# 9. Funkcijos (function) tipas

Šiame skyriuje supažindinama su C++ programų struktūravimo priemone funkcijos tipu. Skaitytojas susipažįsta su funkcijos aprašo ir kreipinio sintakse bei naudojimosi tvarka. Skyriuje aptariama funkcijos prototipas ir naudojimosi taisyklės. Skaitytoja susipažįsta su kintamųjų galiojimo srities samprata, galiojimo srit į rodančiais kintamųjų tipais: lokaliaisiais, globaliaisiais, automatiniais ir statiniais . Be to nagrinėjama šių kintamųjų aprašo sintaksė ir kintamųjų naudojimosi tvarka. Skaitytojas išmoksta dviem būdais siųsti duomenis funkcijoms - siunčiant reikšmę ir siunčiant adresą.

➢ **Mokymosi tikslai**

Mokydamasis ir baigęs šį skyrių skaitytojas turi:

- žinoti funkcijos aprašo ir kreipinio sintaksę;
- mokėti naudotis funkcijos prototipu;
- žinoti kintamųjų galiojimo sritis rodančius kintamųjų tipus;
- žinoti naudojimosi globaliaisiais, lokaliaisiais, automatiniais ir statiniais kintamaisiais taisykles;
- mokėti funkcijoms siųsti kintamųjų reikšmes ir adresus.

## 9.1 Funkcijos aprašas

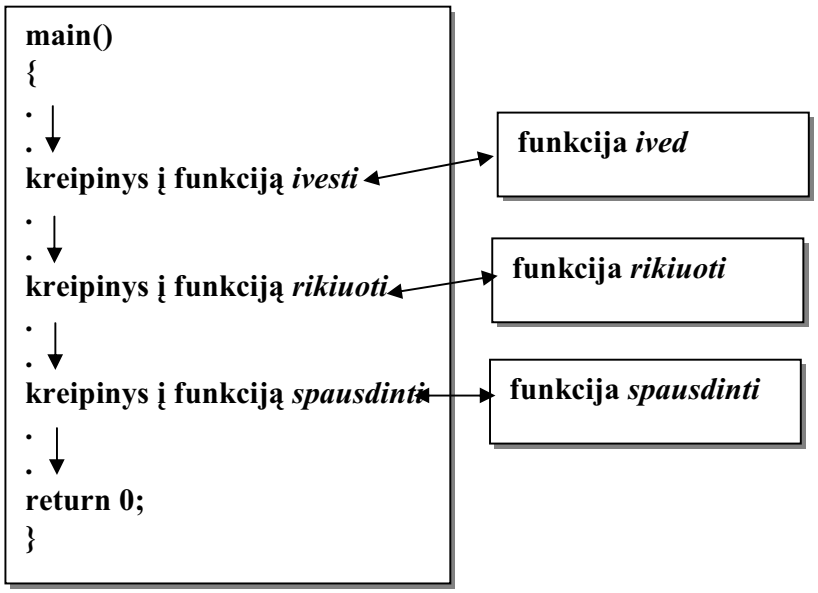
Labai dažnai programose be pagrindinės funkcijos *main()* naudojamos programuotojų sukurtos, originalios funkcijos, kurioms iš pagrindinės funkcijos *main()* gali būti siunčiamos įvairios duomenų reikšmės, o suskaičiuotas rezultatas grąžinamas kviečiančiajai funkcijai. Projektuojant programą, uždavinys suskaidomas į tam tikros paskirties dalis, kurios vadinamos funkcijomis.

> *Funkcija – tai programos dalis, kurią galima atlikti daugiau negu vieną kartą.*

Reikalavimai funkcijoms:

1. Visos programos turi vieną pagrindinę funkciją. Funkcijos vardas prasideda raide arba pabraukimo simboliu ir baigiamas lenktinių skliaustelių pora, tarp kurių rašomi funkcijos parametrai.
2. Funkcijos tekstas aprėminamas figūriniais skliausteliais.
3. Funkcijos aprašomos nuosekliai viena po kitos, įskaitant ir pagrindinę funkciją.
4. Funkcijų aprašai gali būti rašomi prieš arba už pagrindinės funkcijos.
5. Jeigu naudotojo funkcija rašoma už pagrindinės funkcijos, tai programos pradžioje būtina parašyti funkcijos prototipą.
6. Kiekvienos funkcijos pabaigoje rašomas sakinys *return*, nurodantis funkcijos grąžinama reikšmę.

Struktūruotos programos valdymo ryšiai tarp pagrindinės funkcijos *main()* ir vidinių funkcijų rodomi 9.1 paveiksle.




