

REKURSINĖS FUNKCIJOS

REKURSIJA.

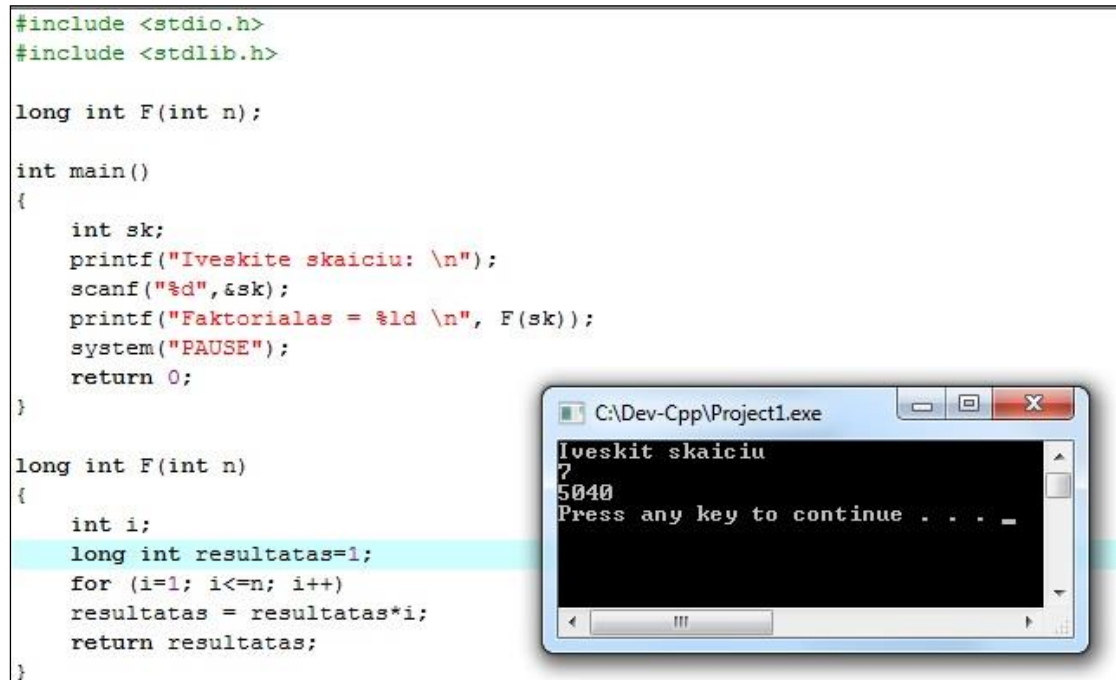
C/C++ kalboje funkcija arba procedūra (paprogramė) savo veiksmų dalyje (bloke) gali turėti kreipinį į save pačią. Tokios paprogramės vadinamos *rekursinėmis*. Programuotoju požiūriu rekursija – tai dar vienas būdas kartoti veiksmus (panašiai kaip ciklai). Iš tikrųjų, rekursija galima būtų pakeisti ciklu ir atvirkščiai, tačiau kiekvienas būdas turi savo pranašumų ir trūkumų. Pailiustruokime tai dviejų programų pavyzdžiu, t.y. realizuodami tam tikrą skaičiavimo uždavinį ciklu ir rekursija. Apskaičiuokime faktorialą. Priminsime, kad *faktorialas* yra natūraliųjų skaičių sandauga:

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n.$$

Pirma, realizuokime faktorialo skaičiavimą naudojant ciklą. Skaičiuojant sandaugą reikia priskirti pradinę reikšmę 1 ir toliau pasinaudoti ciklu su žinomu skaičiavimų skaičiumi (t.y., **for**):

```
faktorialas=1;
for (i=1; i<=n; i++)
    faktorialas = faktorialas*i;
```

Pateiksime visos programos ir jos vykdymo rezultato vaizdą C/C++ programavimo terpėje (žr. 7.1 pav.):



```
#include <stdio.h>
#include <stdlib.h>

long int F(int n);

int main()
{
    int sk;
    printf("Iveskite skaiciu: \n");
    scanf("%d",&sk);
    printf("Faktorialas = %ld \n", F(sk));
    system("PAUSE");
    return 0;
}

long int F(int n)
{
    int i;
    long int rezultatas=1;
    for (i=1; i<=n; i++)
        rezultatas = rezultatas*i;
    return rezultatas;
}
```

The console window shows the following output:

```
C:\Dev-Cpp\Project1.exe
Iveskit skaiciu
?
5040
Press any key to continue . . .
```

7.1. pav. Faktorialo skaičiavimas naudojant ciklą.

