultetas. Reikia suformuoti duomenų sąrašą pagal fakultetus.

Panaudosime sąrašą, kurio schema parodyta 9.2 pav. Duomenų skaitymo metu ieškosime šakos su studento fakulteto pavadinimu. Jeigu nebus, sukursime naują elementą pagrindiniame sąraše su naujo fakulteto pavadinimu ir atidarysime šaką su to studento pavarde ir vardu. Jeigu rasime elementą su tuo fakultetu, tai šakoje sukursime naują elementą su naujo studento pavarde ir vardu. Baigus skaityti duomenų failą turėsime šakotą sąrašą, kuriame studentai bus suskirstyti fakultetais.

Sąrašo papildymas naujais duomenimis yra tapatus formavimui. Šalinant studentus iš sąrašo gali susidaryti situacija, kai fakultete neliks nei vieno studento. Tuo atveju tikslinga pašalinti iš pagrindinio sąrašo elementą su fakulteto pavadinimu. Galimas atvejis, kai nešaliname fakulteto elemento, bet nuorodoje į studentų sąrašą parašome tuščio adreso reikšmę `NULL`.

Duomenų failas: "Studf.dat"			
Petraitis	Petras	IF	8.9
Jurgelis	Jurgis	HM	5.6
Batuotas	Katinas	EK	7.8
Lapinas	Rudas	IF	10
Medinis	Jurgis	IF	8
Ramusis	Petras	HM	6.3

```
//                Sąrašo panaudojimo pavyzdys
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

typedef struct studentas { char pavard[10];
                           char vardas[10];
                           char fakult[10];
                           float vidur;  };

typedef struct sar { studentas st;
                     sar *kitas;  };

typedef struct fakultetas { char fak[10];
                            fakultetas *sek;
                            sar *stud;  };
//                  Klasės Uni aprašymas
class Uni {
  fakultetas *P;                // Dinaminio sąrašo pradžia
  public:
    Uni() { P = NULL; };        // Konstruktorius
    void Formuoti(FILE *);      // Formuoja sąrašą
    void Spausdinti(sar *);     // Spausdina sąrašą
    void Visi_stud();
    void Pasirink();
};
//                   Pagrindinė programa
void main() {
  FILE *D;
  Uni A;                                // Klasės Uni objektas
  clrscr();
  D = fopen("studf.dat", "r");
  if (D == NULL) {
    cout << "Failo Studf.dat nepavyko atidaryti ";
    exit(1);  }
  A.Formuoti(D);
  fclose(D);

  A.Visi_stud();
  A.Pasirink();
}
```

```
//                    Sąrašo formavimas
void Uni::Formuoti(FILE *F) {
  fakultetas *R;
  sar *A;
  int yra;

  while(!feof(F)) {
    A = new sar;
    fscanf(F, "%s%s%s%f", A->st.pavard,  A->st.vardas,
                          A->st.fakult, &A->st.vidur);

  yra = 0;
  R = P;
  while(!yra && R)
    if (strcmp(A->st.fakult, R->fak) == 0)  yra = 1;
    else  R = R->sek;

  if (!yra) {
    R = new fakultetas;
    R->stud = NULL;
    strcpy(R->fak, A->st.fakult);
    R->sek = P;
    P = R;  }

  A->kitas = R->stud;
  R->stud = A;
  }
}
//          Spausdina vieno fakulteto studentų sąrašą
void Uni::Spausdinti(sar *R) {
  while(R) {
    cout << R->st.pavard << " "
         << R->st.vardas << " "
         << R->st.vidur  << "\n";
  R = R->kitas;  }
  cout << "------------------------\n";
}
//          Spausdina visų fakultetų studentų sąrašus
void Uni::Visi_stud() {
  fakultetas *R = P;
  cout << "=== Fakultetų ir studentų sąrašai ===\n\n";
  while(R) {
    cout << "Fakultetas " << R->fak << ":\n";
    Spausdinti(R->stud);
```

