1 – 2 – 3 PASKAITOS

Turinys: Rodyklės
Rodyklės sąvoka. Nuoroda. Adresai. Prieitis prie kintamojo rodykle. Rodyklės į masyvus. Argumentų perdavimas į funkcijas rodyklėmis.
Dinaminis atminties skyrimas. *new* ir *delete* operacijos. Rodyklės į eilutes.
Rodyklės į objektus. Rodyklių į objektus masyvai. Rodyklės į rodykles. Rodyklės į funkcijas.

Rodyklės sąvoka. Nuoroda. Adresai

Rodyklė (angl. pointer) – C++ duomuo kintamojo adresui saugoti. Rodyklės konstrukcija taikoma:

- prieičiai prie kintamojo
- prieičiai prie masyvo elementų
- argumentų perdavimui į funkcijas, funkcijų kvietimui rodykle
- masyvų ir eilučių perdavimui į funkcijas
- dinaminės atminties skyrimui
- duomenų struktūrų (pavyzdžiui, saitinio sąrašo) kūrimui

Daugeliu atvejų tai galima atlikti ir be rodyklių, bet rodyklėmis – efektyviau. Jos būtinos dinaminės atminties skyrimui, sąrašų kūrimui ir kai kurioms virtualių funkcijų galimybėms realizuoti.
Tai – pavojinga klaidoms konstrukcija, todėl naujesnėje Java kalboje tokios konstrukcijos iš viso atsisakyta.

Visi programos kintamieji kompiuterio atmintyje saugomi skirtingo ilgio ląstelėse. Ląstelių baitai turi unikalius adresus. Kur konkrečiai kintamasis bus saugomas, priklauso nuo sisteminės atminties apimties, pačios programos apimties; nuo to, kelios programos šiuo metu paleistos ir pan. – kiekvienąkart paleidus programą, adresai bus skirtingi.
Kintamojo *k* adresą galima sužinoti adreso operacija *&k* (nemaišyti su *duomens tipas &k* operatoriumi – tai nuorodos (angl. reference) operatorius). Adresai išvedami šešioliktainėje skaičiavimo sistemoje.
Yra kintamųjų, galinčių saugoti šiuos adresus, tipai: *int*, long*, double*, …, void** – rodyklės.
Pavyzdys:

```
#include <iostream>
using namespace std;
//
int main( ){
        int k1 = 10,
            k2 = 20,
            k3 = 30;
        cout<<&k1<<endl<<&k2<<endl<<&k3<<endl; // adresai; skirsis 4 baitais
        int* p;
```

```
        p = &k1;        //rodyklė
        cout<<p<<endl;  // k1 adresas
        p = &k3;
        cout<<p<<endl;  // k3 adresas
        return 0;
}
```

Kompiliatorius turi žinoti, į kokio tipo kintamąjį nutaikytos rodyklės: kad, pavyzdžiui, pereinant nuo vieno masyvo elemento prie kito, būtų aišku, keliais baitais turi skirtis adresas.
Neinicializavus rodyklės (pavyzdžiui, prieskyros operatoriumi), ji rodo atsitiktinį adresą.
Inicializavimo nuline reikšme pavyzdys:

```
int *pint = NULL;
```

Dabar galima tikrinti, ar rodyklė yra kur nors nukreipta:

```
if( pint == NULL ) …
```
arba
```
if (pint != 0 ) …
```

Sintaksė – stilius:

```
char*  pch;     ≡     char   *pch;
char *pch1, *pch2, *pch3;    ≡     char*   pch1, *  pch2, *  pch3;
```

Rodyklė ir nuoroda (angl. reference). Nuoroda skelbiama panašiai kaip rodyklė, po kintamojo formato nurodžius ženklą & (vėliau, vėl panaši konstrukcija kaip rodyklės atveju, prieš kintamojo vardą įrašius šį ženklą tai bus jau kintamojo adresas). Skirtumas nuo rodyklės: nuoroda yra tik kito kintamojo alternatyvus vardas:

```
…
double d = 5.;
double&  rd = d;
rd += 10.;   //kintamojo d reikšmė padidinama 10.
...
```