*9.1 pav. Valdymo ryšiai struktūruotoje programoje.*

*Funkcijos aprašo sintaksė:*

**funkcijos reikšmės tipas  funkcijos vardas ([ parametrų sąrašas ])**

**{**

**lokaliųjų kintamųjų aprašai**

**tekstas**

**return [(funkcijos reikšmė)];**

**}**

*Funkcijos reikšmės tipas* - nurodomas baziniais duomenų tipais ir jų modifikacijomis. Jeigu funkcijos reikšmė neskaičiuojama, vietoje tipo rašomas žodis *void* (tuščias).

*Funkcijos vardas* prasideda raide arba pabraukimo simboliu.

*Parametrų sąrašas* - aprašant parametrą, nurodomas parametro reikšmės tipas ir vardas. Parametrų aprašai sąraše vienas nuo kito skiriami kableliais.

*Parametrų sąrašo sintaksė:*

***(tipas$_1$ vardas$_1$, . . . , tipas$_n$ vardas $_n$)***

*Funkcijos reikšmė* - rašoma sakinyje *return* lenktiniuose skliausteliuose. Reikšmė gali būti nurodoma kintamojo vardu, konstanta arba rodykle. Jeigu funkcijos reikšmė neskaičiuojama, rašomas operatorius *return*; Funkcija gali grąžinti tik vieną reikšmę.

Įvairiose programos vietose, parašius funkcijos kreipinį, galima atlikti funkcijoje numatytus veiksmus. Jeigu skaičiuojama funkcijos reikšmė, tai rezultatas įstatomas išraiškoje, kreipinio vietoje.

*Kreipinio sintaksė:*

***funkcijos vardas ([parametrų sąrašas])***

**1 Pavyzdys**

```
// funkcijos f1 realizacijos aprašas
float f1(int i, float j)
{
        float s;
        s = i / sqrt(j);
        return (s);
}
```

**1 Pratimas**

```
// Programa demonstruoja funkcijų iškvietimą
# include <iostream.h>
        void f2(void)
        {
        cout << "Dirba antra funkcija \n";
        return;
        }

        void f3(void)
        {
        cout << "Dirba trečia funkcija \n";
        return;
        }
main()
{
        cout << "Dirba pagrindinė f-ja \n";
        f2( );
        f3( );
        cout << "Pagrindinės funkcijos pabaiga \n";
        return 0;
}
// programos rezultatas - ekrane rodomos keturios eilutės:
```

**Dirba pagrindinė f-ja**
**Dirba antra funkcija**
**Dirba trečia funkcija**
**Pagrindinės funkcijos pabaiga**

## Funkcijų prototipai

Funkcijos prototipas sutampa su funkcijos aprašo pirmąja eilute, tik baigiamas kabliataškiu. Programoje naudojamų funkcijų prototipai rašomi po instrukcijų pirminiui procesoriui.

Pateikiame keletą patarimų:

1. Prototipai turi atitikti pirmą funkcijos pavadinimo eilutę. Prototipo skliausteliuose nereikia vardinti visų parametrų, užtenka nurodyti jų tipą.
2. Būtina aprašyti visų funkcijų, kuriomis naudojamasi programoje, prototipus. Tačiau funkcijų, kurios yra kviečiamos iš bibliotekų, prototipų aprašyti nereikia, nes jų prototipai yra pavadinimų bylose.
3. Prototipų aprašuose nurodomi funkcijų parametrų tipai ir, jeigu reikia, grąžinamųjų reikšmių tipai.
4. *void,* parašytas vietoj funkcijos reikšmės tipo, rodo, kad funkcija negrąžina jokios reikšmės. void, parašytas vietoj parametrų rodo, kad parametrai funkcijai nėra perduodami.

**2 Pavyzdys**

```
// Funkcija skaičiuoja trijų kintamųjų reikšmių vidurkį
#include <iostream.h>
int sk(int n1, int n2, int n3)
{
        int vid;
        vid=(n1+n2+n3)/3;
        return (vid);                   // reikšmės grąžinimas
}
//*****************************************************
main()
{
        int n1, n2, n3;
        int vidurkis;
        cout << "Įveskite tris skaičius";
        cin >> n1 >> n2 >>n3;
        vidurkis = sk (n1, n2, n3);     // kreipinys į funkciją
        cout << "Vidurkis yra " << vidurkis;
        return 0;
}
```

**2 Pratimas**

```
#include <iostream.h>
#include <iomanip.h>
float keisti (float c);    // Funkcijos prototipai
main()
{
        float c;
        float f;
        cout << "Kokia temperatūra pagal Celsijų?";
        cin >> c;
        f =keisti(c);
        cout << "Ekvivalentas pagal Farenheitą  yra " << setprecision(2) << f <<'\n';
return 0;
}
float keisti (float c)
{
        float f;
        f =c*(9.0/5.0)+32.0;
return (f);
}
```

## 9.2 Kintamųjų galiojimo sritys

Lokalieji ir globalieji kintamieji

⚐ *Globalieji kintamieji* galioja visoje programoje ir jų reikšmės gali būti keičiamos įvairiose programos funkcijose. *Globalieji* kintamieji galioja nuo jų apibrėžimo vietos iki programos pabaigos.