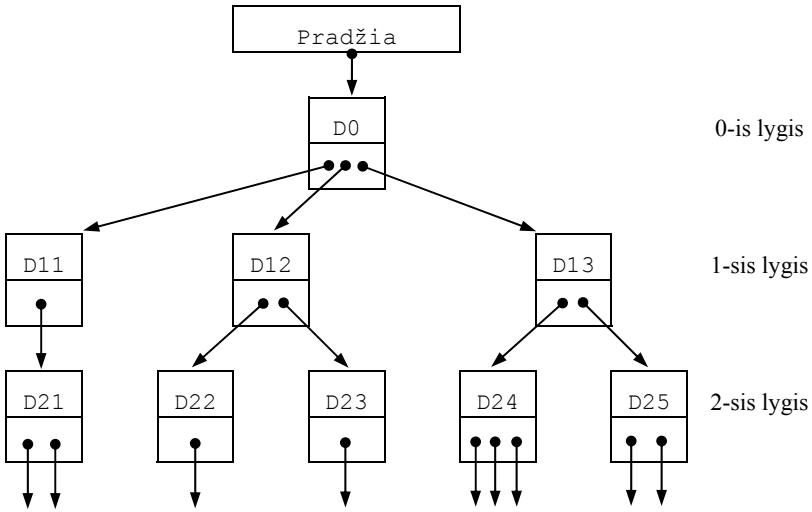
```
    R = R->sek;
  }
}
//       Pageidaujamo fakulteto studentų sąrašo spausdinimas
void Uni::Pasirink() {
  fakultetas *R;
  char fk[10], s;
  int yra, rinkti = 1;

  while(rinkti) {
    cout << "Fakultetas= ";
    cin >> fk;
    yra = 0;
    R = P;
    while(!yra && R)
      if (strcmp(fk, R->fak) == 0)  yra = 1;
      else  R = R->sek;

    if (yra) {
      cout << R->fak << ":\n";
      Spausdinti( R->stud );  }
    else  cout << fk << " sąraše nerastas\n";

    cout << "------------------------\n";
    cout << "Ar dar renkate fakultetą(T)?";
    cin >> s;
    if ((s != 'T') && (s != 't'))  rinkti = 0;
  }
}
```

10 skyrius. Netiesiniai sąrašai

10.1. Sąrašo struktūra

Sąrašas, kurio elementus ryšio tvarka negalima išdėstyti į vieną grandinę, vadinamas **netiesiniu**. Tai grafus vaizduojantys sąrašai. Čia viršūnės atitinka sąrašo elementų reikšmes, o viršūnes jungiantys lankai atitinka rodyklėmis aprašomus ryšius tarp jų. Nuo grafo viršūnių sujungimų būdų priklauso jo savybės, grafo viršūnių perrinkimo būdai. Veiksmų su sąrašais sudėtingumas priklauso nuo grafų struktūros ir duomenų apdorojimo algoritmų.

Patys paprasčiausi ir dažniausiai vartojami yra **medžio** tipo netiesiniai sąrašai (10.1 pav.), kurių elementus galima išdėstyti į kelis hierarchijos (pavaldumo) lygius taip, kad pradiniame, nuliniame, lygyje būtų tik vienas elementas, o į kiekvieną žemesnio hierarchijos lygio elementą vestų tiktai viena nuoroda ir tik iš aukštesnio hierarchijos lygio elemento.



10.1 pav. Medžio tipo sąrašo grafinis vaizdas

Aprašant programose medžio tipo struktūras, yra paprastai ribojamas nuorodų į žemesnius hierarchijos lygius (rodyklių) skaičius. Jei jis apribojamas dviem, tai toks medis yra vadinamas **binariniu**.

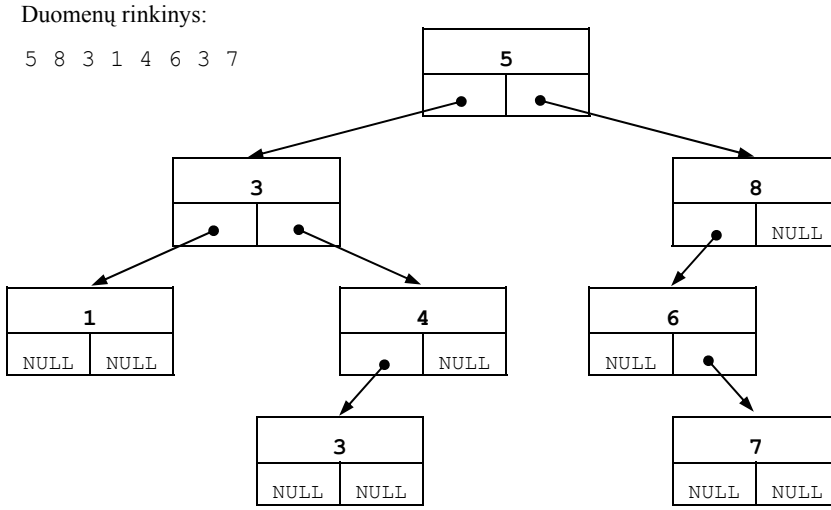
Binarinių medžių elementai yra aprašomi taip pat, kaip ir dvikrypčių sąrašų elementai, ryšio dalyje formuojant dvi nuorodas. Tiktai priimta šias nuorodas žymėti kitokiais vardais, pabrėžiant, kad medžio šakos yra nukreiptos į dešinę arba kairę pusę (`de`, `ka`):

```
struct medis {
  void *data;          // Duomenų rodyklė
  medis *ka;           // Kairės medžio šakos rodyklė
  medis *de; }         // Dešinės medžio šakos rodyklė
```

Binariniai medžiai labai gerai tinka organizuoti tokioms saugojimo struktūroms, iš kurių dažnai tenka pagal požymius atrinkinėti pavienius elementus arba nedideles jų grupes. Aprašant medžio tipo struktūras, taip pat reikia apibrėžti jų formavimo, tvarkymo bei apdorojimo procedūras.

