

Лекция 9.

Рекурсия. Рекурсивни обекти и процеси. Рекурсивни подпрограми

*Един начин да намалим
сложността*

Рекурсия: Същност и примери

- **Произход на думата “рекурсия” (*recursion*):**

На английски: *recur* — връщам се, повтарям се, случвам се отново (на повтарящи се интервали)

Има латински произход: (*re* = обратно) + (*currere* = бягам)

- **Примери на рекурсивни явления:**

- ако в телевизионно студио камерата снима изображението на телевизор, който излъчва същото предаване, то на екрана на домашния телевизор се вижда картина, която се съдържа в себе си отново и отново
- ако някакъв предмет е поставен между две огледала, намиращи се едно срещу друго, то и в двете огледала отражението представлява рекурсивно изображение
- приказката: “Имало едно време един поп. Той си имал кученце. Попът умрял и на гроба му пишело: “Имало едно време ...”. Това е приказка, която се съдържа сама в себе си т.е. “рекурсивна приказка”

- **Определение:**

Рекурсивен обект или процес е такъв, който частично или изцяло се съдържа в себе си или се дефинира чрез себе си

Рекурсия: Примери от математиката

Рекурсията широко се използва в математиката за дефиниране на математически понятия или функции:

А) Рекурсивна дефиниция на понятието **естествено число**:

1. 1 е естествено число

2. Ако N е естествено число, то $N+1$ е естествено число

Б) Рекурсивна дефиниция на **функцията за $N!$** (N факториел):

1. $0! = 1$

2. $N! = (N-1)!N, \quad \forall N > 0$

В) Рекурсивна дефиниция на **редицата от числата на Фибоначи** $f_0, f_1, \dots, f_{N-1}, f_N, f_{N+1}, \dots$:

1. $f_0 = 0, f_1 = 1$

2. $f_{N+1} = f_N + f_{N-1}, \quad \forall N > 0$

Пример за първите няколко ЧФ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,...

 Рекурсията представлява лаконичен начин за задаване на множество от безброй много обекти или безкрайни процеси

Рекурсията в информатиката


В информатиката рекурсията се използва за:

- описание на данни – **рекурсивни структури** от данни т.е. структури, които се съдържат в себе си:

Рекурсивен тип данни T е съставен тип, който съдържа компоненти от типа T (напр. масив от масиви, списък, стек и т.н.)

- за описание на обработката на данни – **рекурсивни алгоритми**, т.е. алгоритми, които се описват чрез себе си

 Тук се разглеждат само възможностите за компютърно моделиране на рекурсивни алгоритми


 Средството в ЕП, което осигурява възможност за представяне на рекурсивни алгоритми са подпрограмите

Рекурсивни подпрограми – първи пример на C++

- **Определение:** Рекурсивна подпрограма е такава подпрограма, която директно или косвено извиква себе си
- **Пример:** Рекурсивна функция, която връща n-то число от редицата на Фибоначи

Програмна реализация

```
int fib(int x) {  
    if (x == 0)  
        return 0;  
  
    if (x == 1)  
        return 1;  
  
    return fib(x-1)+fib(x-2);  
}
```

 Основно изискване при съставяне на компютърни рекурсивни алгоритми е те да бъдат **крайни**

Рекурсивни подпрограми – втори пример на C++



Съставяне на рекурсивна подпрограма е изключително лесно, ако предварително за съответния алгоритъм е намерена рекурсивна дефиниция

- **Пример:** Програма за намиране на $N!$ по дадено $N \geq 0$ с използване на рекурсивна подпрограма-функция

```
int factorialfinder(int x)
{
    if (x == 1)
    {
        return 1;
    }else
    {
        return x*factorialfinder(x-1);
    }
}
```

