9. EILUTĖS. STRUKTŪROS

Šiame skyriuje kalbama apie nestandartinius C++ kalbos duomenų tipus – eilutes ir struktūras. Eilutės duomenyse saugomas tekstas, o anot kai kurių programavimo autoritetų (R. Lafore) apie 70 % visų programų skirtos būtent darbui su tekstu. Su tokiais duomenimis negalima atlikti jokių aritmetinių operacijų, tačiau teksto analizės ar apdorojimo veiksmų yra daug: norimų simbolių atpažinimas tekste, jų vietos radimas; simbolių sekų atpažinimas ir jų vietos radimas, simbolių sekų sulyginimas pagal tam tikrus požymius, simbolių ar simbolių sekų šalinimas iš teksto arba jų pakeitimas kitais simboliais, simbolių ar simbolių sekų įterpimas į tekstą norimoj vietoj, simbolių sekų kopijavimas ir pan. Kai kuriuos iš tų veiksmų šiame skyriuje iliustruosime.

Struktūros – paties programuotojo kuriamas, iš vidinių C++ duomenų sudarytas duomenų formatas. Tuo būdu, tai yra komponentinis duomenų formatas. Viena struktūra gali turėti komponentą – kitą struktūrą ir t.t. Susipažinsime su sintakse, taikoma programose kuriant struktūras.

9.1 EILUTĖS

Eilutės realizuojamos trimis skirtingais būdais:

- *char* formato masyvu
- bibliotekine C++ klase *string* (reikia antraštinio failo *<string>*)
- *char** formato rodykle

Šiame skyriuje nagrinėsime pirmąjį eilutės duomenų realizavimo būdą. Prireikus, kai jau žinosite klases, su antruoju būdu bus nesunku susipažinti patiems: tereikia tik išnagrinėti antraštiniame faile aprašytus klasės *string* konstruktorius ir metodus. Keli programų pavyzdžiai su trečiuoju teksto realizavimo būdu yra pateikti rodyklių skyriuje.

Su eilutės duomenimis-simbolių masyvu susidūrėm jau 8-ojo skyriaus 2-ajame programos pavyzdyje, kur duomenų ir rezultatų failų pavadinimai buvo talpinami vienmačiuose *char* masyvuose. Taigi, matėme, kad vienas simbolis saugomas kaip vienas masyvo elementas ir užima vieną baitą atminties. Masyvas visada baigiamas masyvo pabaigos požymiu – valdančiąja seka *\0,* todėl masyvo ilgis bent vienetu turi būti ilgesnis nei jame yra prasminių simbolių.

Tokį masyvą galima paskelbti, apibrėžti ir inicializuoti reikšme „Eilutinis duomuo“ tiesiog programos tekste mums jau žinoma masyvų sintakse taip:

char e1[ ] = { ‘E‘,‘i‘,‘l‘,‘u‘,‘t‘,‘i‘,‘n‘,‘i‘,‘s‘,‘ ‘,‘d‘,‘u‘,‘o‘,‘m‘,‘u‘,‘o‘,‘\0‘ };

Tai labai nepatogu, todėl yra ir paprastesnė sintaksė:

char e2[20] = “Eilutinis duomuo“; arba tiesiog
char e3[ ] = “Eilutinis duomuo“;

Masyvo pabaigos simbolis bus įrašytas automatiškai. Masyve *e2* liktų 3 neinicializuoti galiniai elementai (tekstas sudarytas iš 16 simbolių ir dar vieną masyvo elementą užims pabaigos simbolis *\0*). Masyvui *e3* kompiliatorius skirs 17 ląstelių atminties.

Simbolių masyvą įvedant operacija >>, kaip minėtoje 8-ojo skyriaus programoje, svarbu, kad tarp simbolių nebūtų intervalo simbolių: sutiktas intervalas iškart nutraukia įvedimą. Jei intervalų ir nėra, toks įvedimas vis tiek yra vengtinas: tada bus įvesti visi simboliai iki *\0* simbolio, nežiūrint ar masyve dar yra jiems vietos, ar ne. Taip yra dėl to, kad C++ kompiliatorius netikrina, ar masyvo

ribos viršijamos. Toks dalykas neišvengiamai sukeltų neprognozuojamo pobūdžio vykdymo meto klaidą. Nuo šių atminties klaidų galima apsisaugoti įvesties metu teikiant manipuliatorių *setw* su nurodytu pozicijų skaičiumi *n+1*, kur *n* yra maksimalus simbolių eilutėje skaičius, o dar vienas simbolis skiriamas masyvo pabaigai. Tuo būdu, programos fragmentas turėtų būti perrašytas taip:

```
. . .
char fvd[ 16 ], fvr[ 16 ];  // failų pavadinimai iš ne daugiau 15 simbolių
ifstream is;
ofstream os;
    //
cout<<“Iveskite duomenu failo varda (ne daugiau 15 simboliu)\n“;
cin>>setw(15)>>fvd;
cout<<“Iveskite rezultatu failo varda (ne daugiau 15 simboliu)\n“;
cin>>setw( 15 )>>fvr;
. . .
```