10.2. Formavimas

Formavimo veiksmai priklauso nuo sąrašo struktūros ir duomenų tarpusavio priklausomybės. Universalių paprogramių negalima sudaryti.



10.2 pav. Binarinis medis

Medžio formavimą iliustruosime tokiu pavyzdžiu. Tarkime, kad reikia sudaryti medį, kuriame būtų saugomi elementai, turintys sveikų skaičių reikšmes. Skaičiai gaunami iš failo, kuriame jie surašyti atsitiktine tvarka. Medyje juos reikia patalpinti didėjimo tvarka (10.2 pav.).

Formuosime binarinį medį. Tai bus padaryta taip. Pirmas medžio elementas turės pirmojo skaičiaus reikšmę. Kitiems skaičiams bus sukuriami elementai, kurių vieta medyje bus randama tokiu dėsniu: jeigu nauja reikšmė yra mažesnė už nagrinėjamo medžio elemento saugomą reikšmę, tai pereiname prie kairiojo elemento, kitaip prie dešiniojo. Naujas elementas "pakabinamas" medyje toje vietoje, kur randamas šakos galas, t.y. elementas su rodyklės reikšme `NULL`. Naujas elementas su nauja reikšme turi turėti `ka` ir `de` reikšmes `NULL`.

```
//                  Medžio sudarymo pavyzdys
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

typedef struct medis { int sk;
                       medis *de;
                       medis *ka; };
//               Klasės Binmedis aprašymas
class Binmedis {
  medis *P;                    // Medžio viršūnė
  public:
    Binmedis() { P = NULL; }; // Konstruktorius
    void Formuoti(FILE *);     // Formuoja medį
    medis *Get() { return P; };
    void Padeti(medis *);      // Naujo elemento vieta medyje
    void Spausdinti();         // Spausdina reikšmes didėjimo tvarka
                               // Spausdina reikšmes mažėjimo tvarka
    void SpausdintiRek(medis *);
    medis *Rasti(int);         // Duotos reikšmės paieška medyje
};
//                    Pagrindinė programa
void main() {
  FILE *D;
  medis *R;
  Binmedis A;                                  // Objekto aprašymas
  clrscr();
  D = fopen("duom.dat", "r");
```

```
  if (D == NULL) {
    cout << "Failo Duom.dat nepavyko atidaryti ";
    exit(1);  }
  A.Formuoti(D);
  fclose(D);

  A.Spausdinti();                          // Didėjimo tvarka
  cout << "\n" << "======================\n";
  R = A.Get();
  A.SpausdintiRek(R);                      // Mažėjimo tvarka
  cout << "\n" << "======================\n";

  for(int i=0; i<=10; i++) {               // Paieškos iliustracija
    R = A.Rasti(i);
    if (R)  cout << i << "  Radau!\n";
    else  cout << i << "* neradau\n";  }
}
//                    Medžio formavimas
void Binmedis::Formuoti(FILE *F) {
  medis *R;
  while(!feof(F)) {                  // Naujas elementas
    R = new medis;
    R->ka = NULL;
    R->de = NULL;
    fscanf(F, "%i", &R->sk);
    if (!P)  P = R;                  // Elementas dedamas į medį
    else  Padeti(R);  }
}
//                    Padėti elementą medyje
void Binmedis::Padeti(medis *R) {
  medis *D = P, *T;
  while(D) {
    T = D;                           // Tuščios šakos paieška
    if (R->sk < D->sk)  D = D->ka;
    else  D = D->de; }
  if (R->sk < T->sk)  T->ka = R;  // Elemento prijungimas
  else  T->de = R;
}
//                Spausdinimas didėjimo tvarka
void Binmedis::Spausdinti() {
  medis *R = P;
  struct stekas {medis *S;        // Stekas medžio trasavimui
                 stekas *sek; } *A = NULL, *D;
```

```
  while(R || A) {
    if (R)  {
      D = new stekas;     // Viršūnės adreso užrašymas į steką
      D->sek = A;
      A = D;
      A->S = R;
      R = R->ka; }        // Perėjimas į kairę
    else  {
      R = A->S;           // Kraštinio elemento adresas imamas iš steko
      D = A;
      A = A->sek;
      delete(D);
      cout << R->sk << " ";       // Spausdinama reikšmė
      R = R->de;  }               // Perėjimas į dešinę
  }
}
//          Spausdinimas mažėjimo tvarka (su rekursija)
void Binmedis::SpausdintiRek(medis *R) {
  if (R) {
    SpausdintiRek(R->de);
    cout << R->sk << " ";
    SpausdintiRek(R->ka);  }
}
//          Elemento, turinčio nurodytą reikšmę paieška
medis *Binmedis::Rasti(int Sk) {
  medis *R = P;
  while(R && (R->sk != Sk))
    if (Sk < R->sk)  R = R->ka;
    else  R = R->de;
  return R;                   // Neradus grąžinama reikšmė NULL
}
```

