Įžanga

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, sugeba naudoti standartines duomenų struktūras, valdančias struktūras, programavimo aplinkas.

1 tema. C++ elementai

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 kalbos realizacijos turi labai įvairias ir galingas bibliotekų sistemas;

✧ 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                         */
main()
      {
          /*    Pagrindinės funkcijos tekstas   */
      }
        /*    Vartotojo 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, tačiau 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 grąžina funkcijai, kuri analizuoja atsakymą ir išveda ekrane pilną pranešimą.

```
// Pratimas1              Programos struktura
#include <iostream.h>  // Įvedimo/išvedimo srautai
#include <conio.h>     // Tekstinio ekrano tvarkyklė

void Atsakymas( int ); // Funkcijos prototipas

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;
}//-----------------------------------------------
void Atsakymas( int Ats ){          // Funkcija
  if ( Ats == 1 )
       cout<<" giedras dangus!!\n";
  else cout<<" Dangje 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).

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:

`int` - sveikasis,

`float` - realusis,

`char` - simbolinis.

Tokio aprašo pavyzdys programoje 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.

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

```
const  int     A = 125;
const  float   B =  13.5;
```

Pagrindiniai tipai.

Tipas	Galimos reikšmės	
`char`	-128..127	simbolinės reikšmės

`int`	-32768..32768	sveikieji skaičiai
`long`	-2147483648..2147483647	sveikieji skaičiai
`float`	$-3.4\times10^{-38}..3.4\times10^{38}$	realieji skaičiai
`double`	$1.7\times10^{-308}..1.7\times10^{308}$	realieji skaičiai

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
```

Pagrindiniai operatoriai.

Priskyrimo sakinyje panaudojome opratorių **=** (lygybės ženklas). Kiti dažniausiai vartojami parodyti 1.1 lentelėje. Operacijoms žymėti naudojami ženklai surašyti 1.2 lentelėje.

1.1 lentelė. Populiariausi 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;

1.2 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`**.

Funkcijos.

Pirmo pratimo programos 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 valdymo operatoriai ir komentarai.

Kintamųjų galiojimo sritys.

Kintamasis galioja nuo 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 nerekomenduotinas programavimo stilius.

Duomenų įvedimas/išvedimas

Programoje duomenų įvedimui bei išvedimui panaudojamas išorinių srautų bibliotekos `iostream.h` nukreipimo operatoriai << ir >> aprašant programos ir vartotojo dialogą. 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 (1.3 lentelė).

Jų sintaksė: **\⟨simbolis⟩**

1.3 lentelė. Valdančių simbolių pavyzdžiai

Pavadinimas	C kalboje	ASCII kodas
Skambutis	\a	7
Eilutės pradžia	\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 ir galioja tik vienam į kintamajam:

setw(n) **n –** lauko plotis simboliais,
setprecision(n) **n –** slankaus kablelio skaičių tikslumas.

Jei **setw** nurodyto lauko dydžio skaičiui nepakanka, manipuliatorius ignoruojamas. Formatuoto išvedimo pavyzdys:

```
cout<< "x = " <<setw(10) <<x;       //  čia  int x;
                                    //  float a;
cout<< "a = " <<setw(10) <<setprecision( 3 ) <<a;
```

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;

Valdymo struktūra if.

Programoje panaudota klasikinė visose programavimo kalbose vartojama valdymo struktūra - operatorius **if.** Jo užrašymo sakinio sintaksė:

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

Paprasčiausios sąlygos yra aprašomos tradiciniu būdu - santykiais, kuriuose vartojamos sulyginimo ir loginės operacijos.

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 (≠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 gdangus!!\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 `else` šaka bus vykdoma visais atvejais, išskyrus atsakymą lygų 1.

2 tema. Masyvai, failai, eilutės

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 saugomis 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 nurodomo 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`**.

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 elemetai. Masyvo narius žymi kintamieji su indesais, kurių vardų sintaksė:

⟨masyvo vardas⟩[⟨indeksas⟩]

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

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

☞ Masyvo vardas be indekso reiškia jo nulinio elemento adresą.