- **Анализ** на рекурсивната дефиниция на $N!$ и съответния алгоритъм за решаване на задачата:
 - задачата зависи от един **параметър** – цяло неотрицателно число
 - чрез т. 2 от дефиницията, **задачата е разбита на подзадачи**, като подзадачата за стойност N на параметъра (с размерност N) е изразена чрез друга подзадача за стойност $N-1$ на параметъра (с размерност $N-1$)
 - чрез т. 1 от дефиницията е **осигурен край на алгоритъма**, тъй като при $N=1$ стойността на $N!$ се намира непосредствено без помощта на рекурсия

Съставяне на рекурсивни алгоритми

Общи стъпки при съставяне на рекурсивни алгоритми:

1. Определят се параметрите от които зависи задачата (параметризация)
2. Извършва се тъй наречения рекурсивен анализ, т.е. задачата се разбива на подзадачи, като всяка подзадача се описва чрез един или повече случаи на същата задача, но за други (обикновено по-малки) стойности на параметрите (редукция)
3. Осигурява се край на рекурсията, т.е. намира се такава подзадача, чието решение не е рекурсивно т.е. води до завършване на алгоритъма. Определят се стойностите на параметрите на алгоритъма, на които тази подзадача съответства (условие за край)

Съставяне на рекурсивни алгоритми – трети пример на C++

- **Условие:** Да се напише програма, която отпечатва двоичното представяне на дадено десетично число $N > 0$ (като се използва рекурсия)
- **Алгоритмизация:** Извършва се целочислено деление на 2 докато се получи частно 0 (най-напред даденото десетично число N , а след това всяко новополучено целочислено частно). Получената в резултат на това редица от остатъци, записана в обратен ред е търсеното двоично представяне на N .
Например:

$6 : 2 = 3$	$\begin{array}{ c } \hline 0 \\ \hline \end{array}$	\uparrow
$3 : 2 = 1$	$\begin{array}{ c } \hline 1 \\ \hline \end{array}$	
$1 : 2 = 0$	$\begin{array}{ c } \hline 1 \\ \hline \end{array}$	

$6_{(10)} = 110_{(2)}$

Съставяне на рекурсивния алгоритъм (съгласно общите стъпки):

1. **Параметризация:** Поставената задача зависи от един параметър $N_{(10)} > 0$ – цяло
2. **Редукция:** Вярно е, че ако знаем двоичното представяне на $(N/2)$, т.е то вече е отпечатано, тогава за да получим двоичното представяне на N остава да се отпечата и остатъкът $(N \% 2)$
3. **Условие за край:** Поредицата от целочислени деления завършва, когато се получи частно 0 (при стойност на параметъра $N = 0$)

Програмна реализация на примера

В следващата програма рекурсивният алгоритъм е реализиран чрез рекурсивната функция `Dec_Bin`

🔊 Единствената особеност е, че отпечатването на поредния остатък се извършва след рекурсивното извикване на `Dec_Bin`

```
//примерна програма за рекурсивна функция
```

```
#include <iostream.h>
```

```
void Dec_Bin(int N)
```

```
//рекурсивна процедура за отпечатване на двоичното представяне на N
```

```
{  
    if (N>0)  
    {  
        Dec_Bin(N / 2);  
        cout << N % 2;  
    };  
}
```

```
void main()
```