10.3. Peržiūra

Perrenkant medyje esančias reikšmes yra vadovaujamasi medžiui sukurti naudotais principais. Pavyzdžio medį galima peržiūrėti skaičių didėjimo arba mažėjimo tvarka. Peržiūra pradedama nuo medžio viršūnės ir ieškome toliausiai kairėje esančios šakos viršūnės, kurios reikšmė bus mažiausia. Ją spausdiname ir pasukame į artimiausią dešinę šaką, nuo kurios vėl ieškome toliausiai į kairę pusę esančios viršūnės. T.y. kartojame medžio formavimo procesą. Kadangi medžiu galime judėti tik viena kryptimi (nėra nuorodų į aukštesnį lygį), tai būtina atsiminti viršūnes, per

kurias buvo nusileidžiama iki atitinkamos viršūnės. Tam naudojamas stekas, kurį galima organizuoti masyve, tiesiniu sąrašu arba išreikšti netiesiogiai panaudojant rekursinį paieškos būdą. Pavyzdžio programoje yra funkcija `Spausdinti`, kuri skirta spausdinti medžio reikšmes didėjimo tvarka ir kuri stekui naudoja tiesinį sąrašą. Funkcija `SpausdintiRek` skirta spausdinti medžio saugomas reikšmes mažėjimo tvarka ir yra rekursinė.

Ieškant medyje duotos reikšmės yra vadovaujamasi vienu iš medžio peržiūros būdų: kairiniu arba dešininiu. Pavyzdžio programoje funkcija `Rasti` demonstruoja paiešką kairiniu būdu. Šią funkciją galima patobulinti: duomenys yra tvarkingai sudėti, todėl paiešką galima nutraukti, kuomet tolesnė peržiūra tampa beprasmė (ieškoma reikšmė tampa mažesne už aktyvaus elemento saugomą reikšmę).

Binarinių medžių **skleidimu** yra vadinamas visų jų elementų perrinkimas. Nuo perrinkimo būdo priklauso, kokia tvarka medyje saugomi duomenys yra perduodami juos apdorojančioms funkcijoms. Naudojami trys pagrindiniai skleidimo būdai: kairysis, dešinysis ir centrinis.

Naudojant **kairįjį** skleidimo būdą, kiekviename elemente iš pradžių yra perrenkama kairiosios rodyklės rodoma medžio šaka. Po to apdorojami elemento saugomi duomenys ir perrenkama dešinioji šaka. Skleidimas yra pradedamas nuo medžio kamieno.

Dešinysis skleidimas vykdomas priešinga kairiajam skleidimui tvarka, o **centrinio** skleidimo atveju, iš pradžių apdorojamas duomenų elementas ir tik po to perrenkamos kairioji ir dešinioji šakos.

Visi aptarti skleidimo būdai yra labai glaustai aprašomi rekursinėmis funkcijomis. Pavyzdžiu gali būti pavyzdyje pateiktos medyje saugomų reikšmių spausdinimas.

11 skyrius. Savarankiškas darbas

11.1. Programos pavyzdys

Darbo su sąrašinėmis struktūromis įgūdžiai susidaro savarankiškai rašant programas. Siūlome pabandyti visas nagrinėtas dinamines struktūras sudėtingumo seka. Tai galite padaryti su tuo pačiu vienu uždaviniu, modifikuojant padarytas programas. Jūs galėsite palyginti sąrašų taikymo galimybes, jų efektyvumą. Imdami kiekvienai struktūrai skirtingas užduotis, turėsite galimybę išbandyti ne tik sąrašines struktūras, bet ir pagilinti žinias programų struktūros kūrime.

Sąrašinių struktūrų studijas siūlome baigti atskiro uždavinio programos padarymu. Čia reikėtų suprojektuoti tokią savo dinaminę duomenų struktūrą, kuri būtų racionaliausia pasirinktam uždaviniui.

Pavyzdžio užduotis. Kelionių agentūra turi siūlomų turistinių kelionių sąrašą. Žinomas kelionės tipas, trukmė, vieta ir kaina. Keliautojas renkasi kelionę pagal du kriterijus: trukmę ir kainą. Reikia sudaryti sąrašą kelionių, neviršijančių būsimo keliautojo galimybių.

