Rekurze v programování

Jitka Kreslíková, Aleš Smrčka

2021

Fakulta informačních technologií
Vysoké učení technické v Brně

IZP – Základy programování

Rekurzivní metody v programování

- Definice rekurze
- Rekurze v programování
- Rekurze a iterace
- Rekurze a zásobníková paměť
- Rekurzivní funkce

Rekurze v programování

- Co je rekurze?
  - Způsob specifikace entity odkazem na sebe sama.
- Rekurzivní funkce
  - Funkce, která je definována pomocí volání sebe samé.

    Příklad: definice faktoriálu

    faktoriál (0) = 1

    faktoriál (n) = n × faktoriál (n−1), pro n > 0
- Každá rekurzivní definice musí mít explicitní definici pro alespoň jednu hodnotu argumentu, jinak by šlo o kruhovou definici.

Rekurze v programování

- Podmíněný výraz zápisu rekurze:

  $[b_1 \to e_1,\ \ b_2 \to e_2, \ldots b_{n-1} \to e_{\ n-1}, e_n]$

  b_i - označuje logickou podmínku,

  e_i - označuje výraz.
- Hodnota výrazu se získá hledáním pravdivé b_i zleva a výsledkem pak je odpovídající e_i.

Příklad: definice funkce faktoriál.

faktoriál (n) = $[(n=0) \to 1, n \times$ faktoriál(n-1)$]$

Rekurze v programování

Příklad: rekurzivní funkce pro výpočet faktoriálu.

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

Rekurze v programování

- Rekurze
  - Programovací technika – opakované použití programové konstrukce při řešení téže úlohy uvnitř téže konstrukce.
  - Musí obsahovat ukončovací podmínku.
  - Používá se tam, kde lze tentýž postup řešení aplikovat na každou podúlohu.
  - Některé úlohy a datové struktury jsou rekurzivní samy o sobě
    - Seznam – struktura, kde prvek obsahuje ukazatel na zbytek seznamu
  - Každou rekurzi lze nahradit iterací (a naopak).

Rekurze a iterace

- Iterace
  - Cyklus – sekvence příkazů vykonávaná opakovaně na základě testu podmínky.
  - Podmínka ukončení je nezbytná, jinak by vznikl nekonečný cyklus. Může se objevit na začátku nebo na konci cyklu.
  - Rekurze – zobecněná iterace – podmínka ukončení se může vyskytovat v rekurzivní definici kdekoli.
  - Iterace bývá efektivnější, rekurzi lze popsat průzračnějším algoritmem.

Rekurze a iterace

Příklad: uvažujme cyklus:

```
int i = 1;
while (i <= N)
{
  //sekvence příkazů Si
  i++;
}
```

```
void tisk(int n)
{
  if (n > 1) tisk(n-1);
  printf("%d ", n);
  return;
}
//  1 2 3 . . . . n
```

- Příkazy S_i nechť proměnnou i nemění. Iterace zde proběhne jako posloupnost bloků:

  S_1; S_2; … S_n;
- To by odpovídalo rekurzívní funkci se strukturou:

  F(n) = { if (n > 1) F(n - 1); S_n; }

Rekurze a iterace

Příklad: cyklus s podmínkou na konci:

```
int i = N;
do {
 //sekvence příkazů Si
 i--;
} while (i > 0);
```

```
void tisk(int n)
{
  printf("%d ", n);
  if (n > 1) tisk(n-1);
  return;
}
//  n . . . 4 3 2 1
```

- Posloupnost bloků:
  S_n; S_{n-1}; … S_1;
- Odpovídající rekurzivní funkce:
  F(n) = { S_n; if (n > 1) F(n - 1); }

Rekurze a iterace

- Předchozí rekurzivní struktury nejsou proti iteraci žádným přínosem.
- Iterace je v těchto případech vždy efektivnější.
- Uvažujme nyní rekurzivní funkci F:

$F(k)=\{\ S_k;\ \text{if } (k>1)\ F(k-1);\ \text{else } S;\ T_k;\ \}$

S_k a T_k – posloupnosti příkazů závislé na k, ale proměnnou k nemění.