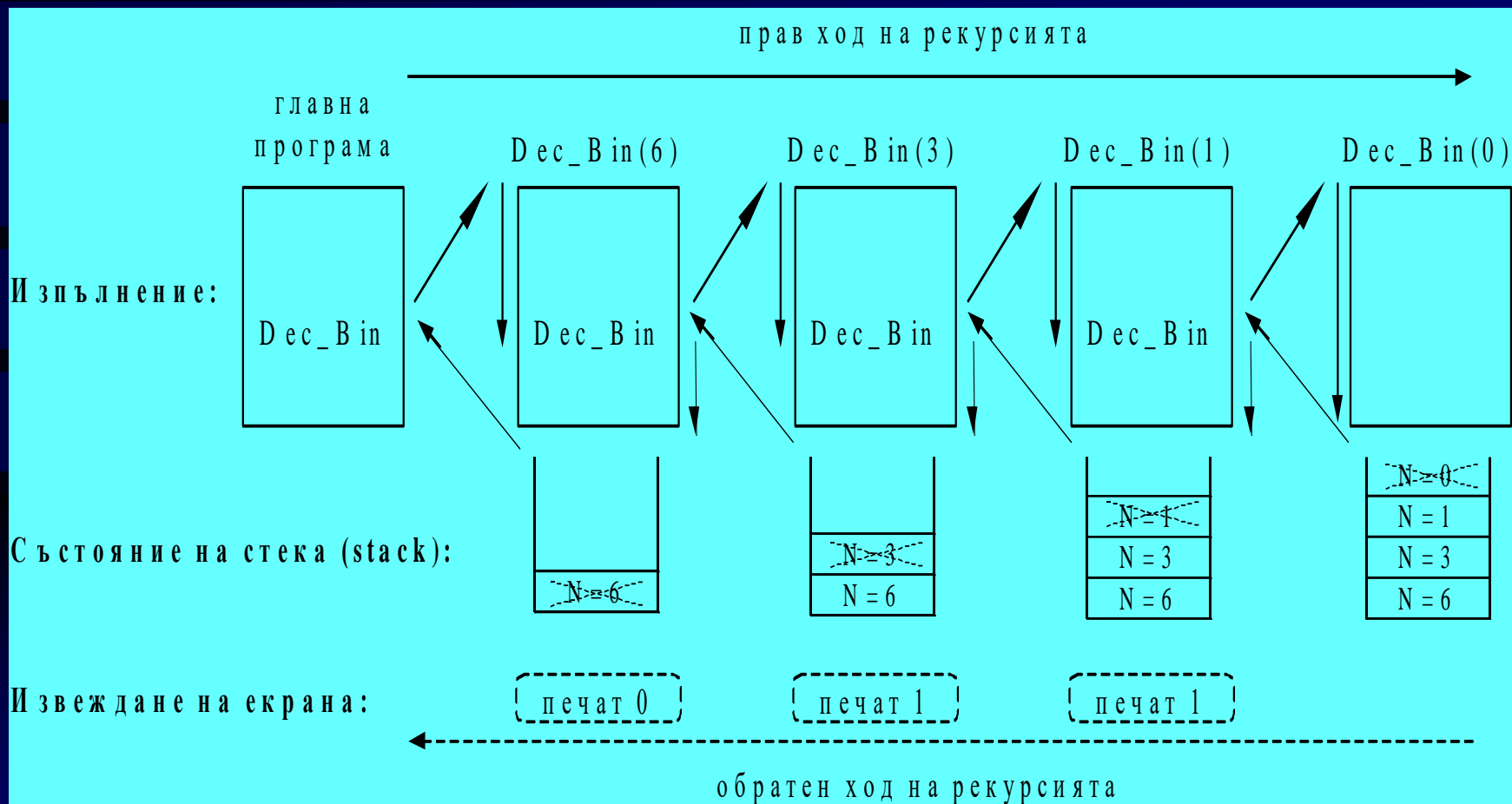
Функции в C++: Изпълнение на функции – поглед отвътре

Изпълнението на всяка функция се извършва по следната схема:

- (1) При активиране на функцията в стековата памет се съхранява запис за текущото ѝ състояние. Записът съдържа:**
 - стойностите на фактическите параметри
 - стойностите на локалните променливи на функцията
 - адресът за връщане
- (2) Функцията се изпълнява**
- (3) След приключване на изпълнението, записът от стековата памет се изчиства и по адреса за връщане изпълнението на програмата продължава с оператора следващ след оператора за активиране на функцията**