***Duomenų sąrašo pavyzdys*.** Duomenys saugomi tekstiniame faile "`Kelione.sar`" (11.1 pav.). Vienoje eilutėje yra vienos kelionės duomenys, surašyti tokia tvarka: pirmose 10 pozicijų kelionės tipas, toliau trukmė dienomis, 10 pozicijų skiriama vietovės pavadinimui ir eilutės gale – kaina litais. Duomenų faile tuščių eilučių negalima palikti.

```
Dviratis  15 Lietuva   200
Eiti      10 Dzukija   100
Traukinys 15 Turkija   1200
Laivas    25 Afrika    2500
Joti       5 Utena     230
Valtis    12 Neris     345
```

11.1 pav. Kelionių sąrašo pavyzdys

Rezultatų pavyzdys. Rezultatai lentelių formoje kaupiami tekstiniame faile vardu "`Kelione.rez`" (11.2 pav.). Pavyzdyje yra duomenų ir pasirinkimui tinkamų kelionių lentelės. Pasirinkimo sąrašas yra surikiuojamas pagal trukmę ir kainą.

```
Siulomu kelioniu sarasas

   +----+------------+--------+------------+---------+
   | Nr.| Keliones t.+ Trukme +   Vieta    +  Kaina  +
   +----+------------+--------+------------+---------+
   |  1 |     Valtis |     12 |      Neris |     345  |
   |  2 |       Joti |      5 |      Utena |     230  |
   |  3 |     Laivas |     25 |     Afrika |    2500  |
   |  4 |  Traukinys |     15 |    Turkija |    1200  |
   |  5 |       Eiti |     10 |    Dzukija |     100  |
   |  6 |   Dviratis |     15 |    Lietuva |     200  |
   +----+------------+--------+------------+---------+

Atrinktu kelioniu sarasas

   +----+------------+--------+------------+---------+
   | Nr.| Keliones t.+ Trukme +   Vieta    +  Kaina  +
   +----+------------+--------+------------+---------+
   |  1 |       Joti |      5 |      Utena |     230  |
   |  2 |       Eiti |     10 |    Dzukija |     100  |
   |  3 |     Valtis |     12 |      Neris |     345  |
   |  4 |   Dviratis |     15 |    Lietuva |     200  |
   |  5 |  Traukinys |     15 |    Turkija |    1200  |
   +----+------------+--------+------------+---------+
```

11.2 pav. Rezultatų failo pavyzdys

Programos duomenų struktūra. Kelionės duomenims aprašyti skirta struktūra:

```
struct kelione { char tipas[10];
                 char vieta[10];
                 int  trukme;
                 int  kaina; };
```

Duomenys bus saugomi tiesiniu vienkrypčiu sąrašu:

```
typedef struct sar { kelione S;
                     sar *sek;  };
```

Programoje sukurta klasė `Kelionės`. Duomenų sąrašo rodyklė `P`, atrinktų kelionių sąrašo rodyklė `K`.

- Pagrindinė funkcija.
  `P` – duomenų sąrašo rodyklė.
  `K` – rezultatų sąrašo rodyklė.
  `F` – failinis kintamasis.
  Čia atidaromi ir uždaromi failai, formuojamas duomenų sąrašas `P` ir rezultatų sąrašas `K`, rikuojamas sąrašas `K`, spausdinami sąrašų duomenys lentelėmis. Veiksmams atlikti naudojami padaryti metodai.
- `void Keliones::Formuoti(FILE *F);`
  Funkcijai perduodamas atidarytas duomenų failas `F`. Čia skaitomi duomenys po vieną eilutę ir formuojamas duomenų sąrašas. Sąrašo pradžia `P`. Sąrašas formuojamas steko principu. Panaudojama funkcija `Padeti`
- `void Keliones::Padeti(sar **P, kelione N);`
  Sukuria naują elementą su duotos kelionės `N` duomenimis ir prijungia prie sąrašo `P` pradžios.
- `void Keliones::Spausdinti(sar *P, FILE *F);`
  Sąrašo `P` saugomus duomenis lentelės forma surašo į tekstinį failą `F`.
- `void Keliones::Rikiuoti(sar *P);`
  Sąrašo `P` duomenis surikiuoja pagal nurodytą raktą, aprašytą funkcijoje `Raktas`. Sąrašas rikiuojamas daliniu Minmax būdu. Rikiavimo metu duomenys keičiami vietomis.
- `int Keliones::Raktas(kelione A, kelione B);`
  Palyginami dviejų kelionių `A` ir `B` duomenys. Jeigu `B` kelionės trukmė mažesnė už `A` arba, kai trukmės vienodos, `B` kelionės kaina yra didesnė už `A` kelionės kainą, tai grąžinama reikšmė `1` (reikia sukeisti vietomis `A` su `B`), kitaip grąžinama reikšmė `0`.
- `void Keliones::Rinkti();`
  Čia keliautojo paklausiama didžiausia galima kelionės kaina ir ilgiausia galima trukmė. Iš sąrašo `P` atrenkami tinkami duomenys ir suformuojamas naujas sąrašas `K`. Sąrašas formuojamas steko principu. Naudojama funkcija `Padeti`.
- `void Keliones::Naikinti(sar *P);`
  Funkcija pašalina sąrašą `P` iš atminties.