2.1 pratimas. Programa demonstruoja situaciją, kai gretimuose atminties laukuose surašyti vienodo tipo duomenys (a, b, c, d) gali būti laikomi masyvu, kurio pradžią nurodo pirmojo kintamojo duomenų adresas: `pi = &a;`

```
//    Pratimas 2
#include <iostream.h>
void pabaiga();
int  a=1,  b=2,  c=3,  d=4,   *pi;
main() {
   int  i= 0,    mas[4];
   pi= &a;                      // Kintamojo a adresas
                                // Masyvo formavimas
   while ( i<4 ) mas[i++]= *(pi++);
       // Masyvo išvedimas į ekraną atvirkščia tvarka
   while (i)     cout<< mas[--i] << "\t";
   cout<<endl;
   pabaiga();                  // Ekrano užlaikymas
}
void pabaiga(){
 char simbolis;
 cout << endl
 << "Paspauskite bet koki simbolini klavisa ir ENTER"
      << endl;
 cin >> simbolis;
}
```

Indekso reikšmė cikle yra keičiama sveiko tipo kintamiesiems taikoma reikšmės didinimo operacija (1 lentelė), kuri gali būti užrašoma dviem būdais:

```
<kintamasis>++
++<kintamasis>.
```

Pirmasis variantas nurodo, kad kintamojo reikšmė iš pradžių turi būti panaudojama programos struktūroje, o po to didinama **1**, o antrasis variantas,- kad reikšmė iš pradžių padidinama ir tik po to vartojama programoje.

Kai reikšmių didinimo ir 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.

Cikliniam indekso ir rodyklės reikšmių keitimui programoje yra vartojamas ciklas `while`, kurio sintaksė:

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

Panašus yra ciklas `do`, kuris visuomet vykdomas bent vieną kartą. Jo aprašo sintaksė:

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

Labai patogus yra `for` ciklas, kurio strktūra:

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

Sakinyje nurodyta išraiška **i1** skaičiuojama tik vieną kartą, prieš pradedant ciklą, o išraiškos **i2** reikšmė perskaičiuojama po kiekvieno ciklo. Tai ciklo kartojimo sąlyga. Paprastai **i1** apibrėžia ciklo parametro pradinę reikšmę, o **i3** - jo kitimo dėsnį. Pavyzdžiui, analizuojamoje programoje masyvą **mas** galima formuoti tokiu ciklu:

```
for (i=0;  i<4;   i++)  mas[i]= *(pi++);
```

✓Papildykite programą taip, kad ji parodytų ekrane visų joje vartojamų kintamųjų ir masyvo **mas** elementų adresus. Aprašant adresų išvedimą, yra vartojamas rodyklių šablonas **%p**. Adresai yra išvedami šešioliktainėje skaičiavimo sistemoje.

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. Funkcijos prototipe nurodomi rodyklės tipo parametrai, pavyzdžiui:

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

Čia pirmasis parametras rodo, kad kreipinio metu reikia perduoti integer 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 );
```

2.2 pratimas. Funkcija `Keisti` demonstruoja abi parametrų formas. Pagrindinė funkcija parodo ekrane gaunamus rezultatus. Funkcija `getch` panaudota programos veiksmams sustabdyti: galima stebėti ekrane esamą situaciją. Ši funkcija reaguoja į klavišo paspaudimą.

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

void  Keisti( int *, int );

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";
         getch();
 }
void Keisti ( int *p1,  int p2 )
{  *p1 = *p1 + 10;   p2 = p2 + 10;  }

/*  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
*/
```

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 );

void main() {
 int  A[10] ; int 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 );
 cout << endl;
}
int Suma( int * pr,  int k )
{  int S = 0,  i = 0;
   while ( i<k ){  S = S + *(pr+i);  i+=1;  }
   return S;
}
```

Dinaminis atminties išskyrimas.

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

```
         #include <stdlib.h>
arba     #include<alloc.h>
  void *malloc( Kiekis );
```

Čia nurodomas prašomos atminties `Kiekis` baitais. Jeigu atmintis buvo skirta, tai grąžinama rodyklė į pirmąjį tos atminties baitą, kitaip rodyklės 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 kintamuoju 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
//-------------------------------------------------
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 užimata atmintis atlaisvinama
 cout << endl;
}//-------------------------------------------------
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;
}//-------------------------------------------------
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;      }
}//-------------------------------------------------
void  Sp      ( int * x,  int n ){
  int  i = 0;
  while ( i<n ){
    cout << (i+1) <<"-elementas : " << *( x+i )
         << endl;
    i++ ;   }
}//-------------------------------------------------
```

✓ Patogu naudoti operatorių **`new`** atminties skyrimui. Pakeiskite 2.4 pratimo funkcijos `Duomenys` eilutę:

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

nauja eilute: `x = new  int [n];`

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

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];
```

Čia sukurti fiksuotos apimties masyvų tipai. Dabar galima aprašyti keletą to paties tipo masyvų:

```
masyvas  A, B, C;
matrica   P, D, R;
```

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

```
//  Matrica
#include <iostream.h>
#include <alloc.h>
typedef   int   mas [15];
typedef   int   matr[10][15];
                              // Duomenų įvedimas
int   Duomenys( matr ,  int *, int *);
void  Sp      ( mas  ,  int     );  // Spausdinimas
int  Suma    ( mas ,   int); // Masyvo elementų suma
//-------------------------------------------------
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 spaudinimas
} cout << endl;
}//-------------------------------------------------
int  Duomenys( matr  D, int * n, int * m  ){
  int i, j, *x;
  cout <<"Iveskite n reiksme: ";    cin>> *n;
  cout <<"Iveskite m reiksme: ";    cin>> *m;

  x = (int *) malloc(  (*n * *m) *  sizeof( int ));
  if ( x == NULL )
           { cout <<" Truksta atminties" << endl;
             return 0;   }
  else {
  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;}
}//-------------------------------------------------
void  Sp  ( mas x,  int n ){
  int  i = 0;
    while ( i<n  )   cout <<  x[i++] << "  ";
    cout << endl;
 }//-------------------------------------------------
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];
                        }
}
```

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 naujai.

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

2.6 pratimas. Duomenų faile ”duom7.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>
int       Kiek = 15;
typedef   int   mas [15];
FILE *F;

void  Sp   ( mas ,  int ); // Spausdinamas masyvas
int   Suma ( mas ,  int ); // Sumuojami masyvo elementai
//-------------------------------------------------
void main() {
  mas A;         int n = 0;

  F = fopen( "duom7.dat", "r" );
  if ( F == NULL ) printf( "Failas neatidarytas");
  else{
       while( !feof(F) && ( n< Kiek)) {
          fscanf( F, "%d", &A[n]);  n += 1; }
       Sp( A, n );
       printf( "Suma= %5d \n Pabaiga", Suma( A, n ));
      }  getch();
} //-----------------------------------------------
void  Sp      ( mas X,  int n ){
      int  i = 0;
  while ( i<n  )   cout <<  X[i++] << "  ";
  cout << endl;
 }//-----------------------------------------------
int  Suma(  mas R, int k ){
     int i = 0,     S = 0;
  while( i<k ) S += R[i++];
  return S;
}//-----------------------------------------------
```

Programoje panaudotas formatinis duomenų skaitymas iš failo ir spausdinimas ekrane:

```
#include <stdio.h>
int fscanf(FILE *stream,
```

```
          const char *format[, address, ...]);
int printf(const char *format[, argument, ...]);
```

2.7 pratimas. Duomenų faile "duom8.dat" surašyti skaičiai. Reikia duomenis iš failo surašyti į masyvą. Masyvo elementų reikšmės atspausdinamos kitame faile.

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

int       Kiek = 10;
typedef   int   mas [10];
                // Masyvo elementų išvedimas į failą
void  SpF    ( mas ,  int, char );
int   Ivesti ( mas ,  int *   );// Skaitymas iš failo
//---------------------------------------------
void main() {
  mas A;  int n = 0;
  if ( Ivesti ( A, &n ))
     { cout<<"\nDuomenų failo nėra\n";
       getch(); exit( 1 );  }
  SpF( A, n, 'A' );
  getch();
} //---------------------------------------------
int   Ivesti    ( mas A,  int *n ){
 FILE *F;
 F = fopen( "duom8.dat", "r" );
 if ( F == NULL ){ printf( "Failas neatidarytas");
                   return 1; }
 else{
   while( !feof(F) && ( *n< Kiek)) {
      fscanf( F, "%d", &A[*n]);  *n += 1; }
   fclose( F );  return 0; }
}//---------------------------------------------
void  SpF    ( mas X,  int n , char R ){
  FILE *F;
  int  i = 0;
  F = fopen( "duom8.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: **Duom8.dat**

```
5  6 98  -5  12  3 4 5
21  -3  4 5 23 1
23 34 45 5 6 -123
```