⚐ *Lokalieji kintamieji* galioja ir jų reikšmės gali būti keičiamos tik toje funkcijoje, kurioje jie aprašyti. *Lokaliųjų* kintamųjų reikšmių negalima pakeisti kitose funkcijose. *Lokalieji* kintamieji ištrinami iš atminties funkcijai baigus darbą.

Kintamųjų galiojimo sritys rodo, kuriose funkcijose galioja kintamieji ir kokiuose skaičiavimuose galima naudotis jų reikšmėmis.

**3 Pavyzdys**

**1. lokalieji kintamieji i,j**

```
main()
{
int      i, j;
}
```

**2. globalieji kintamieji g,h**

```
int g, h;
main()
{
}
```

**3 Pratimas**

```
// Kintamųjų galiojimo sritys
#include <iostream.h>
#include <iomanip.h>
float z=9.0;                              // globalusis kintamasis
void f1 (void)
     {
     int j=5;                             // lokalusis funkcijos f1 kintamasis
     cout <<j << ' ' << z << endl;
     return ;
     }
int i=0;                                  // globalusis kintamasis

main()
{
     float p;                             // yra lokalusis main() funkcijos kintamasis
     p=9.0;
     cout << i << " * " << p << " * " << z << endl;
     f1 ();                          // parodo globaliuosius kintamuosius ir lokalųjį p
     return 0;
}
```

Programoje naudojami lokalieji ir globalieji kintamieji, pasižymintys šiomis savybėmis:

1. *j* lokalusis *f1* funkcijos kintamasis. Funkcijoje *main()* juo naudotis negalima. Funkcijai *f1* baigus darbą, *j* kintamasis ištrinamas iš atminties.
2. *z* ir *i* globalieji kintamieji prieinami visose funkcijose nuo jų aprašymo vietos. Funkcijoje *f1* negalima naudotis *i* kintamuoju.
3. *p* - tai funkcijos *main()* lokalusis kintamasis. Todėl funkcijoje *f1* juo naudotis negalima.
4. Jeigu lokalieji kintamieji aprašomi skirtingose funkcijose, tai jie gali turėti tokį patį vardą. Tai skirtingi kintamieji, tačiau šiuo atveju reikia būti ypač atsargiems.

Aprašius kintamuosius kaip lokaliuosius, apsaugomos jų reikšmės, nes kintamuoju galima naudotis tik toje funkcijoje, kurioje kintamasis aprašytas.

Ką daryti, jei lokaliuoju kintamuoju reikia naudotis kitoje funkcijoje? Galimi du atvejai:

1. Aprašyti kintamąjį kaip globalųjį.
2. Persiųsti lokalųjį kintamąjį iš vienos funkcijos į kitą.

Siunčiant lokalųjį kintamąjį iš vienos funkcijos į kitą:

- funkcijai siunčiamos *argumentai*;
- funkcijoje naudojamasi *parametrais*.

**4 Pratimas**

```
// Programa skaičiuoja kiek kartų funkcijos reikšmė buvo teigiama
#include <iostream.h>
#include <math.h>
float f1(int x);
main()
{
int kiek = 0;
float x,y;
for ( x = 0; x <= 1; x+=0.2 )
{
  y = f1 (x);
  cout << "y=" << y << endl;
  if ( y > 0) kiek++;
}
cout << "Funkcijos reiksme y buvo teigiama " << kiek << "kartus"<<endl;
return 0;
}
//-------------- f1 --------------------
float f1(int x)
{
        float y;
        y = 2*sin(x)-cos(x)-1;
return (y);
}
```

Pavyzdyje programa skaičiuoja kiek kartų funkcijos reikšmė teigiama. Pagrindinėje funkcijoje generuojamos kintamojo x reikšmės atkarpoje nuo 0 iki 5. Kiekviena reikšmė perduodama funkcijai f1, kurioje skaičiuojama y reikšmė grąžinama pagrindinei funkcijai. Programoje naudojami kintamieji pasižymintys šiomis savybėmis:

1. *x* parametras tai lokalusis *f1* funkcijos kintamasis.
2. Parametro tipas rašomas lenktiniuose skliausteliuose prieš parametro vardą, ir atskiriamas tarpu.
3. *y* tai lokalusis *f1* funkcijos kintamasis.
4. *x, y, kiek* tai lokalieji *main()* funkcijos kintamieji.