- `sar *Keliones::GetP() { return P; };`
  Grąžina visų duomenų sąrašo pradžios rodyklę `P`.
- `sar *Keliones::GetK() { return K; };`
  Grąžina atrinktų duomenų sąrašo pradžios rodyklę `K`.

Programos tekstas

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>

struct kelione { char tipas[10];
                 char vieta[10];
                 int trukme;
                 int kaina;  };

typedef struct sar { kelione S;
                     sar *sek;  };
//                 Klasės Keliones aprašas
class Keliones {
  sar *P;                              // Visų kelionių sąrašo pradžia
  sar *K;                              // Atrinktų kelionių sąrašas
  public:
    Keliones() { P= NULL; K= NULL; };// Konstruktorius
    void Formuoti(FILE *);             // Formuoja sąrašą
    void Padeti(sar **, kelione);
    sar *GetP() { return P; };
    sar *GetK() { return K; };
    void Spausdinti(sar *, FILE *);  // Spausdina sąrašą
    void Rikiuoti(sar *);            // Rikiuoja
    int Raktas(kelione, kelione);    // Rikiavimo raktas
    void Rinkti();                   // Atrenkami duomenys
    void Naikinti(sar *);            // Sąrašo šalinimas
};
//                    Pagrindinė programa
void main() {
  FILE *F;
  Keliones A;                        // Klasės Keliones objektas
  sar *T;                            // Pagalbinis kintamasis
  clrscr();
  cout << "Programa darba pradejo\n\n";
```

```
  F = fopen("Kelione.sar", "r");
  if (F == NULL) {
    cout << "Failo 'Kelione.dat' nepavyko atidaryti ";
    exit(1);  }
  A.Formuoti(F);
  fclose(F);

  F = fopen("Kelione.rez", "w");
  if (F == NULL) {
    cout << "Failo 'Kelione.rez' nepavyko sukurti";
    exit(1);  }
  fprintf(F, "Siulomu kelioniu sarasas\n\n");
  T = A.GetP();
  A.Spausdinti(T, F);

  A.Rinkti();
  T = A.GetK();
  A.Rikiuoti(T);
  fprintf(F, "Atrinktu kelioniu sarasas\n\n");
  T = A.GetK();
  A.Spausdinti(T, F);
  fclose(F);

  T = A.GetP();
  A.Naikinti(T);
  T = A.GetK();
  A.Naikinti(T);
  cout << "Pabaiga. Rezultatai faile 'Kelione.rez'";
}
//                    Elemento įrašymas
void Keliones::Padeti(sar **P, kelione N) {
  sar *D = new sar;
  D->S   = N;
  D->sek = *P;
  *P     = D;
}
//                    Sąrašo formavimas
void Keliones::Formuoti(FILE *F) {
  kelione S;
  do {
    fscanf(F, "%s%i%10s%i", S.tipas, &S.trukme, S.vieta,
                                                &S.kaina);
    Padeti(&P, S);  }
```

```
  while(!feof(F));
}
//                    Sąrašo spausdinimas
void Keliones::Spausdinti(sar *P, FILE *F) {
  sar *D = P;
  int i = 1;
  fprintf( F,"+----+-----------+--------+-----------");
  fprintf( F,"+---------+\n" );
  fprintf( F,"| Nr.| Keliones t.+ Trukme +   Vieta    ");
  fprintf( F,"+  Kaina  +\n" );
  fprintf( F,"+----+-----------+--------+-----------");
  fprintf( F,"+---------+\n" );
  while(D) {
    fprintf( F,"| %2d | %10s | %6d | %10s | %6d  |\n",
    i++, D->S.tipas, D->S.trukme, D->S.vieta, D->S.kaina);
    D = D->sek;  }
  fprintf( F,"+----+-----------+--------+-----------");
  fprintf( F,"+---------+\n" );
  fprintf( F, "\n\n\n");
}
//                    Rikiavimo raktas
int Keliones::Raktas(kelione A, kelione B) {
  return(B.trukme < A.trukme) ||
        ((B.trukme == A.trukme ) && (B.kaina > B.kaina));
}
//                    Rikiavimas
void Keliones::Rikiuoti(sar *P) {
  sar *R = P, *D;
  kelione K;
  while(R) {
    D = R->sek;
    while(D) {
      if (Raktas(R->S, D->S))
        { K = R->S;  R->S = D->S;  D->S = K; }
      D = D->sek;  }
    R = R->sek;  }
}
//                    Duomenų atrinkimas
void Keliones::Rinkti() {
  sar *D = P;
  int Sk1, Sk2;
  cout << "Kokia maksimali kaina ?  ";
  cin  >> Sk1;  cout << "\n";
  cout << "Kokia maksimali trukme?  ";
```

