5 tema. Objektai ir objektiškai orientuotas programavimas

5.1. Klasės apibrėžimas.

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`** (blobalusis). 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 destruktoriu 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.

🖬 5.1 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.
//-----------------------------------------------------
class Studentas{ char  *pav;
                 float svoris;
        public:
                        // Metodas.
         void Skaito(){
              cout<< "  Iveskite pavarde: \n";
              cin>> pav;
              cout<< "  Iveskite svori:\n";
              cin>> svoris;   };
                        // Metodas.
         void Rodo(){
              cout<< pav<<"  "<< svoris<<endl; };
                   };
//-----------------------------------------------------
void main(){
       Studentas A, B;          // Sukuriami objektai.
       A.Skaito();    B.Skaito();
       B.Rodo();      A.Rodo();
   getch();              // Ekrano lango "užlaikymas".
}
```

5.2 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 objektai. Vidinių metodų informacijai apdoroti klasėje nėra. Klasės objektai skiriami tik duomenų registracijai.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <conio.h>
//-----------------------------------------------------
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;  }
  };
//-----------------------------------------------------
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() << endl;

// Duomenų persiuntimas kitam objektui, t.y. Y = X
Y.Duomenys( X.RodoVarda(), X.RodoSvori() );

Y.Rodo();                       // Objekto Y duomenys
  getch();
  }
//-------------------------------- Metodų aprašai ---
 student:: student( char * a, float b )
          { vardas = a;  svoris = b;         }
void student:: Rodo( ){
            cout<< vardas <<" : "<< svoris << endl; }
//-----------------------------------------------------
```

Ekrane matysime:

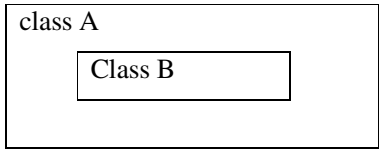
```
Petraitis : 13.5
Petraitis : 13.5
Petraitis : 13.5
```

5.3 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 kievienos klasės) aprašų.

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

```
class sar {  int A[ dydis ];
int ilg;
public:
      void init(  );       // Konstruktorius.
      void add ( int  );  // Sąrašo papildymo metodas.
      void del ( int  );  // Reikšmės šalinimo metodas.
      void get (  );   // Sąrašo rodymo ekrane metodas.
      };
//-----------------------------------------------------
void sar:: init(  ){ ilg =   0; } // Konstruktorius.
                                  // Papildymo metodas.
void sar:: add ( int naujas ){
           // Sąrašas papildomas, jeigu sąraše dar nebuvo.
if ( ilg == dydis ){ cout<<"sarasas pilnas";
                     return;                    }
for ( int i = 0; i < ilg; i++)
  if ( A[i] == naujas ) return;
     A[ ilg++ ] = naujas;                }
                                 // Šalinimo metodas.
void sar:: del ( int elem ) {
  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--;    }}
                           // Išvedimo ekrane metodas.
void sar:: get (  ){
   for ( int i = 0; i < ilg; i++)    cout<< A[i]<<" ";
   cout<<endl;           }
//-----------------------------------------------------
void  main(){
  clrscr();
  sar Rasa;                    // Objekto aprašas.
         Rasa.init();          // 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
    getch();
}
```

5.2. 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*.

5.4 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>
//-----------------------------------------------------
class Langas{ int x1, y1, x2, y2;  // Lango koordinatės
              int fonas;           // ir fono spalva.
      public:
         Langas( int, int, int, int, int );
         void Ekranas();
       };
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();}
//-----------------------------------------------------
 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.
                    };
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); }
//---------------------------------------------------
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();
 }
```

Ekrane rudo fono lange matysime:

```
Grazu:   5     2.50      b      Žalia spalva
Grazu:  15     2.50      b      Žydra spalva
Grazu:  15    27.80      b      Raudona spalva
Grazu:  15    27.80      e      Violetine spalva
```

5.5 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 paveldimama laukui **x** suteikia reikšmę **5**.