> *Lokaliųjų kintamųjų vardai ir parametrų vardai vienoje funkcijoje turi būti unikalūs.*
>
> *Lokaliųjų kintamųjų vardai ir parametrų vardai skirtingose funkcijose gali sutapti.*

### Automatiniai ir statiniai kintamieji

Terminai "automatinis" ir "statinis" paaiškina, kas atsitinka su lokalaus kintamojo reikšme, kai valdymas perduodamas kviečiančiajai funkcijai. Visi lokalieji kintamieji aprašomi kaip automatiniai. Tai reiškia, kad funkcijai baigus darbą, jie ištrinami iš operatyviosios kompiuterio atminties.

*Automatinio* kintamojo aprašo sintaksė:

**auto tipas kintamojo vardas;**

Priešdėlis a*uto* nebūtinas.

**4 Pavyzdys**

```
auto int x;
     int y;
```

Visi globalieji kintamieji aprašomi kaip *statiniai*. Funkcijai baigus darbą, statiniai lokalieji kintamieji iš atminties neištrinami.

*Statinio kintamojo aprašo sintaksė:*

***static tipas kintamojo vardas;***

**5 Pavyzdys**

```
static int x;
```

Aprašant statinį kintamąjį, būtina priskirti pradinę reikšmę. Pradinė reikšmė priskiriama tik pirmą kartą, kviečiant funkciją. Statinis kintamasis lieka atmintyje funkcijai baigus darbą.

Jeigu aprašant kintamąjį pradinė reikšmė nenustatoma, tai pirmą kartą kviečiant funkciją, kintamojo reikšmė prilyginama nuliui.

> *Statiniais kintamaisiais patogu naudotis sumas skaičiuojančiose funkcijose.*

## 9.3 Duomenų siuntimo funkcijoms būdai

C++ programose galima naudotis dviem duomenų siuntimo būdais:

1. siunčiama kintamojo reikšmė;
2. siunčiamas kintamojo adresas.

### Kintamojo reikšmės siuntimas

Siunčiant argumento reikšmę, ji priskiriama kviečiamos funkcijos parametrui. Jeigu siunčiama daugiau nei viena argumento reikšmė, tai jų reikšmės priskiriamos atitinkamiems parametrams.

**6 Pavyzdys**

```
pr ( raide, i )                 // kreipinyje raide ir i - tai pr funkcijos argumentai
void pr ( char raide, int i )   // pr funkcijos apraše raide ir i parametrai
```

Siunčiant reikšmę, patogu naudotis tais pačiais argumentų ir parametrų vardais.

Sekančiame pavyzdyje funkcijai *sp* siunčiama kintamojo *i* reikšmė. Jeigu funkcijoje *pr* pakeisime *i* reikšmę, tai *i* reikšmė funkcijoje *main()* nepasikeis. Automatiškai siunčiamos lokaliųjų kintamųjų reikšmės. Tačiau galima naudotis specialiaisiais operatoriais, leidžiančiais siųsti kintamųjų adresus.

**7 Pavyzdys**

```
#include <iostream.h>
main()
{
   int  i=5;
   sp(i);                  // kreipinys į funkciją sp
   return 0;
}
void sp( int i)
{
   i=i+1;
   cout<<i;
   return;                 // reikšmė negrąžinama
}
```

## Kintamojo adreso siuntimas

Visų kintamųjų reikšmės operatyviojoje atmintyje saugomos ląstelėse, kurias kreipiamasi adresu. Aprašant kintamąjį, jam automatiškai operatyviojoje atmint yje rezervuojama vieta ir jos adresas priskiriamas kintamajam.

Siunčiant argumento adresą, kviečiamoje funkcijoje parametrui priskiriamas argumento adresas. Šiuo atveju parametras, tai rodyklės tipo kintamasis. Parametro reikšmė surandama pagal argmento adresą. Kai siunčiama daugiau argumentų, kiekvieno iš jų adresas priskiriamas kviečiamos funkcijos atitinkamiems parametrams. Jeigu kviečiamoje funkcijoje keičiama parametro, kurio adresas buvo perduotas, reikšmė, tai pasikeičia ir argumento reikšmė kviečiančioje funkcijoje, nes ištikrųjų tai ta pati sritis kompiuterio operatyviojoje atmintyje. .