Rekurze a iterace

- Při volání pro k = n, n > 1, bude sled operací při výpočtu F(n) následující:

$$
\begin{aligned}
F(n) &\rightarrow S_n\ F(n-1)\\
&\rightarrow S_n\ S_{n-1}\ F(n-2)\\
&\rightarrow \ldots\\
&\rightarrow S_n\ S_{n-1} \ldots S_2\ S_1\ S\ T_1\\
&\rightarrow S_n\ S_{n-1} \ldots S_2\ S_1\ S\ T_1\ T_2\\
&\rightarrow \ldots\\
&\rightarrow S_n\ S_{n-1} \ldots S_2\ S_1\ S\ T_1\ T_2 \ldots T_n
\end{aligned}
$$

- Rekurzi lze chápat jako obecnější možnost řazení sekvencí příkazů než dovolují iterace.

Rekurze a zásobníková paměť

- Zásobník
  - Struktura do níž se prvky vkládají na konec (vrchol) a pak se od konce vybírají – pořadí prvků při vkládání a vybírání je opačné.
  - Stejně funguje zásobníková paměť při volání funkcí.
  - Vztah k rekurzi
    - Na vrchol zásobníku se ukládá stav dílčích podúloh (hodnoty mezivýsledků).
- Rekurzi z předchozího slajdu lze převést na iteraci pouze pomocí pomocného zásobníku – **nelze** ji nahradit pouhým zřetězením cyklů.

Rekurze a zásobníková paměť

- Při volání funkce se na vrcholu zásobníku vymezí oblast potřebná pro:
  - lokální proměnné,
  - argumenty, se kterými byla funkce volána,
  - výsledek funkce (jen pro funkce, které vracejí hodnotu),
  - informace pro správný návrat z funkce.
- Tato oblast se nazývá **aktivační záznam funkce**, který vzniká při každém volání funkce.