rezultatų failas: **Duom8.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.

```
//  Matrica
#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   = "Duom9.dat";
const     char *Rezultatai = "Duom9.rez";

void RezFailas();       // Rezultatų failo paruošimas.
int  Failas();          // Randa kiek yra duomenų.
mas *Ivesti(  int );    // Duomenų įvedimas.
                        // Rezultatų spausdinimas.
void Spausdinti( mas *, int, int );
void Atstumai( mas *, int );  // Taškų atstumai.
void Vieta( mas *, int );   // Taškų padėtis plokštumoje.
//---------------------------------------------------
  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 );
  getch();
}//-----------------------------------------------
// 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 koordimates (duomenis);
kai 1, spaudina 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, "*********************\n");
         fprintf( F, "  Nr.   x        y   \n");
         break;
 case 1:
     fprintf( F, "*******************************\n");
     fprintf( F, "  Nr.   x        y   Atstumas \n");
     break;
 case 2:
   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 (nulis), 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
*********************
  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
*********************
*****************************
  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
*****************************
************************************
  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
************************************
```

Simbolių eilutės.

Simbolių masyvai gali būti vartojami simbolių eilutėms apdoroti. Naudojant masyvus simboliams saugoti, reikia atsiminti, kad C kalboje eilučių reikšmių pabaigas primta 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".

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

```
// Eilutės
#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;
  getch();
}
```

✓ 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

2.10 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.

```
// Eilutės
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
typedef char Mas[ 80 ];
void zodis( Mas, Mas, int *);   // Žodžio atskyrimas
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' ){// Žožių atskyrimas:
   c = i;               //   žodžio pradžia;
               // surastas žodis pradedant i vieta;
   zodis( Eil, Z, &i );
   if ( Z[0] != '\0' )
             // jeigu buvo žodis, tai spausdinamas
     cout<<" Pradžia: "<<c << " Pabaiga: "<< (i-1)
         <<" *"<<Z<<"*"<<endl;
     i++;                //   Praleidžiamas tarpas.
 }
  getch();
}//-----------------------------------------------
/* Iš eilutės A, pradedant simboliu nr. n iškiriamas
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).

2.11 pratimas. Patikrinkite programą, kuri ekrane praneša apie pirmą raidės 'a' poziciją argumento eilutėje. Pozicijos numeruojamos pradedant nuo nulio.

```
#include <iostream.h>
#include <string.h>
#include <conio.h>
main()    {
  char string[30];
  char *ptr, c = 'a';
  strcpy(string, "Čia yra trumpa eilutė");
  ptr = strchr(string, c);
  if (ptr)
 cout<<"Simbolio "<< c <<" pozicija yra "
     << (ptr-string)<< endl;
 else   cout<<"Simbolio nėra\n";
  getch();
 }
```

Programoje vartojamos eilučių tvarkymo funkcijos.

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

Kopijuoja į rodyklės **dest** rodomą vietą eilutę **scr**.

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

Grąžina rodyklės į simbolio **c** poziciją eilutėje **s** reikšmę.

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

2.12 pratimas. Žodžių atskyrimui iš eilutės, kai yra žinoma skirtukų aibė, pritaikyta funkcija

strtok: **`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. Funkcijos taikymo pavyzdys:

```
#include <string.h>
#include <iostream.h>
#include <conio.h>
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
   getch();
}
```

✓ 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.

2.13 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, " ");  strcat(z, x); // Eilučių sujungimas
   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;
    getch();
}
```

✓ 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.

Naudingos funkcijos.

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

Eilutę `src` kopijuoja į `dest` iki nulinio simbolio. Grąžina rodyklė 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);`

Elutės `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.

```
#include <string.h>
#include <stdio.h>
int main(void)
{ char string[15];
  char *ptr, c = 'r';
    strcpy(string, "This is a string");
    ptr = strchr(string, c);
    if (ptr)
      printf("The character %c is at position: %d\n",
                 c, ptr-string);
    else
        printf("The character was not found\n");
   return 0;
}
```

```
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 mažasias ir didžiasias 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.

```
#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;
}
```

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

Kompiuterio laiką užrašo eilutėje buf, kurios ilgis privalo būti **9** Eilutės forma **HH:MM:SS,** kur HH –valandos, MM – monutės 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.

```
#include <time.h>
#include <stdio.h>
void main(void) {
  char datebuf[9];
  char timebuf[9];
  _strdate( datebuf );
  _strtime( timebuf );
  printf("Date: %s  Time: %s\n",datebuf,timebuf);
}
```