Antra, realizuokime faktorialo skaičiavimą naudojant rekursiją. Užrašykime faktorialą rekursiniu būdu, pastebėję, kad iš

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$$

seka, kad

$$n! = (n - 1) ! \cdot n.$$

Taigi, faktorialas gali būti išreiškiamas faktorialu nuo vienetu mažesnio skaičiaus ir daugyba. Nustatę ribinę faktorialo reikšmę $0! = 1$, gauname rekursinį faktorialo apibrėžimą:

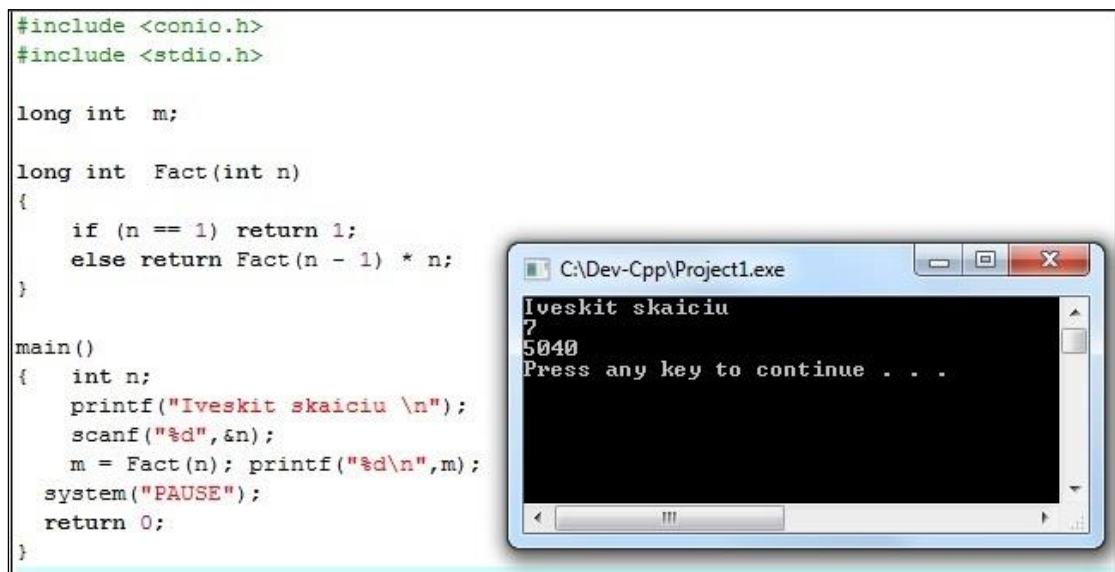
$$n! = 1, \quad \text{jei } n=0,$$

$$n! = (n - 1) ! \cdot n, \quad \text{jei } n>0.$$

Pastebėjime, kad toks rekursinis apibrėžimas ypatingas tuo, kad funkcijos (faktorialo) apibrėžime vėl naudojama ta pati funkcija, bet su vienetu mažesniu argumentu ($n-1$). Realizuokime šią rekursinę funkciją programoje:

```
int Fact(int n)
{
    if (n == 0) return 1;
    else return Fact(n - 1) * n;
}
```

Pateiksime visos programos ir jos vykdymo rezultato vaizdą C/C++ programavimo terpėje (žr. 7.2 pav.):



