# C programavimo kalba

**4 paskaita**

**(valdymo operatoriai, funkcijos )**

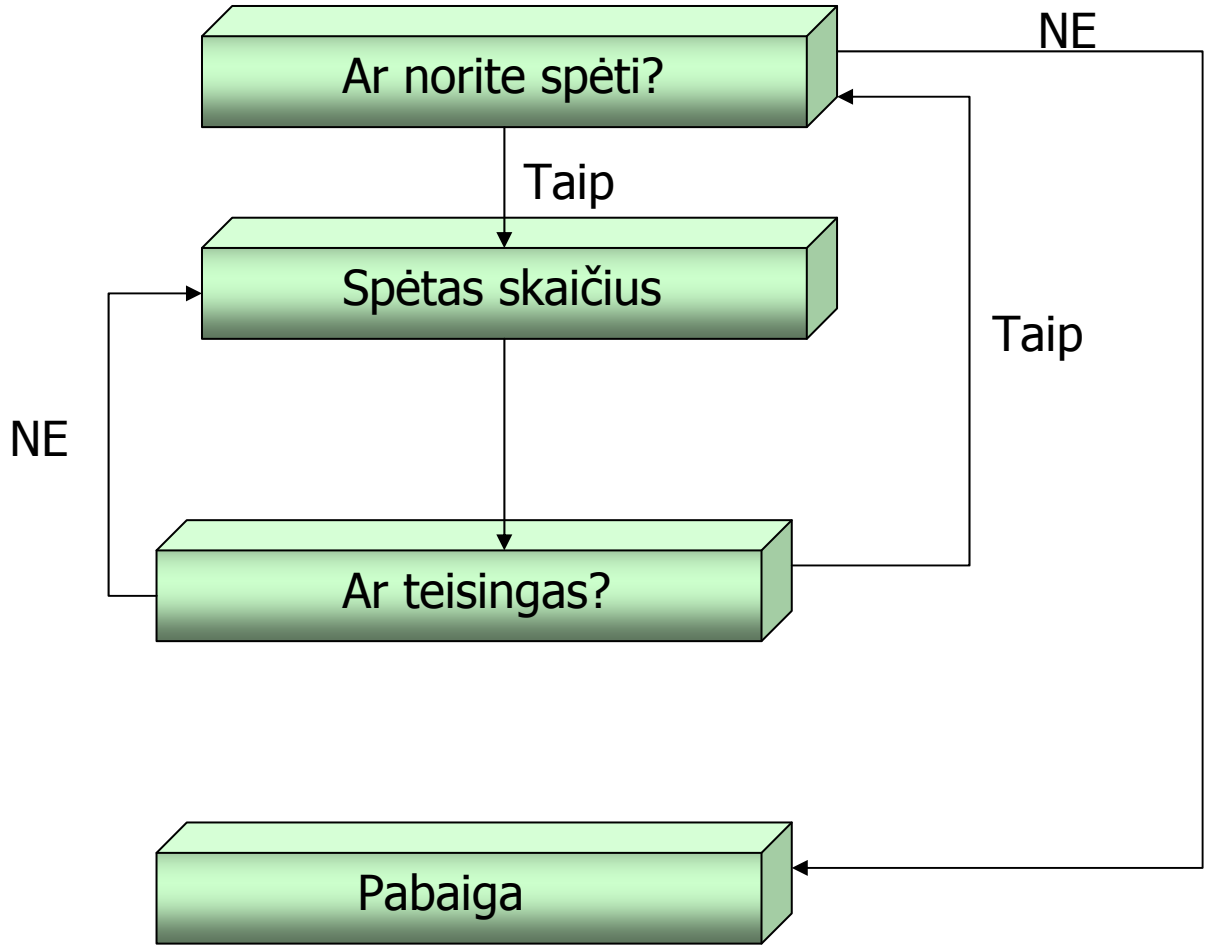
# Valdymo operatorius *brake*

Jei reikia išeiti iš ciklo *do*, *for*, *while* nesulaukus ciklo pabaigos, naudojamas operatorius *brake*. Šis operatorius naudojamas ir daugiavariantiniame sąlygos operatoriuje *switch*.