Функции в C++: Изпълнение на рекурсивни функции – поглед отвътре

На схемата е изобразено изпълнението на рекурсивната ф-я Dec_Bin, ако за Decimal е въведена стойност 6:



Опашата рекурсия

- **Прав ход на рекурсията:** Това са всички действия, които се извършват, докато условието за край на рекурсията настъпи (с плътна линия на схемата от слайд 11) – изпълняват се всички оператори от рекурсивната ф-я, които се намират преди рекурсивното извикване, толкова пъти, колкото извиквания настъпват и за стойности на параметрите съответни на реда на извикванията
- **Обратен ход на рекурсията:** Това са всички действия, които се извършват след като условието за край на рекурсията настъпи (с пунктирна линия на схемата от слайд 11) – изпълняват се всички оператори от рекурсивната ф-я, намиращи се след рекурсивното извикване, като съответните стойности на параметрите са в ред, обратен на реда на извикванията
- **Опашата рекурсия:** Рекурсия, при обратния ход, на която се изпълняват някакви действия се нарича *опашата рекурсия*

Рекурсия и итерация

Рекурсията, като средство за организация на обработката на данните е сходна в голяма степен с **итерацията (цикъл)**, тъй като дава възможност за описание на повтарящи се действия



В повечето случаи дадена задача има, както рекурсивно, така и итерационно решение:

```
int N, Fact=1;  
// ...  
for (int i=N; i>0; i--) Fact*=i;
```



Понякога рекурсивното решение е неефективно, в сравнение със съответното итерационно решение, защото изпълнението на рекурсивни ф-ии е свързано с голям разход на ресурси (при всяко рекурсивно извикване в стека се прави нов запис на текущото състояние на ф-ята, създават се и се унищожават локални променливи и т.н.)

Кога се препоръчва да се използва рекурсия

- Сравнение на итерационното и рекурсивното решение на задачата за $N!$:

Ресурс	Нерекурсивно решение	Рекурсивно решение
Операции (бързодействие) – брой умножения	N (брой повторения на цикъла)	N (брой рекурсивни извиквания)
Памет – брой целочислени променливи	2 (за N и $N!$)	N (брой записи в стека)

- Правило:** Ако при съпоставяне на рекурсивното и нерекурсивното решение на дадена задача, количеството необходими за изпълнение ресурси е сравнимо, то се препоръчва да се използва рекурсията, тъй като тя е по-лаконичен и по-ясен начин за описание на алгоритми



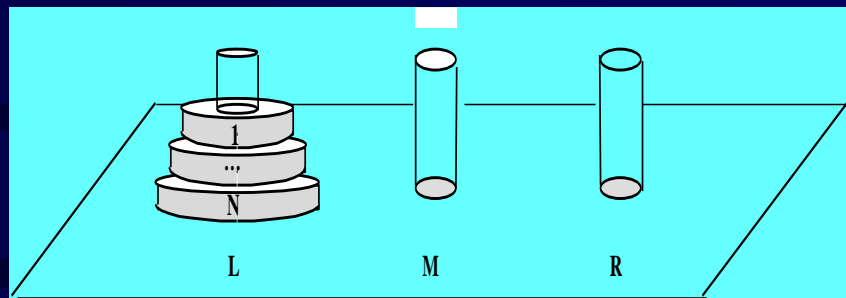
Рекурсивното решение (Dec_Bin) на задачата за двоичното представяне на десетично число е достатъчно ефективно:

Ресурс	Нерекурсивно решение	Рекурсивно решение
Операции (бързодействие) – брой деления	$\log_2 N + 1$ (брой повторения на цикъла)	$\log_2 N + 1$ (брой рекурс. извиквания)
Памет – брой променливи	$\log_2 N$ (за съхранение на остатъците)	$\log_2 N + 1$ (брой записи в стека)

- Ефективността на алгоритмите се оценява обикновено по отношение на два фактора – **бързодействие** (брой трудоемки операции) и необходима **памет**