Prieitis prie kintamojo rodykle

Žinant kintamojo adresą, galima gauti ir jo reikšmę, arba jam priskirti reikšmę, taikant įreikšminimo operaciją **rodyklė.*
Pavyzdys:

```
#include <iostream>
using namespace std;
//
int main( ){
        int k1 = 10, k2 = 20;
```

```
        int* p;
        p = &k1;
        cout<<p<<*p<<endl;   //adresas ir reikšmė
          //
        *p = 30;    // k1 = 30
        k2 = *p;    // k2 = 30
        cout<<*p<<k2<<endl;
        return 0;
}
```

Čia *k1 = 30* - tiesioginė kreiptis į kintamąjį; **p = 30* – netiesioginė kreiptis per rodyklę.

Rodyklės į masyvus

Visi prieigos prie masyvo elementų būdai parodyti šiame pavyzdyje:

```
#include <iostream>
using namespace std;
//
int main( ){
        int array[ ] = { 10, 20, 30, 40, 50 };
        int* p = array;
        for( int i = 0; i < 5; i++ )
                cout<<array[ i ]<<endl;          // 1 pastaba
        for( i = 0; i < 5; i++ )
                cout<<p[ i ]<<endl;              // 2 pastaba
        for( i = 0; i < 5; i++ )
                cout<<*(p+i)<<endl;              // 3 pastaba
        for( i = 0; i < 5; i++ )
                cout<<*p++<<endl;                // 4 pastaba
        for( i = 0; i < 5; i++ )
                cout<<*(array+i)<<endl;     // 5 pastaba
        for( int i = 0; i < 5; i++ )
                cout<<*array++<<endl;     // 6 pastaba
        return 0;
}
```

Pastabos:

1. Tradicinė forma – adresavimo operacija.
2. Adresavimo operacij1 galima taikyti ir rodyklei. Dažniausia taikoma forma.
3. Įreikšminimo operacija. Kiekviename ciklo žingsnyje pereinama prie kito elemento adreso pridedant 4 (tai – ląstelės baitų skaičius *int* formatui)**i*.
4. Rodyklei galima inkremento (čia – postinkremento) operacija.
5. Masyvo vardas kalboje faktiškai yra pirmojo masyvo atminties bloko baito adresas, nors ir nežymimas *&* ženklu prieš vardą. Todėl **(array + i)* yra nulinio elemento adresas + (4* *i*) .
6. Klaida. *array* yra adresas – konstanta, o jai inkremento operacija negalima.

Argumentų-skaliarų perdavimas į funkciją rodyklėmis

Visų trijų perdavimo būdų – reikšme, nuoroda ir rodykle – pavyzdžiai. Antras ir trečias būdai – labai panašūs, yra tik sintaksiniai skirtumai. „Nuoroda“ reiškia, kad funkcijai atiduodamas simbolinis kintamojo vardas – faktiškojo argumento vardas. „Rodykle“ reiškia, kad funkcijai atiduodamas pats kintamojo adresas. Beje, pats adresas perduodamas reikšme. Abiem atvejais supaprastintai galima sakyti, kad formalusis ir faktiškasis argumentai saugomi toje pat atminties vietoje ir faktiškai tėra tik skirtingi tos atminties vietos vardai.
Pavyzdys: temperatūros perskaičiavimas iš Farenheito skalės į Celsijaus skalę.

Reikšme:

```
#include <iostream>
using namespace std;
//
int main( ){
        double convert( double );
        double temp = 32.;
        cout<<"Farenheito laipsniai: "<<temp<<endl;
        cout<<"Celsijaus laipsniai:   "<<convert( temp )<<endl;
        return 0;
}
//
double convert( double t ){
        return ((t - 32.) * 5./9.);
}
```

Nuoroda:

```
#include <iostream>
using namespace std;
//
int main( ){
        void convert( double& );
        double temp = 32.;
        cout<<"Farenheito laipsniai: "<<temp<<endl;
        convert ( temp );
        cout<<"Celsijaus laipsniai:   "<<temp<<endl;
        return 0;
}
//
void convert( double& t ){
        t = (t - 32.) * 5./9.;
}
```