**Pavyzdys**

```
#include <stdio.h>
void main()
{ int answer, Sum = 0;
    do
     { printf ("Iveskite sveika teigiama skaiciu arba 0 darbui baigti:");
       scanf ("%d", &answer);
         if ( answer < 0 )
           { printf ( " Ivestas neigiamas skaicius. Programa baigia darba\n");
             break;}
       Sum +=answer;
      }
     while (answer)
       printf ("Skaiciu suma lygi %d\n"), Sum;
}
```

# Pavyzdys (atspėk skaičių)



# Pavyzdys (atspėk skaičių)

```
#include <stdio.h>
#include <stdlib.h>
main()
{ int target;
   int again;
   int guess;

   int counter = 0;
   printf("\n Ar spesite skaiciu 1,2,3… – taip, 0 – ne:  ");
   scanf("%d", &again);
   srand(again);
```

# Pavyzdys (atspėk skaičių)

```
while (again)
{ target = rand() % 100;
        printf('\n Spekite skaiciu: ");
        scanf("%d",&guess);
        while (target != guess)
          { if ( target > guess )
                printf ( “Per mazas “ );
            else
                printf(“Per didelis ");
            printf('\n Spekite dar karta: ");
            counter++;
            scanf("%d",&guess);
           }
  printf("\n Atspejai per %d kartus !  \n“, counter);
  printf("\n Ar spesi dar karta? 1,2,3... – taip, 0 - ne");
  scanf("%d“, &again); }

} }
```

# Operatorius *continue*

Operatorius *continue* naudojamas tuomet, kai reikia praleisti dalį ciklo operatorių ir pereiti prie naujos iteracijos.

**Pavyzdys** (daliklio tikrinimas):

```
#include <stdio.h>
void main()
{ int div0, div1; char ch;
   do
   { printf ("Iveskite skaiciu:"); scanf ("%d", &div0);
     printf ("Iveskite dalikli:");   scanf ("%d", &div1);
        if (div1 == 0 )
          { printf ("Blogas daliklis\n"); continue }
     printf ("Dalybos rezultatas yra %f", (float)div0/div1);
     printf ("Dar norite testi? ");
     scanf("%c",ch); }
   while (ch !='n')
}
```

# Operatorius *goto*

Šio operatoriaus paskirtis – perduoti programos vykdymą programinio kodo daliai, kuri pažymėta žyme. Žymė – tai žodis, kurio gale rašomas dvitaškis. Paprastai reikia vengti naudoti operatorių ***goto***.

**Pavyzdys:**

```
if (...)
{.....
    goto label;
}
.......
label:
   operatoriai
```

# Funkcijos

**Funkcija** – tai programos kodo dalis, turinti savo pavadinimą. Ši dalis gali būti iškviesta iš bet kurios kitos programos.

**Kodėl naudojamos funkcijos?**

- Sukuria aiškią programos struktūrą;
- Skaido programą į dalis;
- Leidžia lengvai panaudoti programos kodo dalį keletą kartų.

---

**Pavyzdys**

```
#include <stdio.h>
void starline ();
int main()
{  starline ();
    printf( "Tipas      Skaiciu ribos\n");
    starline ();
    printf( " int        -32768... 32767\n");
   return 0;
}
```

```
void starline ()
{ for (int i =0; i<30;i++)
   printf("*");
   printf("\n");
}
```

# Funkcijos prototipas

Prieš funkciją panaudojant programoje, ji turi būti deklaruota t.y. nurodytas jos prototipas. Dažniausiai tai daroma atskirame faile, tuomet pagrindiniame faile direktyvos **#include** pagalba prijungiamas failas, kuriame yra reikalinga funkcija.

Funkcijos prototipas:

**tipas Funkcijos_vardas (argumentų sąrašas);**

*tipas* – funkcijos grąžinamų duomenų (operatoriumi return) tipas;
Pagal nutylėjimą t.y. jei nenurodytas int

*argumentų sąrašas* – tai funkcijos parametų tipų ir vardų sąrašas. Galima nurodyti tik tipus. Sąrašas gali būti ir tuščias.

**Prototipų pavyzdžiai**

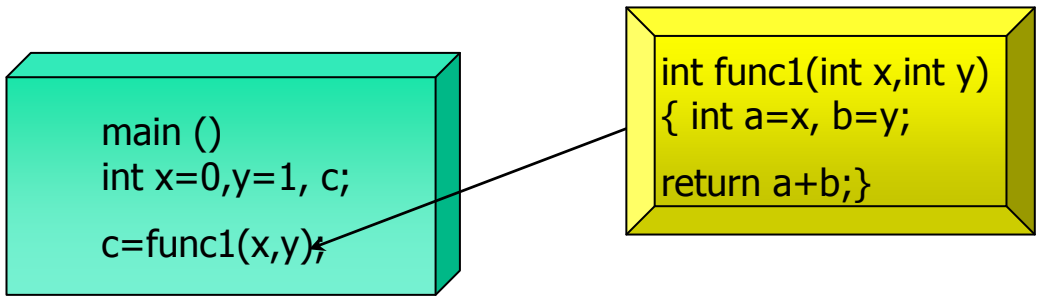
```
int swap (int, int);
double max(double par, double var);
```

# Funkcijos apibrėžimas ir iškvietimas

Funkcija apibrėžiama, kai be funkcijos antraštės užrašomas ir jos tekstas, apskliaustas figūriniais skliaustais. Jei funkcija yra ne **void** tipo, būtinai jos tekste turi būti operatorius **return**.