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

long int m;

long int Fact(int n)
{
    if (n == 1) return 1;
    else return Fact(n - 1) * n;
}

main()
{
    int n;
    printf("Iveskit skaiciu \n");
    scanf("%d", &n);
    m = Fact(n); printf("%d\n", m);
    system("PAUSE");
    return 0;
}
```

7.2. pav. Faktorialo skaičiavimas rekursiniu būdu.

Matome, kad rekursinei faktorialo funkcijai neprireikė ciklo, t. y. joje veiksmai kartojami nenaudojant ciklo, o naudojant kreipinį į save ir mažinant funkcijos argumentą vienetu. Toks pakartotinis kreipinys į save bus vykdomas tol kol argumentas n pasidarys lygus 0 ir funkcijai bus priskirta reikšmė 1.

Rekursinė faktorialo realizacija – tai tarsi atitinkamo matematinio apibrėžimo vertimas (atvaizdavimas) į programavimo kalbą. Iš tikrųjų, rekursija leidžia sudėtingo uždavinio sprendimą pavaizduoti kaip vieno tipinio elementaraus (arba bent mažesnio)

žingsnio atkartojimą. Būtent rekursijos vaizdumas, jos atitikimas matematiniam aprašui yra jos pagrindinis pranašumas.

Tačiau rekursija paprastai reikalauja daugiau atminties ir procesoriaus resursų nei ciklai, nes rekursinių programų vykdymo metu vyksta daugkartinis paprogramių (procedūrų ar funkcijų) iškvietimas. Taigi, nėra vieno universalaus recepto kaip rasti geriausią programos vaizdumo ir ekonomiškumo santykį, t. y. pasirinkti rekursiją ar ciklą. Tai sprendžia pats programuotojas atsižvelgdamas į konkretaus uždavinio sprendimo algoritmo savybes.

HANOJAUS BOKŠTO UŽDAVINYS

Tai, iš tikrųjų ne legenda, o uždavinys, kurį 1883 metais suformulavo prancūzų matematikas Eduardas Lukas (*Edouard Lucas*). Originali formuluotė yra tokia:

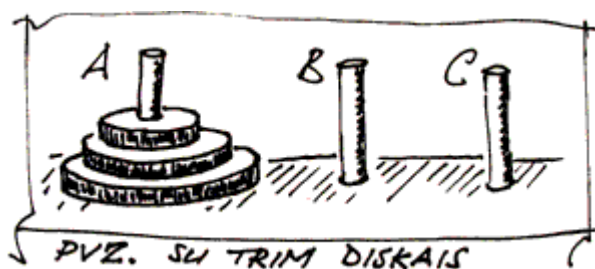
Duoti trys stiebai ir aštuoni skirtingo dydžio diskai. Iš pradžių visi šie diskai sumauti ant pirmojo stiebo: apačioje pats didžiausias diskas, ant jo – mažesnis ir t. t. Viršuje užmautas pats mažiausias iš diskų. Uždavinys prašo perkelti visus diskus nuo pirmojo stiebo ant paskutiniojo laikantis tokių taisyklių:

- *vienu ėjimu galima kelti tik vieną diską;*
- *diską galima užmauti tik ant tuščio stiebo, arba uždėti ant didesnio už jį disko.*

Bendru atveju vietoje aštuonių, reikia perkelti N diskų (tai vienintelis pradinis duomuo, dėl praktinių priežasčių: $1 \leq N \leq 15$), o stebus galima vadinti raidėmis A, B ir C (arba skaičiais 1, 2 ir 3).