Rodykle:

```
#include <iostream>
using namespace std;
//
int main( ){
        void convert( double* );       // *
        double temp = 32.;
        cout<<"Farenheito laipsniai: "<<temp<<endl;
        convert ( &temp );             // &
        cout<<"Celsijaus laipsniai:   "<<temp<<endl;
        return 0;
}
//
void convert( double* pd ){            // *
        *pd = (*pd - 32.) * 5./9.;    // *pd
}
```

Masyvų perdavimas į funkciją rodykle

Tai – dažniausiai taikomas perdavimo būdas.
Pirmas pavyzdys: ta pati programa temperatūros perskaičiavimui, tik perskaičiuojamos kelios temperatūros. Parodytos visos galimos sintaksės formos (alternatyvios formos programoje užkomentuotos).

```
#include <iostream>
using namespace std;
const int N = 5;
//
int main( ){
        void convert( double* );       // *
        double array[ N ] = { 32., 40., 50., 60., 70. };
        double* pd = array;
        for( int i = 0; i < N; i++ ){
                cout<<"Pradinis masyvas - 1: "<<array[ i ]<<endl;       //1 būdas
              //cout<<"Pradinis masyvas - 2: "<<*(array + i)<<endl;   //2 būdas
              //cout<<"Pradinis masyvas - 3: "<<*(pd+ i)<<endl;       //3 būdas
              //cout<<"Pradinis masyvas - 4: "<<*pd++<<endl;          //4 būdas
              //cout<<"Pradinis masyvas - 5: "<<pd[i]<<endl;          //5 būdas
        }
        //
        convert ( array );             // atiduodamas adresas - array
        //
```

```
        for( i = 0; i < N; i++ )
                cout<<"Pakeistas masyvas - 1: "<<array[ i ]<<endl;        //1 būdas
        return 0;
}
//
void convert( double* p ){
        for( int i = 0; i < N; i++ ){
                   p[ i ] = ( p[ i ] – 32. ) * 5./9.;            // 1 būdas
                //*(p + i) = (*( p + i ) – 32. ) * 5./9.;   // 2 būdas
                //*p++ = (*p - 32.) * 5./9.;                // 3 būdas
}
```

Antras pavyzdys. Masyvo rūšiavimas didėjimo tvarka burbulo metodu.

```
#include <iostream>
using namespace std;
//
int main( ){
        void bubbleSort( int, int* );     // masyvo rūšiavimas
         const int N = 5;
        int array[ N ] = { 10, 9, 1, 5, 8 };
          //
        bubbleSort( N, array );
          //
        for( int i = 0; i < N; i++ )
                cout<<"Rezultato masyvas: "<<array[ i ]<<"  ";
        cout<<endl;
        return 0;
}
//
void bubbleSort( int n, int* p ){
        void sort2n( int*, int* );  // dviejų skaičių rūšiavimas
        for( int  i = 0; i < n-1; i++ )
                for( int j = i+1; j<n; j++ )
                        sort2n( p+i, p+j );  // atiduodami du adresai
}
//
void sort2n( int* n1, int* n2 ){
        if( *n1 > *n2 ){
                int temp = *n1;
                *n1 = *n2;
                *n2 = temp;
        }
}
```

Dinaminis atminties skyrimas. *new* ir *delete* operacijos

Dirbant su masyvais, galima tik statinė atmintis, t.y. atmintis masyvui skiriama kompiliavimo metu ir paskirtas atminties blokas programos vykdymo metu negali būti naudojamas kitiems tikslams.
Taigi, atmintį galima skirti tokiu būdu:

```
int array[ 100 ];
```