```
  cin  >> Sk2;  cout << "\n";
  while(D) {
    if ((D->S.trukme <= Sk2) && (D->S.kaina <= Sk1))
      Padeti(&K, D->S);
    D = D->sek; }
}
//                    Sąrašo pašalinimas
void Keliones::Naikinti(sar *P) {
  sar *R = P, *D;
  while(R) {
    D = R;
    R = R->sek;
    delete(D);  }
}
```

11.2. Užduotys

U1. Turime tekstiniame faile meteorologinių stebėjimų rezultatus:

data, kritulių pobūdis, kritulių kiekis.

Suformuoti sąrašą:

kritulių pobūdis, bendras kiekis, vidurkis.

Sutvarkyti duomenis pagal kritulių pobūdį ir kritulių kiekį. Išvesti tris lenteles: duomenų, sutvarkytų duomenų ir skaičiavimų rezultatų.

U2. Turime tekstiniame faile darbuotojų žiniaraštį:

pavardė, adresas, darbo pavadinimas, data, dirbta valandų.

Kitame tekstiniame faile turime darbų įvertinimo normatyvus:

darbo pavadinimas, valandos įkainis.

Suformuoti atlyginimų sąrašą ir jį sutvarkyti pagal atlyginimo dydį ir pavardę:

pavardė, adresas, dirbtas valandų kiekis, atlyginimas.

Išvesti tris lenteles: dvi duomenų ir skaičiavimų rezultatų.

U3. Turime tekstiniame faile bibliotekos lankytojų sąrašą:

pavardė, vardas, skaitytojo bilieto numeris, knygų paėmimo data, paimtų knygų kiekis, leistinas skaityti dienų skaičius.

Suformuoti skaitytojų, kurie vėluoja grąžinti knygas, sąrašą, ir jį sutvarkyti pagal vėlavimą ir pavardę:

pavardė, vardas, skaitytojo bilieto numeris, vėluojamas dienų skaičius, knygų kiekis.

Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U4. Turime tekstiniame faile studentų sąrašą:

pavardė, vardas, grupės šifras, egzaminų pavadinimai ir pažymiai.

Suformuoti pirmūnų, kurių pažymių vidurkis yra duotame intervale *[a, b]*, sąrašą ir jį sutvarkyti pagal grupes ir pavardes:

pavardė, vardas, grupės šifras, pažymių vidurkis.

Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U5. Turime tekstiniame faile studentų sąrašą:

pavardė, vardas, grupės šifras, visų egzaminų pažymiai.

Suformuoti sąrašą:

pavardė, vardas, grupės šifras, pažymių vidurkis.

Jį sutvarkyti pagal grupės šifrą ir pažymių vidurkį. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U6. Turime tekstiniame faile studentų sąrašą:

pavardė, vardas, grupės šifras, visų egzaminų pažymiai.

Suformuoti sąrašą:

pavardė, vardas, grupės šifras, kiek kokių pažymių turi.

Jį sutvarkyti pagal grupės šifrą ir pažymių vidurkį. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U7. Turime tekstiniame faile studentų sąrašą:

pavardė, vardas, grupės šifras, visų egzaminų pažymiai.

Suformuoti sąrašą:

pavardė, vardas, grupės šifras, mokymosi vidurkis.

Jį sutvarkyti pagal grupės šifrą ir mokymosi vidurkį. Išvesti lenteles: duomenų ir skaičiavimų rezultatų, pirmūnų (mokosi vidurkiu >8) ir atsiliekančių (mokosi vidurkiu <6).

U8. Turime tekstiniame faile studentų sąrašą:

pavardė, vardas, grupės šifras, visų egzaminų pažymiai.

Suformuoti sąrašą:

pavardė, vardas, grupės šifras, aštuntukų,devintukų ir dešimtukų kiekiai.

Jį sutvarkyti pagal aštuntukų, devintukų, dešimtukų kiekį. Išvesti lenteles: duomenų ir skaičiavimų rezultatų.

U9. Turime tekstiniame faile studentų darbo prie kompiuterių apskaitos sąrašą:

pavardė, vardas, grupės šifras, kompiuterio registracijos numeris, data, darbo pradžios laikas, darbo pabaigos laikas.

Suformuoti sąrašą:

pavardė, vardas, grupės šifras, kompiuterio registracijos numeris, data, darbo laikas minutėmis.

Jį sutvarkyti pagal dirbtą laiką ir pavardę. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U10. Turime tekstiniame faile studentų apskaitos sąrašą:

pavardė, vardas, įstojimo į Universitetą data, visų egzaminų pažymiai.

Suformuoti sąrašą:

pavardė, vardas, kursas, aštuntukų, devintukų, dešimtukų kiekiai.