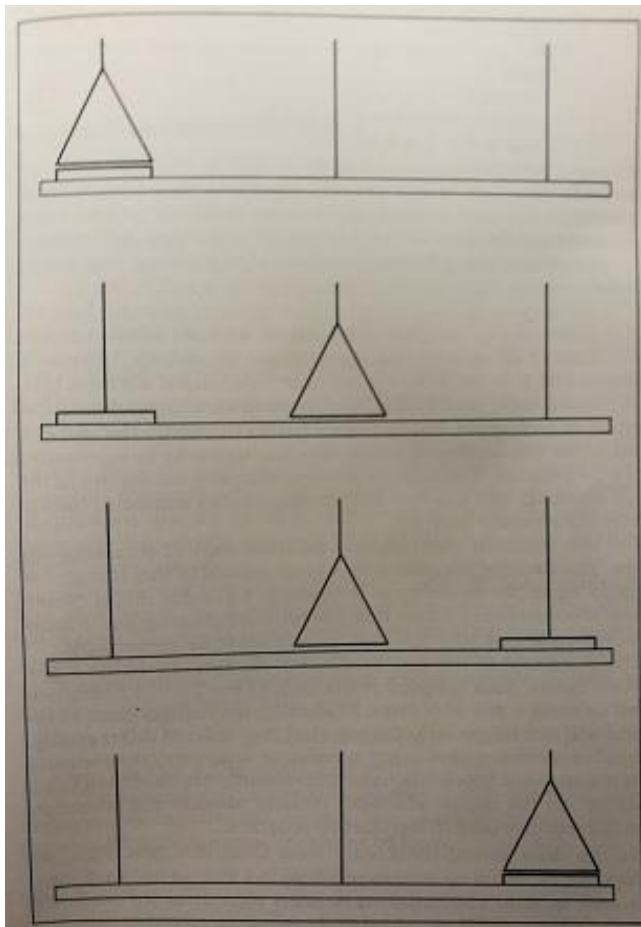
Parašykite rekursinę programą, kuri atspausdintų ėjimus, kuriuos reikia atlikti, kad diskus perkeltume nuo stiebo A ant stiebo C, laikantis minėtųjų taisyklių. Atliekamų ėjimų skaičius turi būti minimalus. Programos rezultatą turi sudaryti seka ėjimų, kuriuos atlikus, visi diskai būtų perkelti nuo stiebo A ant stiebo C.

Sprendimo pavyzdys



Kai $n = 1$, diską perkeliame (ir uždavinį išsprendžiame) vienu žingsniu. Taip pat nesunku jį išspręsti, kai $n = 2$, tam tereikia trijų perkėlimų. Atitinkamai, 7 perkėlimų pakanka, kai $n = 3$. Kodėl? Bandykime rasti rekursinį sprendimą. Prieš tai pastebėkime, kad niekas nepasikeistų, jei uždavinyje būtų reikalaujama diskus perkelti ne ant dešiniojo, o ant vidurinio disko: atliktume tuos pačius ėjimus, tik diskus keltume ne ant dešiniojo, o ant vidurinio stiebo ir atvirkščiai.

Ieškant rekursinio sprendimo, panagrinėkime uždavinį nuo galo, t. y. nuo paskutinio ir priešpaskutinio ėjimų: ko reikia kad galėtume perkelti patį didžiausią diską (n -ąjį diską)? Aišku, kad ant jo neturi būti jokių kitų diskų, o dešinysis stiebas taip pat turi būti laisvas. Reiškia, prieš tai visi kiti mažesni diskai (jų yra $n-1$) jau turi būti perkelti ant vidurinio stiebo (B). Bandydami ($n - 1$) tai padaryti galime visiškai nekreipti dėmesio į didžiausią diską: jis nesutrukdytų, kadangi yra didesnis už visus likusius diskus. Taigi matome, kad ($n - 1$) disko perkėlimas yra visiškai tas pats, tik sumažintas, uždavinys:



Tokiu būdu rekursinio uždavinio sprendimą galima aprašyti tokia bendra schema:

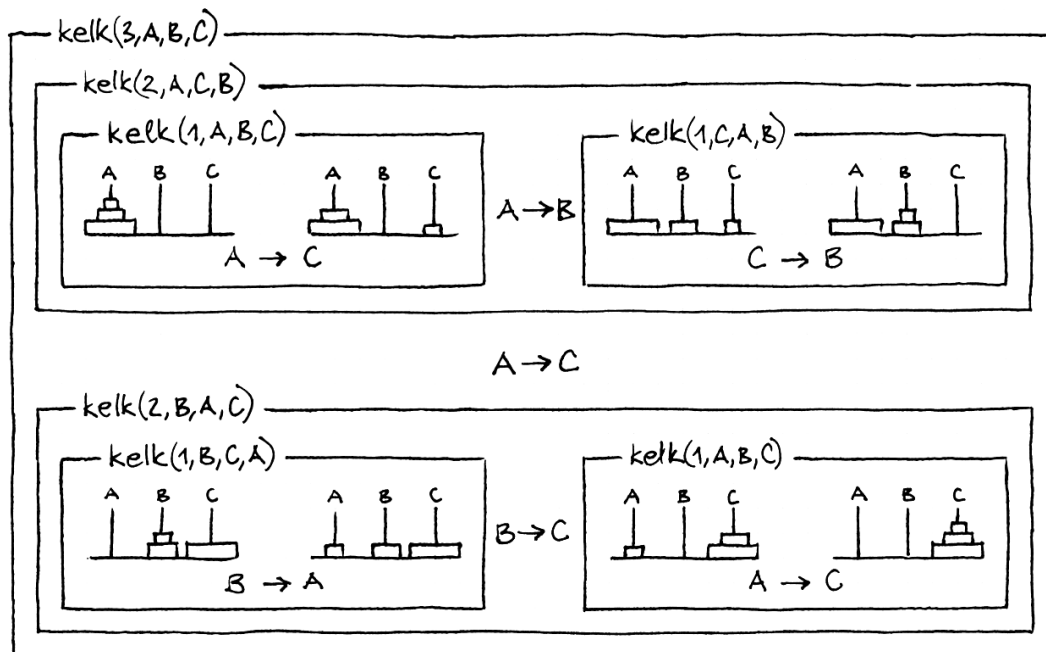
Jei norime perkelti tam tikrą skaičių diskų ($n > 0$):

- *visus mažesnius diskus perkeliame ant tarpinio stiebo;*
- *perkeliame n -ąjį (didžiausią) diską.*
- *visus mažesnius diskus perkeliame ant galinio stiebo.*

Reiškia, rekursinė diskų perkėlinėjimo funkcija turi priklausyti nuo

diskų skaičiaus ir turi žinoti, nuo kurio ir ant kurio stiebo norima perkelti diskus, o kurį stiebą naudoti kaip tarpinį.

Rekursinio sprendimo pavyzdys (3 diskai):



Programos darbo rezultatai:

$A \rightarrow C$
 $A \rightarrow B$
 $C \rightarrow B$
 $A \rightarrow C$
 $B \rightarrow A$
 $B \rightarrow C$
 $A \rightarrow C$

Rekursinės sprendimo funkcijos pavyzdys

Hanojaus bokšto sprendimo vizualizavimo programos fragmentas.