Taip negalima:

```
. . .
int size;
cin>>size;
int array[ size ];
. . .
```

Kartais prieš vykdant programą nežinia, kiek atminties reikės skirti masyvui. Galimas sprendimo būdas – paskirti masyvui atminties tiek, kad bet kokiu atveju jos pakaktų – neracionalus.
Atmintį masyvui dinamiškai (t.y. programos vykdymo metu) paskirti galima tik su *new* operacija ir rodyklės konstrukcija. Operacija išskiria reikiamą bevardį atminties bloką masyvui (ar eilutei arba klasės objektui) ir grąžina to bloko pirmosios ląstelės adresą. Šis adresas priskiriamas rodyklei.
Tą pat atmintį programos veikimo metu išlaisvinti galima *delete* operacija. Jei jos nenaudoti – atmintį pasibaigus programai išlaisvins kompiliatorius.
Pavyzdys: dinamiškai paskiriama reikiamo dydžio atmintis masyvui, masyvo elementams priskiriamos tam tikros reikšmės – elementų eilės numerių kvadratai; masyvo elementų reikšmės atspausdinamos ir atmintis išlaisvinama.

```
#include <iostream>
using namespace std;
//
int main( ){
        int* array;  // 1 pastaba
        int size;
                cout<<"Enter size of array:";
                cin>>size;
                        array = new int[ size ];  // 2 pastaba
                                for( int i = 0; i < size; i++ ){
                                        array[ i ] = i*i;          //3 pastaba
                                        //*(array + i) = i*i;   //3 pastaba
                                        //*array++ = i*i;     //4 pastaba
                                                cout<<array[ i ]<<endl;
                                }
                                        delete [ ] array;   // 5 pastaba
        return 0;
}
```

Pastabos:

1. Atminties bloko, kuris bus paskirtas masyvui, pirmojo baito adresas bus priskirtas šiai rodyklei *array*.
2. Tokia yra *new* operacijos sintaksė: *new*, masyvo formatas ir skliaustuose masyvo dydis.
3. Dabar kreiptis į masyvo elementą galima įprasta forma arba kitais alternatyviais jau nagrinėtais būdais.
4. NENAUDOTI. Šiuo konkrečiu atveju šis visiškai korektiškas kreipinys tačiau netiks: inkremento operacija rodyklei ją perstumia į kitus nei masyvo atminties bloko pirmojo baito adresus, o
5. *delete* operacija išlaisvina *size* ląstelių atminties pradedant nuo dabartinės rodyklės *array* adreso. Taigi, jei rodyklė bus nukreipta nuo pirmojo baito adreso į kokį kitą adresą, bus išlaisvinta kažkokiems kitiems tikslams naudota atmintis - įvyks vykdymo laiko klaida. Gi pirmais dviem būdais kreipiantis į masyvą, rodyklės turinys nekeičiamas. Norint čia taikyti inkrementą, reikėtų įsisvesti dar vieną rodyklę, ją prieš ciklą nutaikyti į tą patį pirmojo baito adresą, po masyvą vaikščioti šia nauja rodykle, o atmintį išlaisvinti senąja rodykle.
   Jei *delete* operacija išlaisvintų ne masyvui skirtą atmintį (pavyzdžiui, klasės objektui) – skliaustelių nereikėtų.

Operacijomis *new* ir *delete* dinaminę atmintį galima skirti ir masyvams, kurių matai yra konstantiniai. Daugiamačių masyvų atveju sintaksė tada yra paprastesnė:

```
...
dm = new double[10][5];
tm = new int[n][10][5];   //tik kairysis indeksas gali būti kintamasis
…
delete [ ] dm;
delete [ ] tm;
```

Rodyklės į eilutes

Plačiai taikomi du eilučių realizavimo būdai - masyvu (jau nagrinėtas) ir rodykle.
Pavyzdys:

```
#include <iostream>
using namespace std;
//
int main( ){
        char eil1[ ] = "Realizavimas masyvu";
        char* eil2   = "Realizavimas rodykle";
        cout<<eil1<<endl;
        cout<<eil2<<endl;
        //eil1++;  klaida! – eil1 yra adresas, t.y. konstanta
        eil2+=13;  // viskas tvarkoj
        cout<<eil2<<endl;
        return 0;
}
```

Eilutė – funkcijos argumentas. Tas pats pavyzdys, tik spausdinimas perkeliamas į funkciją:

```
#include <iostream>
using namespace std;
//
int main( ){
        void showString( char* );
        char eil1[ ] = "Realizavimas masyvu";
        char* eil2   = "Realizavimas rodykle";
        showString( eil1 );
        showString( eil2 );
        //eil1++;  klaida! – eil1 yra adresas, t.y. konstanta
        eil2+=13;  // viskas tvarkoj
        showString( eil2 );
        cout<<eil2<<endl;
        return 0;
}
//
void showString( char* ps ){
        while( *ps )                // kol nesurastas pabaigos simbolis
                cout<<*ps++;
        cout<<endl;
}
```