Jei į masyvą reikia įvesti tekstą su intervalo simboliais – teks naudoti 8-ame skyriuje aprašytą *istream* klasės metodą *get(sm, n),* kur *sm* yra eilutės duomens-masyvo pavadinimas, o *n* – duomens simbolių skaičius + 1. Jei tekste yra simbolių *n*, t.y. tekstas fiziškai išdėstytas per kelias eilutes, reikės to paties perkrauto metodo su trim argumentais *get(sm, n, sk),* kur *sk* yra paties programuotojo pasirinktas skyriklio simbolis, užbaigiantis tekstą. Metodo taikymo pavyzdys-schema:

```
. . .
char e[ 101 ];
cout<<“Iveskite teksta iki 100 simboliu\n“;
cin.get( e, 101, ‘@‘ );                    // pasirinkome skyrikliu @
cout<<“Ivestas tekstas:\n“<<e<<endl;
. . .
```

Šiam fragmentui parašius pilną programą ir įvedus tekstą

```
Eilutes duomuo
per
kelias
eilutes@
```

bei nuspaudus klavišą *Enter*, matysite į ekraną išvestą tiksliai tokią duomens reikšmę be simbolio @.

Eilučių masyvas, išvedant analogiją su aritmetinių duomenų dvimačiu masyvu, gali būti realizuotas kaip dvimatis *char* formato masyvas. Kaip ir vienmačio *char* masyvo atveju, vėl nereiks paelemenčiui suteikti masyvui reikšmių – yra supaprastinta sintaksė. Sakykim, norim suformuoti eilučių masyvą iš savaitės dienų pavadinimų lotyniškais simboliais. Tokiam masyvui reikės skirti 7 eilutes – kiekviena eilutė vienam pavadinimui – ir tegu 15 stulpelių: nė vienas dienos pavadinimas nebus ilgesnis nei 14 simbolių. Programos fragmentas rodo, kaip tokiam masyvui priskirti reikšmes

tiesiog programos tekste ir kaip masyvo reikšmes išvesti į ekraną (ar į failą). Įvesti iš ekrano (failo) reikėtų analogiškai.

```
. . .
char sd[ 7 ][ 15 ] = { “Pirmadienis“,  “Antradienis“,  “Treciadienis“,
                       “Ketvirtadienis“,  “Penktadienis“,  “Sestadienis“,
                       “Sekmadienis“  };
. . .
for( int i = 0; i < 7; i++ )
        cout<<sd[ i ]<<endl;  // kreipinys sd[ i ] apima visą eilinę masyvo eilutę
. . .
```

Tekstui apdoroti yra funkcijų biblioteka *cstring* (norint ja naudotis, į programą teks įterpti antraštinį failą *<cstring>*), kurioje yra visiems gyvenimo atvejams reikalingos teksto apdorojimo funkcijos. Jos dažniausiai grąžina rodykles į eilutės duomenį ir reikalauja argumentų-rodyklių, todėl, kol rodyklių dar nežinom, tikslių funkcijų prototipų nepateiksim, tik parodysim, kaip tomis funkcijomis galima naudotis. Čia yra apibūdintos kelios tipinės bibliotekos funkcijos:

strlen(sm) – grąžina eilutės duomens *sm* ilgį baitais be duomens pabaigos simbolio

strcpy(smd, sms) – nukopijuoja eilutės duomenį *sms* į eilutės duomenį *smd*

strncpy(smd, sms, n) – tas, kas *strcpy*, tik kopijuoja *n* simbolių

strcat(smd, sms) – prie duomens *smd* prijungia *sms*

strcmp(sm1, sm2) – lygina eilutes *sm1* ir *sm2* pagal ASCII kodus. Jei jų simbolių kodai sutampa, grąžina *0*; jei pirmojo duomens kodai didesni už antrojo – grąžina teigiamą skaičių, ir atvirkščiai. Funkcijos reikšmę lemia pirmiausia pirmųjų simbolių lyginimas, jei jie lygūs einama prie antrųjų simbolių lyginimo ir t.t. Pavyzdžiui, *jei sm1[ ] = “ABCD“; sm2[ ] = “abcd“;* - funkcijos rezultatas yra *-1*. Ta pat reikšmė gaunama eilutėms *sm1[ ] = “ABCD“; sm2[ ] = “ABcd“;* ir *sm1[ ] = “ABCD“; sm2[ ] = “ABef“;* o reikšmėms *sm1[ ] = “EBCD“; sm2[ ] = “ABef“; – +1.*