Kol kas nagrinėjome paprastuosius kintamuosius, skiriamus įvairių tipų ir jų modifikacijų reikšmėms saugoti. Rodyklės – tai išvestiniai duomenų tipai, rodantys vietą atmintyje, kur saugomos kintamųjų reikšmės.

> *Rodyklės - kintamieji, kurių reikšmės kitų kintamųjų atminties adresai.*

Operatoriai skiriami darbui su rodyklėmis:

& - reikšmės adresas;

* - reikšmė, į kurią nukreipia adresas.

1. Kviečiančiojoje funkcijoje, prieš argumento vardą rašomas &.
2. Kviečiamojoje funkcijoje parametras - tai rodyklė. Rodyklės vardas pradedamas *.

> Siunčiant masyvą visuomet funkcijai perduodamas masyvo pirmojo elemento adresas.

**5 Pratimas**

```
// kintamojo adreso siuntimas
#include <iostream.h>
void f1 (int *y)                    // int *y          parametras - sveikojo tipo rodyklė
{
        *y=85;
        cout << "Vykdoma  f1 y=" << *y <<endl;
        return;
}
main()
{
int y;
y =100;
        cout << "Pagrindinėje y=" << y << endl;
        f1 (&y);                                        // &y    argumento y adresas
        cout << "Sugrąžintas y =" << y;
return 0;
}
```

**6 Pratimas**

```
// Masyvų adreso siuntimas
#include <iostream.h>
#include <string.h>
void keisti(char c[5])
{
        cout << c << endl;
        strcpy (c,"ŽIEMA");
        return;
}
main()
{
        char mas1[5]="RUDUO";
        keisti(mas1);
        cout << mas1<<endl;
        return 0;
}
```

Programos darbo rezultatas:

```
RUDUO
ŽIEMA
```

*keisti* funkcijoje *c* masyvas tai tas pats *main()* funkcijos *m1* masyvas, nes *keisti* funkcijai perduodamas *m1* masyvo adresas. Abi *main()* ir *keisti* funkcijos dirba su ta pačia atminties sritimi, nors šioje programoje parametro ir argumento vardai skirtingi . Masyvo elementų reikšmes galima keisti vienoje funkcijoje, o pasinaudoti kitoje funkcijoje.

## Klausimai

1. Kokių tipų kintamųjų automatiškai perduodamas adresas?
2. Kokių tipų kintamųjų automatiškai perduodama reikšmė?
3. Jeigu perduodama argumento reikšmė ir vėliau funkcijoje parametro reikšmė keičiama, ar pasikeis argumento reikšmė kviečiančiojoje funkcijoje?
4. Jeigu funkcijai perduodamas argumento adresas ir joje keičiama parametro reikšmė, ar pasikeis argumento reikšmė kviečiančiojoje funkcijoje?
5. Kur nurodomas funkcijos reikšmės tipas?
6. Koks maksimalus grąžinamųjų reikšmių skaičius?
7. Kelių argumentų reikšmes galima siųsti funkcijai?
8. Kodėl iš kviečiančiosios funkcijos nereikia sugrąžinti globaliojo kintamojo reikšmės?
9. Paaiškinkite šį funkcijos prototipą.
   **float mano(char a, int b, float c);**

## Užduotys

1. Parašykite funkciją, kuri skaičiuotų, kiek kartų ji buvo iškviesta. Reikšmių į šią funkciją perduoti nereikia. Rezultatą grąžinti kviečiančiajai funkcijai ir rodyti ekrane.

2. Parašykite funkciją, kuriai perduodami dviejų sveikojo tipo kintamųjų adresai. Funkcijoje aprašykite trečiąjį kintamąjį. Naudodamiesi juo, sukeiskite pirmųjų dviejų reikšmes vietomis.

3. Parašykite programą, kurioje tris kartus klaviatūra įvedama apskritimo spindulio reikšmė. Spindulio reikšmė perduodama funkcijai, skaičiuojančiai apskritimo plotą pagal formulę pl=3,14*$r^2$.Suskaičiuota reikšmė grąžinama kviečiančiajai funkcijai ir rodoma ekrane.

4. Parašykite funkciją, kuriai perduodamas *x* argumento adresas. x kinta ribose nuo 1 iki 57. Funkcija skaičiuoja *y* pagal formulę $y=9x^4+15x^2+x$. Suskaičiuota reikšmė grąžinama kviečiančiajai funkcijai ir rodoma ekrane.

## Santrauka

## Informacijos šaltiniai