Gali kilti klausimas: kodėl programa veikia teisingai, jei funkcijoje rodyklė inkremento operacija nukreipiama į kitą nei pirmojo eilutės atminties bloko baito adresą? – nes pati

rodyklės reikšmė – adresas – į funkciją atiduodama perdavimo reikšme mechanizmu, ir jo pokyčiai funkcijoje nėra matomi programoje. Tą iliustruoja paskutinysis spausdinimas programoje.

Pavyzdys su dvimačiu eilučių masyvu – savaitės dienų pavadinimais. Vėl parodomi abu realizavimo būdai – dvimačiu eilučių masyvu ir vienmačiu rodyklių masyvu į atskiras eilutes.

```
#include <iostream>
using namespace std;
//
int main( ){
        const int D = 7,
                  MAX = 20;
        char sd1[ D ][ MAX ] = { "Pirmadienis", "Antradienis", "Treciadienis",
                                 "Ketvirtadienis", "Penktadienis", "Sestadienis",
                                 "Sekmadienis" };
        for( int i = 0; i<D; i++ )
                cout<<sd1[ i ]<<endl;
        //
        char* sd2[ D ] = { "Pirmadienis", "Antradienis", "Treciadienis",
                           "Ketvirtadienis", "Penktadienis", "Sestadienis",
                           "Sekmadienis" };
        for(  i = 0; i<D; i++ )
                cout<<sd2[ i ]<<endl;
        return 0;
}
```

Pirmuoju atveju eilutės atmintyje būtų saugomos kaip dvimatis (*D* x *MAX*) baitų masyvas, t.y.

Pirmadienis\0 <8 intervalai> Antradienis\0 <8 intervalai> Treciadienis\0 <7 intervalai> . . .

Antruoju atveju tai būtų vienmatis rodyklių masyvas į atskirų eilučių pirmuosius baitus, o tos eilutės atmintyje eitų be jokių tarpų:

Pirmadienis\0Antradienis\0Treciadienis\0 . . .

Tai – daug racionalesnis realizavimo variantas.

Rodyklės į objektus

Pirmajame semestre jau nagrinėtos angliškų atstumų klasės *Distance* pagrindu parodysim tradicinį būdą objektams kurti (čia – konstruktoriumi pagal numatymą) ir naują, *new* operacija:

```
#include <iostream>
using namespace std;
//
class Distance {
        private:
                int feet;
                float inches;
        public:
                void getDistance( ){
                        cout<<"Iveskite feet ir inches: ";
                        cin>>feet>>inches;
                }
                void showDistance( ){
                        cout<<feet<<"  "<<inches<<endl;
                }
};
//
int main( ){
        Distance d;          // tradicinis būdas: kviečiamas standartinis konstruktorius, ir
        d.getDistance( );
        d.showDistance( );
          //
        Distance* dp;     //  . . . kitaip: rodykle
        dp = new Distance;   // skiriama atmintis
        dp -> getDistance( );  // metodo kvietimas: rekomenduojama sintaksė
        dp -> showDistance( );
                //(*dp).getDistance( );  // metodo kvietimas: alternatyvi sintaksė
                //(*dp).showDistance( );
        return 0;
}
```

Rodyklių į objektus masyvai

Tai – lankstesnė konstrukcija už objektų masyvus (tas bus matyt iš pavyzdžio kitame skyriuje). Tegu turim klasę apie asmenį *Person* vien tik iš vieno lauko – asmens pavardės iki 20 simbolių ilgio, bei dviejų metodų jos įvedimui ir parodymui. Klasės objektai kuriami interaktyviai, iki vartotojas nuspręs nebeformuoti kito objekto atsakymu "*n*". Iš