Jį sutvarkyti pagal dešimtukus, devintukus, aštuntukus. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U11. Turime tekstiniame faile studentų sąrašą:

pavardė, vardas, grupės šifras, gimimo data, visų egzaminų pažymiai.

Suformuoti sąrašą:

pavardė, vardas, amžius, kiek kokių pažymių turi.

Jį sutvarkyti pagal amžių ir mokymosi vidurkį. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U12. Turime tekstiniame faile viešbučio gyventojų sąrašą:

pavardė, vardas, atvykimo data, išvykimo data, kaina už parą.

Suformuoti sąrašą:

pavardė, vardas, atvykimo data, išvykimo data, mokestis.

Duomenis ir rezultatus sutvarkyti pagal atvykimo datą. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U13. Turime tekstiniame faile viešbučio gyventojų sąrašą:

pavardė, vardas, atvykimo data, išvykimo data, paros kaina.

Suformuoti sąrašą:

pavardė, vardas, atvykimo data, gyventa parų, mokestis

Jį sutvarkyti pagal gyvenimo trukmę ir mokestį. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U14. Turime tekstiniame faile viešbučio gyventojų sąrašą:

pavardė, vardas, atvykimo data, gyventa parų, paros kaina.

Suformuoti sąrašą:

pavardė, vardas, atvykimo data, išvykimo data, mokestis.

Jį sutvarkyti pagal pavardes ir vardus. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U15. Turime tekstiniame faile viešbučio gyventojų sąrašą:

pavardė, vardas, atvykimo data, gyventa parų, kambario tipas.

Kitame tekstiniame faile turime kainoraštį:

kambario tipas, paros kaina.

Suformuoti sąrašą:

pavardė, vardas, atvykimo data, išvykimo data, mokestis.

Jį sutvarkyti pagal mokestį. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U16. Turime knygų sąrašą:

autorius, pavadinimas, išleidimo metai, tiražas, kaina.

Suformuoti sąrašą:

autorius, pavadinimas, išleidimo metai, tiražo piniginė vertė.

Sąrašą sutvarkyti pagal išleidimo datą. Išvesti lenteles: duomenų ir skaičiavimų rezultatų.

U17. Turime tekstiniame faile viešbučio gyventojų sąrašą:

pavardė, vardas, atvykimo data, gyventa parų, kambario tipas.

Kitame tekstiniame faile turime kainoraštį:

kambario tipas, kaina, gyvenimo kaina,
buitinių paslaugų kaina, pelno procentas.

Suformuoti sąrašą:

pavardė, vardas, atvykimo data, gyventa parų, mokestis.

Jį sutvarkyti pagal atvykimo datą. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U18. Turime sąrašą:

detalė, pagaminimo data, kiekis.

Kitame tekstiniame faile turime kainoraštį:

detalė, gamybos išlaidų kaina, pelno procentas.

Suformuoti sąrašą:

detalė, pardavimo kaina, kiekis.

Jį sutvarkyti pagal kainą ir kiekį. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U19. Turime sąrašą:

detalė, pagaminimo data, kiekis.

Kitame tekstiniame faile turime kainoraštį:

detalė, medžiagų kaina vienete, darbo sąnaudos vienetui,
kitos papildomos išlaidos vienos detalės gamybai.

Suformuoti sąrašą:

detalė, kiekis, pardavimo kaina, priskaičiuotas atlyginimas.

Jį sutvarkyti pagal kiekį ir atlyginimą. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U20. Turime sąrašą:

detalė, parduotas kiekis, pardavimo data.

Kitame tekstiniame faile turime kainoraštį:

detalė, kaina.

Suformuoti sąrašą:

detalė, pardavimo data, pardavimo kaina, suma.

Jį sutvarkyti pagal datą ir kainą. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U21. Turime sąrašą:

prekė, gautas kiekis, parduotas kiekis, pardavimo data.

Kitame tekstiniame faile turime kainoraštį:

prekė, kaina.

Suformuoti sąrašą:

prekė, likutis, pardavimo kaina, pajamos.

Jį sutvarkyti pagal kainą ir pajamas. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.

U22. Turime sąrašą:

prekė, pagaminimo data, vartojimo laikas.

Suformuoti sąrašą:

prekė, pagaminimo data, vartojimo laikas, vartojimo pabaigos data.

Jį sutvarkyti pagal pagaminimo datą ir pavadinimą. Išvesti dvi lenteles: duomenų ir skaičiavimų rezultatų.