Задача за Ханойските кули – пример за предимствата на рекурсията

- **Условие:** Дадени са 3 пилон, означени условно с L – ляв (left), M – среден (middle) и R – десен (right) и N на брой дискове с различни диаметри, разположени, както е показано на схемата:



Целта е да се преместят всички дискове от левия пилон на десния пилон, така че да бъдат подредени в същата последователност, като се спазват следните правила:

- При всяко преместване се мести само 1 диск;
- Диск може да се поставя само върху празен пилон или върху по-голям диск;
- При преместване на дискове от един пилон на друг, третият пилон може да се използва като междинен (помощен).

🔊 Тази задача произхожда от една легенда, която разказва, че в един Ханойски храм няколко китайски монаси решават на практика тази задача за $N=64$ диска и когато завършат преместването на дисковете ще настъпи край на света (Може да се изчисли, че ако едно преместване се извършва дори само за 1 микросекунда= 10^{-6} секунди, то са необходими 300000 години, за да се реши практически задачата)

Задача за Ханойските кули — алгоритмизация

Ако дисковете се номерират с $N, N-1, N-2, \dots, 1$ по намаляване на диаметъра, то:

1. **Параметризация:** Поставената задача зависи от един параметър — цяло положително число N (брой дискове).
2. **Редукция:** Ако преместим по-малките $N-1$ диска от левия на средния пилон (като използваме десния за междинен), то за да решим задачата остава да преместим N -тия (най-голям) диск от левия на десния пилон, след което да преместим по-малките $N-1$ диска от средния на десния пилон (като използваме левия за междинен).
3. **Условие за край:** При брой дискове $N=0$ премествания не се извършват (т.е. рекурсията завършва при стойност на параметъра $N = 0$).

Задача за Ханойските кули



Задача за Ханойските кули – Програмна реализация

В следващата програма рекурсивният алгоритъм е реализиран чрез рекурсивната функция `hanoi_towers`

```
void hanoi_towers(int quantity, int from, int to, int buf_peg)
{
    if (quantity != 0)
    {
        hanoi_towers(quantity-1, from, buf_peg, to);

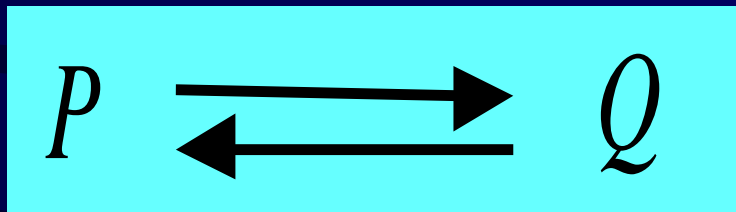
        cout << from << " -> " << to << endl;

        hanoi_towers(quantity-1, buf_peg, to, from);
    }
}
```

Непряко рекурсивни подпрограми

При непряката (косвена) рекурсия една подпрограма активира друга подпрограма, която от своя страна активира първата – **взаимно се извикват** (би могло рекурсивната връзка да е още по-опосредствана)

- **Схема:** Ако подпрограмите са съответно P и Q , то P вика Q и Q вика P :



- **Проблем:** При реализацията на P и Q , като взаимно-рекурсивни функции на C++, която и от двете да декларираме първа ще получим съобщение за грешка защото е направен опит да се извика функция, преди все още да е дефинирана
- **Решение на проблема:** За поне една от функциите се декларира прототип

Непряко рекурсивни подпрограми – пример

```
void Recur2(int max, int& num);  
void Recur1(int max, int& num)  
{ if (num<max) {  
    num++; cout<<"First function: ";  
    Recur2(max, num);  
}  
}  
void Recur2(int max, int& num)  
{ if (num<max) {  
    num++;      cout<<"Second function: ";  
    Recur1(max, num);  
}  
}  
void main()  
{ int count=0; Recur1(10,count); }
```