viso gali būti suformuota iki 100 objektų. Objektai atmintyje išdėstomi bet kur, o į juos nutaikomos rodyklės talpinamos vienmačiame rodyklių masyve.

```
#include <iostream>
using namespace std;
//
class Person {
        private:
                char name[ 20 ];
        public:
                void setName( ){
                        cout<<"Iveskite name: ";
                        cin>>name;
                }
                void showName( ){
                        cout<<name<<endl;
                }
};
//
int main( ){
        Person* personp[ 100 ];      // rodyklių masyvas
        int n = 0;                   // objektų kiekis
        char s;                      // atsakymo simbolis
        do {
                personp[ n ] = new Person;  // skiriama atmintis 1 objektui
                personp[ n ]-> setName( );  // kviečiamas įvedimo metodas
                        n++;
                        cout<<"Testi ivedima? t/n";
                        cin>>s;
        } while (s == 't' );
           /*
              objektų spausdinimas
            */
        for( int i = 0; i<n; i++ ){
                cout<<"Asmuo Nr. "<<i+1;
                personp[ i ]  -> showName( );
                // (*personp++) -> showName( );  alternatyvi sintaksė
        }
}
```

Rodyklės į rodykles

Pratęsim tą pat pavyzdį: taip pat suformuojami keli objektai, įvedamos jų laukų reikšmės, o po to objektai išrūšiuojami pagal abėcėlę pagal jų pavardes. Rūšiuojami bus ne patys objektai (t.y. jų vieta kompiuterio atmintyje liks ta pat), o tik rodyklių į juos masyvas. Taip – žymiai efektyviau.
Algoritmas: jau nagrinėtasis burbulo algoritmas tokioje formoje kaip 5-ame psl., tik čia tarpusavyje lyginami ne du skaičiai, o du C++ vidinės klasės *string* objektai. Kadangi šiai klasei yra perkrautos santykio operacijos (joms z> …>c>b>a), pavardės laukas realizuotas būtent kaip šios klasės objektas. Darbui su klase būtinas antraštinis failas *string*.

```
#include <iostream>
#include <string>
using namespace std;
//
class Person{
        private:
                string name;
        public:
                void setName( ){
                        cout<<"Iveskite name: ";
                        cin>>name;
                }
                string getName( ){
                        return name;
                }
                void showName( ){
                        cout<<"Name = "<<name<<endl;
                }
};
//
int main( ){
        //
        void pointerSort( int, Person** );   // 1 pastaba
        //
        Person* personp[ 100 ];
        int n = 0;
        char s;
        do{
                personp[ n ] = new Person;
                personp[ n ] -> setName( );
                        n++;
                        cout<<"Testi ivedima? t/n ";
                        cin>>s;
        } while( s == 't' );
```

```
        cout<<"Pradinis masyvas "<<endl;
        for( int i=0; i<n; i++ )
                personp[ i ] -> showName( );
        //
        pointerSort( n, personp );     // 2
        //
        cout<<"Isrusiuotas masyvas "<<endl;
        for( i=0; i<n; i++ )
                personp[ i ] -> showName( );
        //
        return 0;
}
//
void pointerSort( int n, Person** p ){
        //
        void sort2p( Person**, Person** );   // 3
        //
        for( int i=0; i<n-1; i++ )
                for( int j=i+1; j<n; j++ )
                        sort2p( p+i, p+j );   // 4
}
//
void sort2p( Person** p1, Person** p2){
        if( (*p1) -> getName() > (*p2) -> getName() ){  // 5
                Person* temp = *p1;
                *p1 = *p2;
                *p2 = temp;
        }
}
```

Pastabos:

1. Jei *personp[k]* elementas yra *Person** tipo, t.y. rodyklė į *k*-ąjį objektą kompiuterio atmintyje, tai *Person*** tipo kintamasis galėtų būti rodyklė į vienmatį rodyklių masyvą ir rodytų į to masyvo atminties bloko pirmąjį baitą.
2. *personp*, teikiamas kviečiant funkciją *pointerSort*, yra tokios rodyklės į rodyklių masyvą reikšmė – rodyklių masyvo pirmojo baito adresas.
3. Rodyklių masyve jo elementai atrenkami rodyklėmis, todėl tų elementų tipas turi būti *Person***.
4. Faktiškieji argumentai *p+i* ir *p+j* leidžia atrinkti rodyklių masyvo elementus pradedant nuo jo pirmojo baito adreso *p*.
5. Būtinos dvi įreikšminimo operacijos lauko reikšmei gauti: *p1* yra rodyklės į rodyklę tipo, todėl operacija **p1* teiks rodyklių masyvo elemento reikšmę – rodyklę; o toliau jau *(…)* - > operacija – teiks atminties bloko, į kurį ši nutaikyta, reikšmę. Tam atminties blokui – klasės objektui kviečiamas metodas *getName* ir grąžina lauko reikšmę – *string* klasės objektą – pavardę. *string* klasės objektams yra perkrauta > operacija.