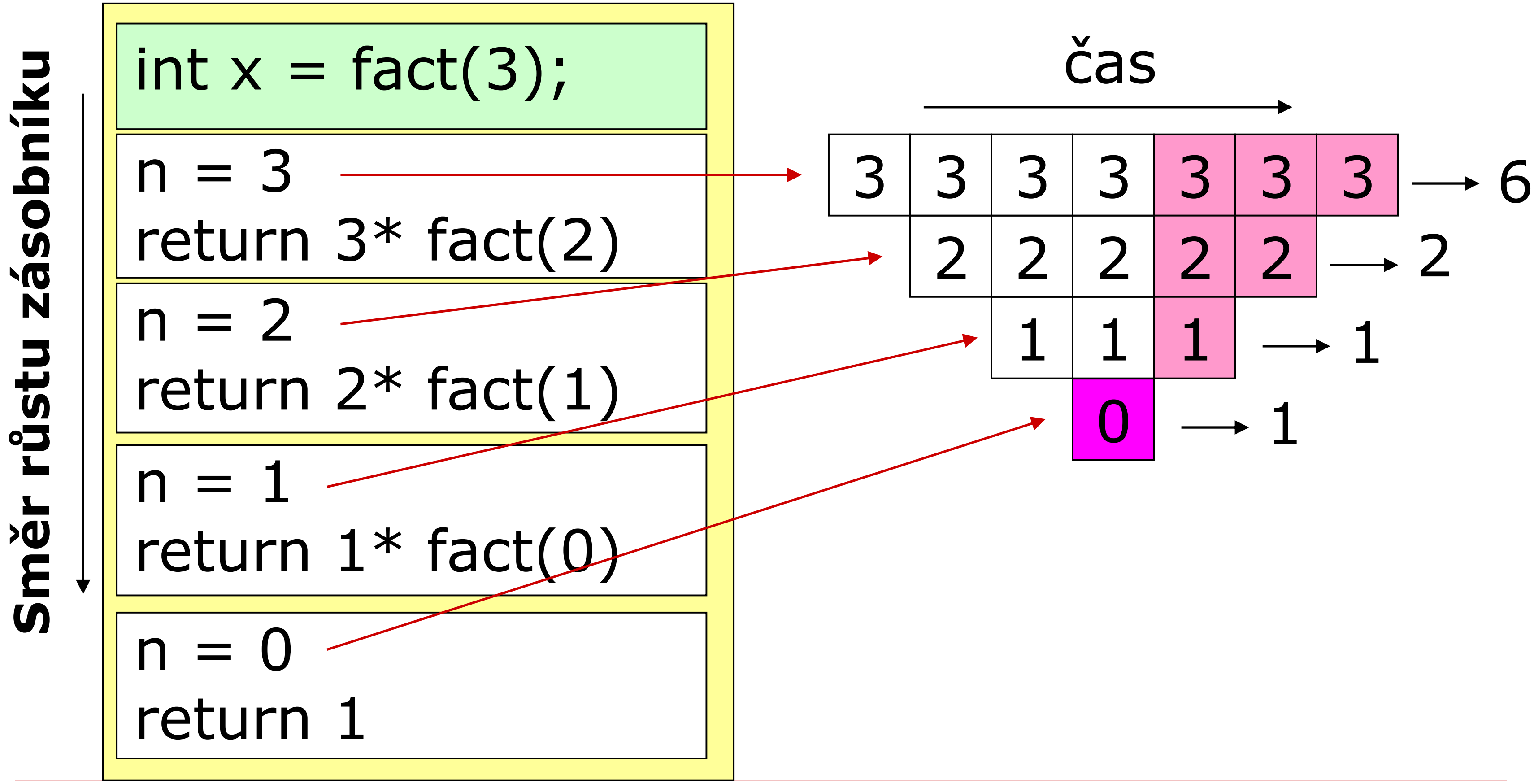
Rekurze v programování (slajd 5)

Příklad: rekurzivní funkce pro výpočet faktoriálu.

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

Rekurze a zásobníková paměť

Příklad:výpočet faktoriálu – demonstrace využití zásobníku při rekurzivním volání funkce.



Rekurzivní funkce

- Rekurzivní volání
  - Znovu táž posloupnost příkazů, aniž by původní byla dokončena
  - Vždy opět nová sada lokálních proměnných
- Podmínka vedoucí k ukončení funkce
  - Musí existovat, jinak vznikne nekonečná rekurze → havárie (vyčerpání zásobníku)
- **Přímá rekurze:** funkce A má ve svém těle opět příkaz volání funkce A.
- **Nepřímá (vzájemná, zprostředkovaná) rekurze** : A volá B; B volá C; C volá A.

Rekurzivní funkce

Příklad:Výpočet binomického koeficientu.

$$\binom{N}{K} = \frac{N!}{(N-K)!\,K!}$$

Nevhodné pro přímý výpočet (hodnoty faktoriálů mohou rychle přesáhnout počítačový rozsah).

Lze odvodit následující vztah:

$$\binom{N}{0} = 1$$

Splňuje všechny podmínky na rekurzivní definici.

$$\binom{N}{K} = \frac{N}{K}\binom{N-1}{K-1}$$

Kombinační číslo je matematická funkce, která udává počet kombinací, tzn. způsobů, jak vybrat k-prvkovou podmnožinu z n-prvkové množiny (k a n jsou čísla přirozená). Kombinační číslo se značí ve tvaru ($^n{}_k$) (čte se „n nad k”), někdy se používá také značení $_nC_k$ či C(n,k).
Kombinační číslo se používá hlavně v kombinatorice, velice důležité je využití v binomické větě (přičemž je zde označováno jako binomický koeficient) či Leibnizově pravidle.