strncmp(sm1, sm2, n) – tas pat, kas *srtcmp(sm1, sm2)*, tik lyginami pirmieji *n* simbolių; n – tik teigiamas

stricmp(sm1, sm2) – tas pat, kas *srtcmp(sm1, sm2)*, tik lyginant raidinius simbolius didžiosios ir mažosios raidės laikomos esant lygiais simboliais

strnicmp(sm1, sm2, n) – tas pat, kas *srticmp(sm1, sm2)*, tik lyginami pirmieji *n* simbolių; n – tik teigiamas

strlwr(sm) – eilutėje *sm* visas didžiąsias raides paverčia mažosiomis

strupr(sm) – eilutėje *sm* visas mažąsias raides paverčia didžiosiomis

strset(sm, s) – eilutę *sm* užpildo simboliais *s*

strchr(sm, s) – ieško eilutėje *sm* simbolio *s*. Jei simbolis randamas – grąžina rodyklę į tą eilutės elementą, jei ne – nulinę rodyklės reikšmę

strrchr(sm, s) – ieško eilutėje *sm* paskutiniojo simbolio *s*. Jei simbolis randamas – grąžina rodyklę į tą eilutės elementą, jei ne – nulinę rodyklės reikšmę

. . .

Toliau parašysime dvi programas, apdorojančias tekstą: pirmoji kopijuos vienos eilutės reikšmę į kitą eilutę, antroji duotąjį tekstą pavers į tekstą vien tik didžiosiomis arba vien tik mažosiomis raidėmis. Nepaisant to, kad tiksliai tokius veiksmus gali atlikti vienos iš aukščiau

išvardintų funkcijų, patys pabandysime užprogramuoti šiuos veiksmus. Vis tik, siekiant universalesnio programų pobūdžio, teks taikyti vieną iš tų funkcijų – *strlen* – eilutės ilgiui nustatyti.

1 pavyzdys. Eilutės reikšmė kopijuojama į kitą eilutę ir išspausdinama į ekraną. Pradinė eilutė – ne ilgesnė nei 79 simboliai.

```
#include <iostream>
#include <cstring>
using namespace std;
//
int main( ) {
        char e1[ ] = "Pirmoji eilute";
        char e2[ 80 ];
        //
        unsigned int i;
        for( i = 0; i < strlen( e1 ); i++ )
            e2[ i ] = e1[ i ];
        e2[ i ] = '\0';   // 1
        //
        cout<<"Antroji eilute  "<<e2<<endl;
        //
        system("pause");
        return 0;
}
```

Gali kilti klausimas, kam reikalingas komentaru 1 pažymėtas operatorius? Reikalingas dėl to, kad funkcija *strlen* įskaičiuoja tik reikšminius simbolius išskirdama galinį simbolį *\0*. Taigi, galinį simbolį į antrąją eilutę teks įrašyti išreikštai. O indekso reikšmė už ciklo ribų turi būti *i*, nes iš ciklo išeinama, kai *i* reikšmė vienetu didesnė už *strlen(e1)*.