Funkcija iškviečiama nurodant funkcijos pavadinimą ir argumentų sąrašą, kurie paprastai būna kintamieji ar konstantos, perduodami funkcijai tolesniam panaudojimui.

Funkcijos gali iškviesti viena kitą, pagal sudarytą herarchiją. Funkcija gali iškviesti ir pačią save. Tokia konstrukcija vadinama rekursija.



# Argumentų perdavimas

**Argumentai** – tai duomenys, kuriuos programa perduoda funkcijai. Argumentai gali būti: *konstantos* ir *kintamieji*.

Funkcijos kuria (arba ne) perduodamų duomenų kopiją. Jei kopija kuriama, toks duomenų perdavimas vadinamas reikšmės perdavimu, jei ne – nuorodos perdavimu.

Pavyzdys

```
#include <stdio.h>

void repchar (char, int);

int main ()
{  char chin = '-'; int nin = 9;
    repchar (chin, nin);
    return 0;
}
```

```
// funkcija repchar ()

 void repchar (char ch, int n)
 {   for (int j = 0; j < n; j++)
          printf("%c", ch);
 }
```

# Argumentai pagal nutylėjimą

C/C++ iškviečiant funkciją galima dalį argumentų praleisti, jei funkcijos prototipe nurodytos argumentų reikšmės pagal nutylėjimą.

**Pavyzdys**

```
int Showint (int i, bool zyme = true, char simbolis = '\n');

// Funkcijos Showint iškvietimas

   Showint (1, false, 'a');
   Showint (2, false);
   Showint (3);
```

# Funkcijos grąžinama reikšmė

Funkcija baigusi darbą gali grąžinti vieną reikšmę ją iškvietusiai programai. paprastai tai atsakymas į problemą, kurią išsprendė funkcija. Grąžinamos reikšmės tipas ir funkcijos tipas, nurodytas prototipe turi sutapti.

Jei funkcijos tipas nenurodytas, laikoma, kad tai **int** tipo funkcija.

```
#include <stdio.h>
float lbstokg(float);
int main()
{  float lbs, kgs;
    printf ("Iveskite svori svarais:");
    scanf ("%f",&lbs);
    kgs = lbstokg(lbs);
    printf("Perskaiciuotas svoris
            kilogramais %f \n", kgs);
    return 0;
}
```

```
// funkcija lbstokg ()

float lbstokg(float svarai)
{
   float kilog = 0.45359*svarai;
    return kilog;
}
```

# Lokalūs ir globalūs kintamieji

Lokalūs kintamieji – tai kintamieji apibrėžti funkcijos tekste. Jie galimi tik tos funkcijos ribose. Kai funkcija grąžina valdymą, ją iškvietusiai programai, lokalūs kintamieji naikinami.

Kiekvieną kintamąjį apibrėžia jo veikimo ribos ir gyvavimo trukmė.

Kintamojo veikimo ribos – tai programos dalis, kurioje galima naudoti kintamąjį.

Kintamojo gyvavimo trukmė – tai programos vykdymo intervalas, kurio metu kompiuterio atmintyje egzistuoja kintamasis.

Globalūs kintamieji – tai kintamieji apibrėžti ne kokioje nors funkcijoje ir gyvuojantys per visą programos laiką.

# Lokalūs ir globalūs kintamieji

```
// lokalūs kintamieji

#include <stdio.h>
int Sum (int, int);

int main()
{ // lokalus x
     int x =2, y=4;
    printf("Suma %d\n",Sum(x,y));
  return 0;
}

int Sum (int a, int b)
{// lokalus x
  int x = a +b;
  return x;
}
```

```
// globalus kintamasis

#include <stdio.h>
void Print (void);

int Test = 200; //globalus kintamasis

int main()
{ int Test = 4;
   printf("Lokalus %d\n", Test);
   Print();
  return 0;
}

void Print (void)
{ printf("Globalus %d\n", Test);
  }
```

# Operacija ::

C/C++ leidžia bet kurioje programos vietoje išsikviesti globalius kintamuosius. Tam naudojami :: prieš globalaus kintamojo pavadinimą.

```
#include <stdio.h>
void Print (void);

int Test = 200;   //globalus kintamasis

int main()
{ int Test = 4;
   printf("Lokalus %d\n", Test);
   printf("Globalus %d\n", ::Test);
  return 0;
}
```

# Kintamųjų modifikatoriai

| Modifikatorius | Taikymas | Veikimo sritis | Gyvavimo laikas |
|---|---|---|---|
| **Auto** | Lokalus | Blokas | Laikinas |
| **Register** | Lokalus | Blokas | Laikinas |
| **Extern** | Globalus | Blokas | Laikinas |
| **Static** | Lokalus/Globalus | Failas | Pastovus |
| **volatile** | Globalus | Failas | Pastovus |