Rekurzivní funkce

Příklad:Výpočet binomického koeficientu.

```
int binKoeficient (int n, int k)
{
  if (k == 0)
    return 1;
  else
    return binKoeficient(n - 1, k - 1)*n/k;
}
...
int n = 5, k = 3;
printf("kombinacni cislo %d nad %d = %d\n", n,k,
       binKoeficient(n,k));
```

Rekurzivní funkce

Příklad: Výpočet binomického koeficientu.

```
int binKoeficient (int n, int k)
{
  if (k == 0)
    return 1;
  else
    return binKoeficient(n-1,k-1)*n/k;
}
```

$$\binom{5}{3} = 10$$

	n	k	binKoeficient
1. volání	5	3	$6 \times 5 / 3 = 10$
2. volání	4	2	$3 \times 4 / 2 = 6$
3. volání	3	1	$1 \times 3 / 1 = 3$
4. volání	2	0	1

Rekurzivní funkce

Příklad: Celé kladné číslo *n* zapsat v soustavě o základu *z*.

```
void prevod (int n, int z)
{
  const char znakCisla[] =
        "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  if (n < z)
    printf("%c", znakCisla[n]);
  else
  {
    prevod(n/z, z);
    prevod(n%z, z);
  }
  return;
}
```

$10 = (1010)_2$

	n	**z**
1. volání	10	2
2. volání	5	2
3. volání	2	2
4. volání	1	2

0
1
0
1

Rekurzivní funkce

Hanojské věže

Mezi klasické problémy použití rekurze patří problém Hanojských věží. Stará vietnamská hra, k níž se váže legenda: Při stvoření světa bylo na jednu ze tří diamantových jehel Velkého chrámu v Benaresu umístěno 64 disků. Od té doby kněží tyto disky přerovnávají na cílovou jehlu, přičemž:

- v daném okamžiku se přenáší jen jeden disk,
- nikdy nesmí být umístěn větší disk na menším.

Až se kněžím podaří disky přenést, chrám se zboří a svět s třeskem zmizí. Protože disků je 64 dává to naději, že to tak hned nebude.

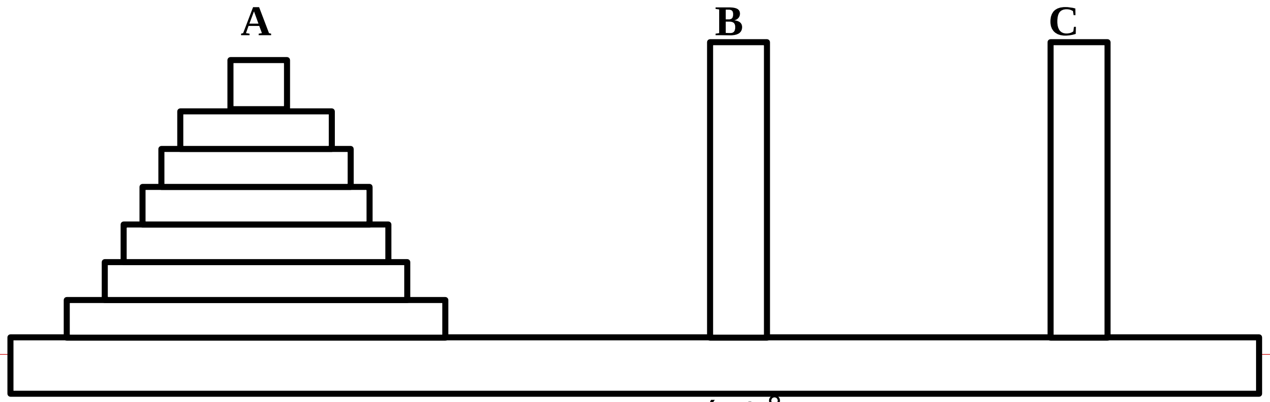
Máme k dispozici tři jehly (A,B,C) a určitý počet disků s různým průměrem a s otvorem uprostřed, které se mohou na jehly nasazovat.