2 pavyzdys. Tekstas keičiamas į tekstą vien didžiosiomis ar mažosiomis raidėmis; kiti simboliai nepertvarkomi. Tekstas spausdinamas į ekraną atskira funkcija. Kad galėtume užprogramuoti šiuos veiksmus, reikia prisiminti, kad C++ naudoja ASCII lentelės simbolius, o *char* formato duomens ląstelėje saugomas ne pats simbolis, bet jo eilės numeris ASCII lentelėje. Šią lentelę galima rasti bet kuriai programavimo kalbai skirtoje knygoje. Joje iš pradžių sudėti specialieji simboliai, po to – didžiosios lotyniškos raidės, toliau – mažosios. Kad tą aiškiai pamatytumėt, programos pradžioje išspausdinami simbolių A, a ir intervalo simbolio eilės numeriai bei skirtumas tarp a ir A simbolių (šis skirtumas toks pat ir kitoms atitinkamoms mažosioms ir didžiosioms raidėms). Pradinis tekstas pateikiamas pačioje programoje angliška fraze.

```
#include <iostream>
#include <cstring>
using namespace std;
//
void showString( char[ ], char[ ] ); // prototipas funkcijos eilutės spausdinimui
void toUpperCase( char[ ] , int );  // funkcijos eilutei pertvarkyti į didžiąsias raides
void toLowerCase( char[ ], int );   // funkcijos eilutei pertvarkyti į mažąsias raides
```

```
//
int main( ) {
   char text[ ] = "This is a Mixed Case";  // pradinis tekstas
                //
   int a = 'a',   A = 'A',    space = ' ', difference = a - A;  // simbolių aritmetiniai kodai
   cout<<"ASCII codes for a, A and ' '   "<<a<<"  "<<A<<"  "<<space
      <<endl<<"  a - A = "<<difference<<endl<<endl;
     //
   showString( "Initial text", text );
   toUpperCase( text, difference );
   showString( "Changed to upper case", text );
   toLowerCase( text, difference );
   showString( "Changed to lower case", text );
     //
   system("pause");
            return 0;
}
//
void showString( char t[ ], char s[ ] ) {
            for( unsigned int i = 0; i < strlen( t ); i++ )
                  cout<<t[ i ];
            cout<<":  ";
            for( unsigned int i = 0; i < strlen( s ); i++ )
                  cout<<s[ i ];
            cout<<endl<<endl;
}
//
void toUpperCase( char s[ ], int d ) {
            for( unsigned int i = 0; i < strlen( s ); i++ )
                  if( s[ i ] >= 'a' )
                        s[ i ] -= d;
}
//
void toLowerCase( char s[ ], int d ) {
            for( unsigned int i = 0; i < strlen( s ); i++ )
                  if( s[ i ] >= 'A'  && s[ i ] <= 'Z' )
                        s[ i ] += d;
}
```

Programos spausdiniai:

```
ASCII codes for a, A and ' '   97  65  32
  a – A =  32

Initial text:  This is a Mixed Case
```

Changed to upper case: THIS IS A MIXED CASE

Changed to lower case: this is a mixed case

9.2 STRUKTŪROS

Visi iki šiol nagrinėti duomenys, ar tai būtų skaliariniai kintamieji, ar masyvai, gali įgyti tik vieno tipo reikšmę (*int, double, char* ir t.t.). Tačiau realaus gyvenimo objektai, abstrakcijomis atspindimi programose, gali būti gerokai sudėtingesni, ir jiems aprašyti gali prireikti bent kelių skirtingų tipų reikšmių. Pavyzdžiui, informacijai apie knygą talpinti galima sukurti duomenį, turintį tokius 6 komponentus: *char* masyvą autoriaus vardui ir pavardei, tokius pat kitus masyvus knygos pavadinimui, leidyklos pavadinimui, ISBN numeriui, *int* duomenis puslapių skaičiui ir leidimo metams. Aišku, galima visus tuos komponentus saugoti kaip atskirus masyvus ir skaliarus, tačiau tuomet lengvai galima prarasti ryšį, jungiantį juos į vieną visumą.

Toks komponentinis duomuo vadinamas struktūra *struct*; struktūros komponentu gali būti ir kitas *struct* duomuo. Struktūrų duomenys – C kalbos reliktas. C++ kalboje struktūrų vaidmenį atlieka ir dar daugiau galimybių turi klasių duomenys, bet kadangi daugelyje programų ir šiandien galima rasti raktažodžius *struct*, būtina struktūras žinoti. Be to, tai bus geras įvadas į klases.

Struktūrų gyvavimo ciklas – toks pat, kaip ir vidinių C++ duomenų: skelbimas, apibrėžimas, inicializavimas, tik papildomai reiks parodyti duomens komponentus (jie dažniausiai vadinami laukais) ir jų formatus. Pavyzdžiui, struktūra *Knyga* galėtų būti aprašyta taip:

```
struct Knyga {
        char autorius[ 80 ];
        char pavadinimas[ 80 ];
        char leidykla[ 80 ];
        char isbn[ 80 ];
        int puslapiai;
        int metai;
};
```

Atkreipkit dėmesį: paskelbus struktūros formatą, jokia atmintis struktūros laukams dar neskiriama. Atmintis skiriama tik paskelbus ir apibrėžus struktūros formato kintamuosius (juos dažniausiai vadinsime stuktūrų objektais, o ne kintamaisiais – tai atitinka objektinio programavimo terminiją). Juos skelbti būsiančius *Knyga* formato, apibrėžti bei kartu dar ir inicializuoti galima taip:

```
Knyga knyga1 = {
        "Vardas Pavarde",
        "Programavimas",
        "Technika",
        "123.456.789",
        250,
        2012
};
```