Rodyklės į funkcijas

Galima deklaruoti rodyklę į funkciją ir ją kviesti per rodyklę; rodyklė šiuo atveju rodo į funkcijai skirtos atminties bloko pirmąjį baitą. Deklaravimo metu nurodomas funkcijos grąžinamos reikšmės formatas, skliausteliuose – pati rodyklė, ir funkcijos argumentų sąrašas (kaip ir funkcijos prototipe). Tokią rodyklę galima nukreipti tik į tas funkcijas, kurių grąžinama reikšmė ir prototipas tiksliai sutampa.
Pavyzdys:

```
#include <iostream>
using namespace std;
//
int sum( int, int );       //abiejų funkcijų prototipai
int product( int, int );  //tiksliai sutampa
//
int main( ){
        int (*pfunction ) ( int, int );     //rodyklė į funkciją; prototipas sutampa su
                                            //funkcijų sum ir product prototipais
        pfunction = product;   //rodyklę privaloma inicializuoti
        cout<<"10*11= "<<pfunction( 10, 11 )<<endl;
        pfunction = sum;
        cout<<"3*(4+5)+6= "<<
                pfunction( product( 3, pfunction( 4, 5 )), 6 )<<endl;
        return 0;
}
```

Rodyklės į funkcijas savo ruožtu gali būti argumentai, masyvo elementai ir pan. Pavyzdys su rodyklėmis į funkcijas-argumentais:

```
#include <iostream>
using namespace std;
//
double square( double );       //abiejų funkcijų prototipai
double cube( double );         //tiksliai sutampa
double sum( double array[ ], int n, double (*pfunction) (double) );     //sąraše
                                        //teikiamas rodyklės į funkciją prototipas
//
int main( ){
        double array[100];
        int n;
        …
        <n ir array įvedimas>
        …
        cout<<"Sum of squares  "<<sum( array, n, square )<<endl;
        cout<<"Sum of cubes  "<<sum( array, n, cube )<<endl;
        return 0;
```

```
}
//
double square( double x ){
        return x*x;
}
//
double cube( double x ){
        return x*x*x;
}
//
double sum( double x[ ], int n, double (*pf) (double) ){
        double s = 0.;
        for( int i=0; i<n; i++ )  s+= pf( x[i] );
        return s;
}
```

Kai kurie C++/CLI dinaminės atminties aspektai

C++/CLI kompiliatorius automatiškai atima atmintį iš nebenaudojamų atminties blokų – nebereikalinga *delete* operacija; tai sumažina „atminties nutekėjimo“ (angl. memory leaks) klaidų. Be to *heap* atmintis automatiškai kompaktizuojama. Todėl vietoj *new* operacijos gali būti taikoma operacija *gcnew* (pirmosios raidės – nuo garbage collector – atminties rinktuvo).
Bet automatiškai kompaktizuojant atmintį, anksčiau nustatytos rodyklės rodys jau į tas atminties ląsteles, kuriose bus visai kitas turinys nei anksčiau. Reikia, kad rodyklės sektų kompaktizavimo metu iš vienos atminties vietos į kitas persiunčiamų duomenų adresus – todėl šiame kalbos standarte vietoje rodyklių naudojamos jų atitikmenys – „handle“ – rankenėlės. Pavyzdys: dinaminės atminties sveikaskaitiniam dvimačiam masyvui iš 4 eilučių ir 10 stulpelių skyrimas:

```
array< int, 2 >^ = gcnew array< int, 2 >(4, 10)
```

Beje, šiame kalbos standarte masyvo elementus galima išrinkti Fortrano stiliumi:
array[i, j].