Rekurzivní funkce

Na počátku jsou všechny disky navlečeny na jedné z jehel (A) tak, že tvoří věž, směrem vzhůru se zužující.
Úkolem je přemístit disky na jehlu (C) s použitím jehly (B), přičemž musíme dodržovat tato pravidla:

- v každém kroku můžeme přemístit pouze jeden disk, a to vždy z jehly na jehlu (disky nelze odkládat mimo jehly),
- není povoleno položit větší disk na menší.

Pro tuto úlohu existuje i nerekurzívní postup řešení.
Ale rekurzívní postup, založený na rozkladu úlohy na úlohy menší (s nižšími hodnotami parametrů) je zde velmi názorný a jednoduchý.



Rekurzivní funkce

Hanojské věže (http://www.animatedrecursion.com/intermediate/towersofhanoi.html)

[on line, cit. 2018-11-12]

- rekurzívní řešení založíme na tom, že problém s N disky bude řešen, když:
  - přeneseme N - 1 horních disků z A na B s použitím C jako odkládací jehly,
  - pak přeneseme největší disk zbylý na A, na jehlu C,
  - a konečně přerovnáme věž s N - 1 disky z B na C s použitím A jako odkládací jehly.

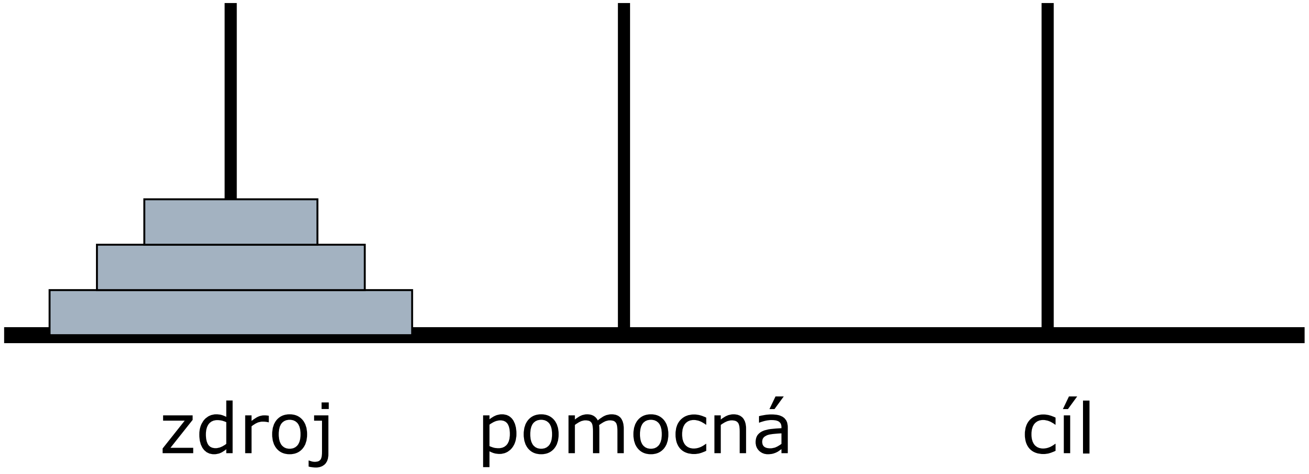
Rekurzivní funkce

- Tento postup zachycuje neformální rekurzívní specifikace úlohy HANOI (A,B,C,N) - přenesení N disků z A na C s pomocí B pro odkládání:

```
HANOI (A : odkud, B : odkládací jehla,
          C : kam, N : počet disků)
pro N = 1 TAH( z A na C)
jinak
( HANOI (A, C, B, N - 1);
   TAH (z A na C);
   HANOI(B, A, C, N - 1))
```

Rekurzivní funkce

Přenos 3 disků z tyče zdroj na tyč cíl.