```
#include <iostream.h> // Paveldimo metodo panaudojimas
#include <conio.h>    // pagrindineje funkcijoje
//---------------------------------------------------
class Pirmas { int  x;
   public:  Pirmas( int a ){ x = a; }
          void Rodo(){
            cprintf( "Pirmas::rodo() %5d \n\r", x); }
 };//--------------------------------------------------
class Antras: public Pirmas { int  y;
      public:  Antras( int b ): Pirmas( 5 ){ y = b; }
               void Rodo(){
             cprintf( "Antras::rodo() %5d \n\r", y); }
  };//--------------------------------------------------
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();                }
```

Ekrane matysime:

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

5.6 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>
//---------------------------------------------------
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; }
 };//-----------------------------------------------
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 );          }
 };//-----------------------------------------------
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++ ;
}//-----------------------------------------------
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();   }
```

Ekrane skirtingomis spalvomis matysime:

```
K       G
 a       R
  u
   n
```

```
    a
     s
```

5.7 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();` bus 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>
//-----------------------------------------------
 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; }
};//-----------------------------------------------
 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 );          }
};//-----------------------------------------------
 class Zodis: public Raide { char *Zd;
   public:
       Zodis( char *A ): Raide( 1, 1, 1, 'A')
```

```
                {   Zd = A ; }
      void Rodo();
};//-----------------------------------------------
 void  Zodis::Rodo(){             // Metodo aprašymas
   int i =0;
   while( *(Zd+i) != '\0' ){
       Kita( 1, 1, 1, *(Zd+i) );
       Raide::Rodo();  i++;
}}//-----------------------------------------------
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();
     }
```

Raides ekrane matysime skirtingomis spalvomis:	A L a G p a R s

5.3. 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: **. .* :: ?:**

5.8 pratimas. Demostruojamas 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>       // Operatoriu perkrovimas
#include <conio.h>
#include <string.h>
 const N = 40;

class Katinas {
   public:
    Katinas( char * ); // Konstruktorius.
    void operator + ( char *);
    void operator - ( char  );
    void Rodo();
    ~Katinas();       // Destruktorius
   private:
     char *Vardas;
}; //-----------------------------------------------------
  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; }
//-----------------------------------------------------
void main( void ){
   Katinas A ("Batuotas ir piktas");
```

```
   A.Rodo();
   A + " Rudas! ";
   A.Rodo();
   A - 'a';
   A.Rodo();
   getch();
}
```

Ekrane matysime:	Batuotas ir piktas Batuotas ir piktas Rudas! Btuots ir pikts Ruds!

5.4. 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.

5.10 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>       // Metodu perdengimas

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

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::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 << endl;
}//-----------------------------------------------
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();
}
```

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

5.5. Virtualios funkcijos.

Tegul duota protėvio klasė `Lapas` ir iš jos palikuonis `KnygaPirma`. Paskelbsime

`Lapas *p` - rodyklė į klasės Lapas objektą.

`Lapas  L1` - klasės Lapas objektas.

`KnygaPirma  Kn1` -klasės KnygaPirma objektas.

C++ naudojama 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;
```

Dabar rodyklės `p` pagalba galima paimti visus `Kn1` elementus, kuriuos jis paveldėjo iš klasės `Lapas`. Tačiau negalima paimti nuosavų klasės `KnygaPirma` elementų. Jeigu duota rodyklė į palikuonio klasės objektą, tai jos pagalba negalima paimti protėvio klasės elementų.

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 (poliformizmo 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` sukurtoje klasėje Knyga???.

`p->Rodo()` išrinks protėvio klasės funkciją.

Jei `p = &Kn1;`, tai `p->Rodo();` išrinks sukurtos klasės 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 sukurtoms 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 sukurtose 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.