```
31 // ----- */
32 /* ----- */
33 /* Funkcija perkelia viena zieda */
34 /* is stovykl n1 i stovpa n2 */
35 /* ----- */
36 void move1(int n1, int n2)
37 {
38     st[n2][nr[n2]++] = st[n1][--nr[n1]];
39     //sleep(1); /* 1 sekunda lauka, funkcija is dos.h */
40     print_st(); /* dabartines pozicijos isvedimas */
41     nmoves++;
42 }
43 /* ----- */
44 /* Funkcija hanoi perkelia virsutiniu prings ziedus */
45 /* is stovpa i1 i stovpa i3, naudodama stovpa i2 kaip tarpini. 1 <= i1,i2,i3 <= 3. */
46 /* ----- */
47 void hanoi(int nrings, int i1, int i2, int i3)
48 {
49     if(nrings == 1)
50         move1(i1, i3);
51     else {
52         hanoi(nrings-1, i1, i3, i2);
53         move1(i1, i3);
54         hanoi(nrings-1, i2, i1, i3);
55     }
56 }
57
58 main()
59 {
60     int nrings;
61     printf("Ziedu skaicius: "); scanf("%d", &nrings); init(nrings);
62     hanoi(nrings, 1, 2, 3);
63     printf("Ziedu perkeltas baigtas.\n");
64     printf("Perkelimo operaciju skaicius - %d.\n", nmoves);
65     getchar(); getchar();
66     return(0);
67 }
68
69
```

Ziedu skaicius: 3

3 2 1

3 2

1

3

2

1

3

2 1

2 1

3

1

2

3

1

3 2

3 2 1

Ziedu perkeltas baigtas.

Perkelimo operaciju skaicius - 7.

Build file: "no target" in "no project" (compiler: unknown) -

Build finished: 0 error(s), 0 warning(s) (0 minute(s), 0 sec