Rekurzivní funkce

- počet tahů:

jestliže T_N je počet tahů pro N disků, pak platí:

$T_1 = 1$ a $T_N = 2T_{N-1} + 1$, pro $N > 1$,

odtud $T_N = 2^N - 1$.

Výchozí stav: 3 disky na věži 1.

1. Tah z 1 na 3
2. Tah z 1 na 2
3. Tah z 3 na 2
4. Tah z 1 na 3
5. Tah z 2 na 1
6. Tah z 2 na 3
7. Tah z 1 na 3

Konečný stav: 3 disky na věži 3.

Rekurzivní funkce

- Rekurze je neefektivní
  - Každé volání funkce má režii – alokace paměti na zásobníku, inicializace aktivačního záznamu – prodlužuje se doba výpočtu
  - Pokud to jde, používáme iteraci

Příklad: Funkce pro výpočet n-tého členu Fibonacciho posloupnosti, která je rekurentně definovaná takto:

$$f_n = \begin{cases} 0 & \text{pro } n = 0 \\ 1 & \text{pro } n = 1 \\ f_{n-1} + f_{n-2} & \text{pro } n > 1 \end{cases}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, …

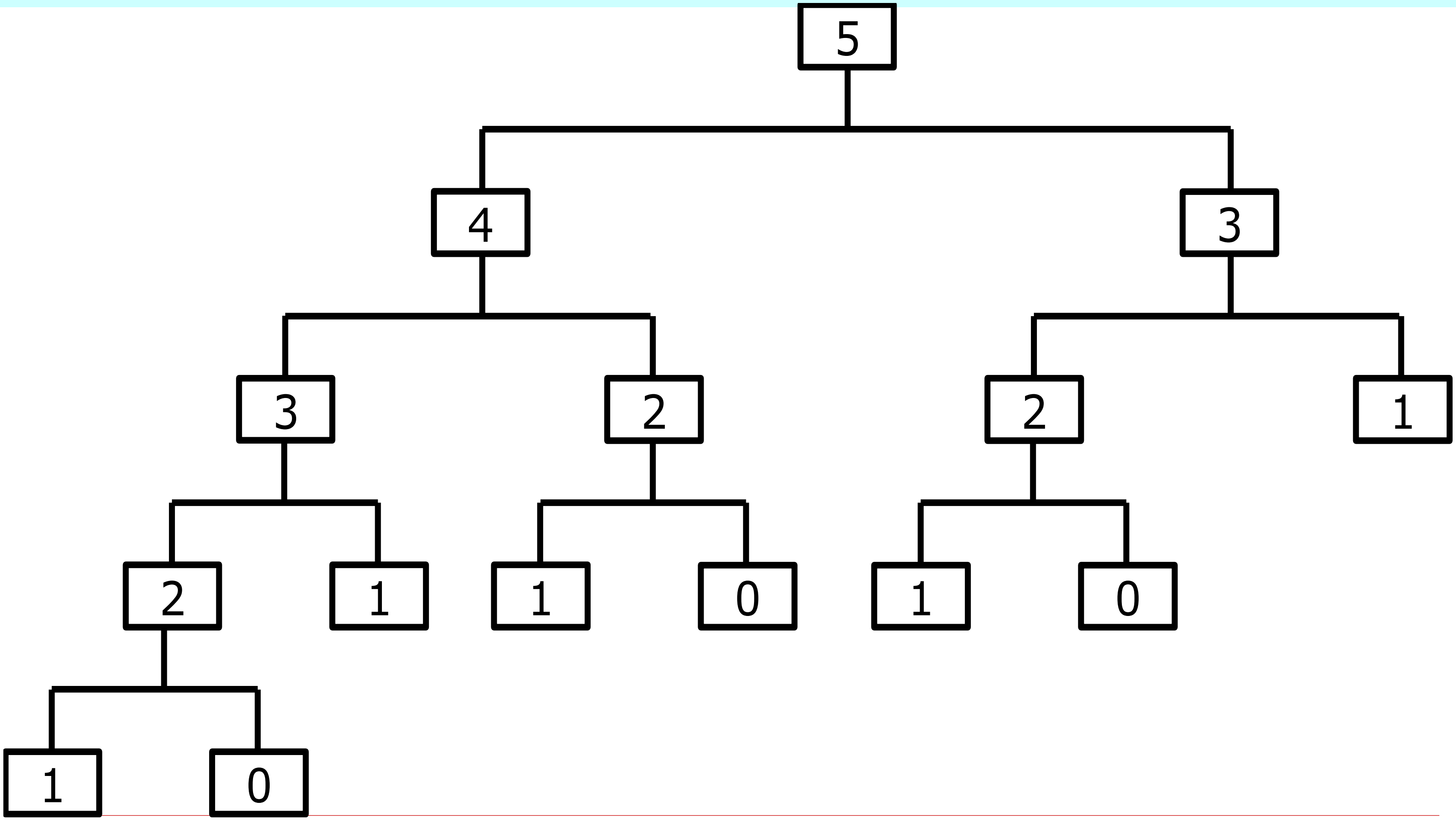
Rekurzivní funkce

Příklad: výpočet n-tého členu Fibonacciho posloupnosti.

```
#define NFIBOS 36
long int fibo (int n)
{
  if ((n == 0) || (n == 1)) return n;
  else return fibo(n-1) + fibo(n-2);
}
int main(void)
{
  for (int i = 0; i < NFIBOS; i++)
    printf("Fib. číslo #%d = %ld\n",i,fibo(i));
  return 0;
}
```

Rekurzivní funkce

Příklad: patnáct vyvolání funkce *fibo* pro n = 5.



Rekurzivní funkce