Tik dabar bus sukurtas objektas *knyga1*, t.y. jam paskirti 328 baitai atminties ir į šią atminties sritį bus įrašytos parodytos reikšmės. Struktūrų objektų skelbimą bei apibrėžimą ir inicializavimą galima išskirti į du atskirus žingsnius. Tada iš pradžių suformuojama objekto atmintis:

Knyga knyga2;

o į tą atmintį įrašyti laukų norimas reikšmes reikia pakomponenčiui tokiu būdu:

```
strcpy( knyga2.autorius, “Vardas Pavarde“ );
strcpy( knyga2.pavadinimas, “C++“ );
strcpy( knyga2.leidykla, “Technologija“ );
strcpy( knyga2.isbn, “987.654.321“ );
knyga2.puslapiai  = 300;
knyga2.metai  = 2012;
```

Jei toliau programoje reikia kreiptis į atskirus objektų laukus ir su jų reikšmėmis atlikti įvairius veiksmus, kreipinio forma turi būti panaši. Kaip matot, nedidelių keblumų kyla tik su laukais-masyvais, kadangi dabar supaprastinta masyvų inicializavimo sąrašu sintaksė (žr. 4.1 skyrių) nebegalima; eilutės duomenis kopijuoti į laukus *autorius, pavadinimas, leidykla* ir *isbn* tenka bibliotekine funkcija *strcpy*. Su laukais galima atlikti visas operacijas, leidžiamas lauko formato, pavyzdžiui

```
knyga1.puslapiai = knyga2.puslapiai;
knyga1.metai++;
```

ir pan. Galima nukopijuoti visus vienos struktūros laukus į kitos atitinkamus laukus taip:

```
knyga1 = knyga2;
```

tačiau su visu struktūros objektu atlikti kokių nors kitų operacijų, pavyzdžiui, *knyga1++* – nevalia (nebent operacijos operatorius *++* būtų perkrautas; žr. atitinkamą skyrių toliau). Kodėl taip yra – akivaizdu: juk operatorių prasmė apibrėžta tik vidiniams C++ duomenims. Kai duomuo-struktūra turi kelis laukus, ir dar galima skirtingų formatų, kompiliatoriui būtina pranešti, kaip mes norime tuos operatorius taikyti visiems struktūros laukams.

3 pavyzdys. Duomenų faile *duomenys.txt* yra trikampio viršūnių koordinatės tokia tvarka: pirmosios viršūnės *a* koordinatė *x*, koordinatė *y*, toliau antrosios *b* ir trečiosios *c* viršūnių koordinatės ta pat tvarka. Turime suskaičiuoti trikampio plotą. Jei trikampio viršūnės apeinamos prieš laikrodžio rodyklę, dvigubą trikampio plotą galima suskaičiuoti taip:

$$2\Delta = \begin{vmatrix} 1 & x_a & y_a \\ 1 & x_b & y_b \\ 1 & x_c & y_c \end{vmatrix}$$

Uždavinio duomenims talpinti sudarysime dvi struktūras: struktūrą *vertex* vienos viršūnės koordinatėms saugoti, ir sudėtinę *triangle* – trims *vertex* formato viršūnių laukams.

Dabar kreipinys į kurios nors viršūnės kurią nors koordinatę bus su dviem taško operacijomis: *(triangle objektas).(vertex formato laukas – viršūnė).(double formato laukas – koordinatė).* Struktūras aprašysime antraštiniame faile *trikampis.h*.

Antraštinis failas *trikampis.h*:

```
struct vertex {
        double x, y;
};
//
struct triangle {
        vertex a, b, c;
};
```

Pradinis programos failas:

```
#include <iostream>
#include "trikampis.h"
//
using namespace std;
//
int main( ) {
        triangle t;
        cout<<"Iveskite virsuniu koordinates x ir y: ";
        cin>>t.a.x>>t.a.y
           >>t.b.x>>t.b.y
           >>t.c.x>>t.c.y;
        cout<<"Trikampio virsuniu koordinates yra:\n"
                    <<t.a.x<<"  "<<t.a.y<<endl
                    <<t.b.x<<"  "<<t.b.y<<endl
                    <<t.c.x<<"  "<<t.c.y<<endl;
        cout<<"Trikampio plotas yra:\n"
                    <<( t.b.x*t.c.y - t.c.x*t.b.y
                      -t.a.x*t.c.y + t.c.x*t.a.y
                      +t.a.x*t.b.y - t.b.x*t.a.y )/2.
                   <<endl<<endl;
        system("pause");
    return 0;
}
```