5.10 pratimas. Sukuriama klasė `Lapas`, kutioje 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>
//-----------------------------------------------------
class Lapas {
 public :
   virtual void Rodo( void ){
      cout<<" Klasės  Lapas versija \n";  }
}; //-----------------------------------------------------
class KnygaPirma : public Lapas{
   public :
    virtual void Rodo( void ) {
      cout<<" Klasės  KnygaPirma versija \n";  }
}; //-----------------------------------------------------
class KnygaAntra : public Lapas{
   public :
    virtual void Rodo( void ) {
      cout<<" Klasės  KnygaAntra versija \n";  }
}; //-----------------------------------------------------
void main ( void ){
  Lapas *p,  L1;
  KnygaPirma Kn1;
  KnygaAntra Kn2;
  p = &L1;
  p->Rodo();     // Vykdomas klases Lapas metodas.
  p = &Kn1;
  p->Rodo();     // Vykdomas klases KnygaPirma metodas.
  p = &Kn2;
  p->Rodo();     // Vykdomas klases KnygaAntra metodas.
  getch();
```

```
}
```

Ekrane matysime:	Klasės Lapas versija Klasės KnygaPirma versija Klasės KnygaAntra versija

5.11 pratimas. Modifikuotas 5.10 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>
//-------------------------------------------------
class Lapas {
 public :
   virtual void Rodo( void ){
      cout<<" Klasės  Lapas versija \n";  }
}; //-----------------------------------------------
class KnygaPirma : public Lapas{
   public :
    virtual void Rodo( void ) {
      cout<<" Klasės  KnygaPirma versija \n";  }
}; //-----------------------------------------------
class KnygaAntra : public Lapas{
   public :
      void Matau( void ) {
      cout<<" Klasės  KnygaAntra Matau \n";  }
}; //-----------------------------------------------
class Katinas : public KnygaPirma {
    public:
       void Rodo(){
        cout<<" Klasės Katinas versija \n";  }
}; //-----------------------------------------------
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();
}//-------------------------------------------------
```

Ekrane matysime:	Klasės Lapas versija Klasės KnygaPirma versija Klasės Lapas versija Klasės KnygaPirma versija Klasės Lapas versija Klasės Lapas versija Klasės KnygaAntra Matau Klasės Katinas versija Klasės Katinas versija Klasės Lapas versija Klasės Lapas versija Klasės KnygaPirma versija

5.12 pratimas. Klasėje Lapas aprašomas metodas Rodo abstraktus. Šios plasės palikuonyse yra savos funkcijos, kurios atliekamos vykdymo metu.

```
#include <iostream.h>       // Virtualūs metodai.
#include <conio.h>
//-------------------------------------------------
class Lapas {
 public :
```

```
   virtual void Rodo( void )= 0;
}; //-----------------------------------------------
class KnygaPirma : public Lapas{
   public :
    void Rodo( void ) {
      cout<<" Klasės  KnygaPirma versija \n";  }
}; //-----------------------------------------------
class KnygaAntra : public Lapas{
   public :
      void Rodo( void ) {
      cout<<" Klasės  KnygaAntra versija \n";  }
}; //-----------------------------------------------
class Katinas : public KnygaPirma {
    public:
       void Rodo(){
        cout<<" Klasės Katinas versija \n";  }
}; //-----------------------------------------------
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();
}//-------------------------------------------------
```

Ekrane matysime:	Klasės KnygaPirma versija Klasės KnygaAntra versija Klasės Katinas versija

5.6. 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 skelbima klasė B draugiška,- aprašoma su atributu friend. Klasėje B rašomi metodai, kurie naudojasi klasės A priemonėmis kaip savomis. Draugiška klasė skelbiama public.