- Funkce fibo() je nepoužitelná – v cyklu od 0 do 5 dojde k 34 voláním funkce, každá další iterace tento počet zhruba zdvojnásobí
- Lepší řešení: nerekurzivně pomocí cyklu

```
long fibo[NFIBOS];
int i;
fibo[0] = 0; fibo[1] = 1;
for (i = 2; i < NFIBOS; i++)
  fibo[i] = fibo[i-2] + fibo[i-1];
for (i = 0; i < NFIBOS; i++)
  printf("Fib. číslo #%d = %ld\n",i+1,fibo[i]);
```

Rekurzivní funkce

Posloupnost lze jednoznačně rozšířit na obor celých záporných čísel, kdy první členy od Fib(0) dále k menším číslům jsou:

0, 1, -1, 2, -3, 5, -8, 13, -21, ...

a platí :

Fib(n) = - Fib(n) pro n sudé
Fib(n) =   Fib(n) pro n liché

[Fibonacciho králíkárna](http://tomason.free.fr/kvazi/Kvazi.html) [on line, cit. 2018-11-16]
http://tomason.free.fr/kvazi/Kvazi.html

Rekurzivní funkce

Dělitelnost Fibonacciho čísel
Každá dvě po sobě jdoucí čísla Fibonacciho posloupnosti jsou nesoudělná. Nechť n-tý člen Fibonacciho posloupnosti je její nejmenší kladný člen, dělitelný přirozeným číslem d. Pak všechna čísla Fibonacciho posloupnosti dělitelná číslem n jsou čísla Fib(k*n), kde k je celé číslo, a žádná jiná.

Příklad: pro d=7 je nejmenší kladný člen dělitelný číslem 7 číslo Fib(8) = 21, proto všechna čísla Fibonacciho posloupnosti dělitelná číslem 7 jsou : Fib(0)=0, Fib(8)=21, Fib(16)=987, Fib(24)=46368,... a žádná jiná.

Nechť číslo Fib(n) je prvočíslo různé od 3. Pak n je prvočíslo.
Příklad: 13 je prvočíslo (různé od 3) a 13=Fib(7), proto 7 je prvočíslo.
ALE :
Neplatí, že je-li p prvočíslo, pak Fib(p) je prvočíslem. Nejmenším takovým prvočíslem je p=19, protože Fib(19) = 4181 = 37*113.

Rekurzivní funkce

- Nepřímá rekurze
  - Vzájemné rekurzivní volání více funkcí.
  - Implementační problém: potřebujeme používat funkce, které ještě nebyly implementovány
  - V jazyce C pomocí deklarace prototypu funkce

Příklad: výpis prvků seznamu (samozřejmě to jde i jinak).

```
// prototyp funkce
void vypisSeznam(const TList *list);
```

Rekurzivní funkce

```
// vypíše první prvek seznamu a pak i ostatní
void prvniAZbytek(const TList *list)
{
  vypisData(list->data);
  vypisSeznam(list->dalsi);
}
// definice funkce
void vypisSeznam(const TList *list)
{
  if (list != NULL)
  { // koncová podmínka
    prvniAZbytek(list);
  }
}
```

Rekurzivní funkce

- Výpočet největšího společného dělitele.
  Euklidův algoritmus pro kladná čísla
  NSD(a,b) (a,b > 0; a > b)

1. číslo a dělíme číslem b $\rightarrow$ r_1
2. je-li $r_1 \neq 0$, dělíme číslo b číslem $r_1 \rightarrow r_2$
3. je-li $r_2 \neq 0$, dělíme číslo r_1 číslem $r_2 \rightarrow r_3$
4. .
5. .
6. po n krocích dojdeme ke zbytku:
   $r_n = 0, \rightarrow \text{NSD}(a,b) = r_{n-1}$

Rekurzivní funkce

Příklad: největší společný dělitel čísel 648 a 252.

648	252	2
144		1
108		1
36		3
0		

podíly

zbytky po dělení

r_{n-1}

Rekurzivní funkce

Příklad: největší společný dělitel.

```
long long spolecny_delitel(unsigned long i,
                           unsigned long j)
{
  if(i==0 || j==0)
  { // některé z čísel je nula
    return -1;
  }
  if(i<j)
  { unsigned long k = i; i = j; j = k; }
  return (i%j > 0)? spolecny_delitel(j,i%j) : j;
}
```

Rekurzivní funkce

Příklad: největší společný dělitel.

```
int main(void)
{
  unsigned long int i,j;
  printf("Nejvetsi spolecny delitel dvou celych,"
         " kladnych, nenulovych cisel.\n");
  printf("Zadej dve cela kladna cisla:  ");
  scanf("%lu%lu",&i,&j); //!!
  printf("Nejvetsi spolecny delitel cisel"
         " %lu a %lu je cislo %lu",
         i, j, spolecny_delitel(i, j));
}
```

Rekurze v programování

Kontrolní otázky

1. V čem se liší rekurze a iterace?
2. Proč bývají iterační algoritmy bezpečnější než rekurzívní?
3. V čem tkví nebezpečí při používání rekurzívních algoritmů?
4. Jaký je rozdíl mezi přímou a nepřímou rekurzí?

Úkoly k procvičení

1. Definujte rekurzívní funkci v jazyce C pro výpočet faktoriálu.
2. Napište funkci pro rekurzivní výpočet hodnoty v Pascalově trojúhelníku: pascal(n, k), kde n je index řádku Pascalova trojúhelníku a k je index prvku na tomto řádku. Pascalův trojúhelník vypadá takto:

   1
   1 1
   1 2 1
   1 3 3 1
   1 4 6 4 1
   1 5 10 10 5 1
   .....

3. Napište program v jazyce C pro řešení problému Hanojských věží.
4. Napište program v jazyce C (bez použití rekurze) pro výpočet n-tého členu Fibonacciho posloupnosti.
5. Napište program v jazyce C (bez použití rekurze) pro výpočet největšího společného dělitele dvou celých kladných čísel.