5.13 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
//-------------------------------------------------
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;  }
//-------------------------------------------------
class Lentyna{
  public:
   void Keisti(  Knyga *, char *);
   char *Rasti( Knyga );        };
//-------------------------------------------------
// 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 );      }
//-----------------------------------------------------
void  main( void ){
  Knyga K( "Informatika I dalis",
         " J.Adomavicius ir kt.", "KTU");
  Lentyna L;                          K.Rodo();
  L.Keisti( &K, "Technologija");      K.Rodo();
getch();
}
```

Ekrane:	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`.

5.14 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 paildomą darbui. Konstruktorius formuoja apskritimo centro koordinačių, spindulio ir spalvos pradines reikšmes. Metodas `PutCircle` brėžia apskritima grafiniame ekrane. Klasėje skelbiama funkcija `Lygu`.

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

class  Circle;
//-----------------------------------------------------
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; }
};
//-----------------------------------------------------
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 );
                              };
//-----------------------------------------------------
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 );     };
//-----------------------------------------------------
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"; }
//-----------------------------------------------------
void Grafika();
void main(){
  Grafika();// Ekranas paruošiamas darbui grafiniame režime.

  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();
}
void Grafika(){
  int A = DETECT, B ;
 initgraph( &A, &B, "c:\programs\bc5\bin" );
 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 moetodų (funkcijų) prototipus.

5.15 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 Kitoks objektas CC pakeičia objekto C1 visus duomenis. Nubrėžiamas raudonas apskritimas. Pakeičiamas C1 spindulys ir nubrėžiamas raudomas 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;
//-----------------------------------------------------
class Kitoks{
      public:
        void Dydis(  Circle *, int      );
        void Vieta(  Circle *, int, int );
        void Spalva( Circle *, int);    };
//-----------------------------------------------------
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      );
                  };
//-----------------------------------------------------
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; };
//-----------------------------------------------------
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 voletinis apskritimas
  CC.Dydis( &CC, 30 );     // Keičiamas spindulys.
  C2.PutCircle(); // Brėžiamas C2 voletinis apskritimas

  getch();
  closegraph();
}//-------------------------------------------------
void Grafika(){
  int A = DETECT, B ;
 initgraph( &A, &B, "c:\programs\bc5\bin" );
 if( graphresult() != grOk ){
   printf("Klaida: %i %i\n ", A, B);
   exit(1);}
}//-------------------------------------------------
```

5.7. 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>);
```

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

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
//-------------------------------------------------
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"; }
}; //-------------------------------------------------
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 );
  getch();
}
```

Ekrane matysime:	`Klevo  lapas` `Liepa lapas`

🖬 5.16 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 <conio.h>
#include <stdlib.h>
//-------------------------------------------------
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"; }
}; //-------------------------------------------------
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();
                              // Naikinimas.
  for ( i=0; i<3; i++)    free (p[i]);
    cout<<"Pabaiga\n";
  getch();
}
```

Ekrane matysime:	*** Neturiu lapo *** Neturiu lapo *** Neturiu lapo ---------- Įveskite: Liepa Įveskite: Klevas Įveskite: Maumedis Liepa Klevas Maumedis Pabaiga

5.17 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 <conio.h>
#include <stdlib.h>
//-------------------------------------------------
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 ); }
}; //-------------------------------------------------
     struct sar {  Lapas *Medis;
                   sar   *sek;};
//-------------------------------------------------
   sar * Naikinti( sar * );
void main ( void ){
  sar  *P = NULL, *R;  int i;  char T[10];
                              // Sąrašo formavimas.
  for ( i=1; i<=3; i++)            {
    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";
  getch();
}//-------------------------------------------------
sar * Naikinti( sar *P ){
  sar *R;
    while ( P ){  R = P;    P = P->sek;
                  R->Medis->~Lapas();  free( R );  }
  return P;
}//-------------------------------------------------
```

Ekrane matysime:□Įvesk ite: Liepa Įveskite: Klevas Įveskite: Slyva Slyva lapas Klevas lapas Liepa lapas Sarasas

```
tuščias□Pabaiga
```