Rekurze (1)

- Schopnost objektů definovat se pomocí sebe sama
- Říkáme, že funkce nebo procedura je rekurzivní (je definována rekurzivně), jestliže se použití její definice vyskytuje uvnitř definice samotné

Rekurze (2)

- Základní typy rekurze:
  - přímá: funkce nebo procedura volá sama sebe
  - nepřímá: funkce nebo procedura volá jinou proceduru nebo funkci, která potom volá opět proceduru nebo funkci výchozí
  - lineární: funkce nebo procedura se v jednom průchodu vyvolá jen jednou
  - stromová: funkce nebo procedura se v jednom průchodu vyvolá vícekrát

Rekurze (3)

- Výpočet faktoriálu čísla n:
  - lze využít vztahu:
    0! = 1
    n! = n.(n-1).(n-2). ... .1 = n.(n-1)! pro n > 0
  - jedná se o nepříliš vhodné použití rekurze, protože zápis algoritmu pomocí rekurze nebude jednodušší než zápis bez ní a navíc výsledný program bude pomalejší (způsobeno režií při vyvolávání funkce nebo procedury)

Rekurze (4)

- Výpočet NSD čísel m, n:
  - NSD (m,0) = m
  - NSD (m,n) = NSD (n, m mod n) pro n > 0
  - Tento postup redukuje problém nalezení NSD (m, n) na problém nalezení NSD (n, m mod n), kde $0 \leq m \bmod n < n$
  - Tento proces musí po konečném počtu kroků vést k NSD (a, b), kde b = 0
  - vykazuje stejný problém (je pomalejší) jako rekurzivní výpočet faktoriálu

Rekurze (5)

- Při použití rekurze je nutné vždy dávat pozor, aby počet rekurzivních vyvolání podprogramu (tzv. hloubka rekurze) nebyl příliš vysoký, nebo dokonce nekonečný
- Je tedy nutné, aby v definici rekurzivní procedury (funkce) byla vždy správně uvedena ukončovací podmínka

Rekurze (6)

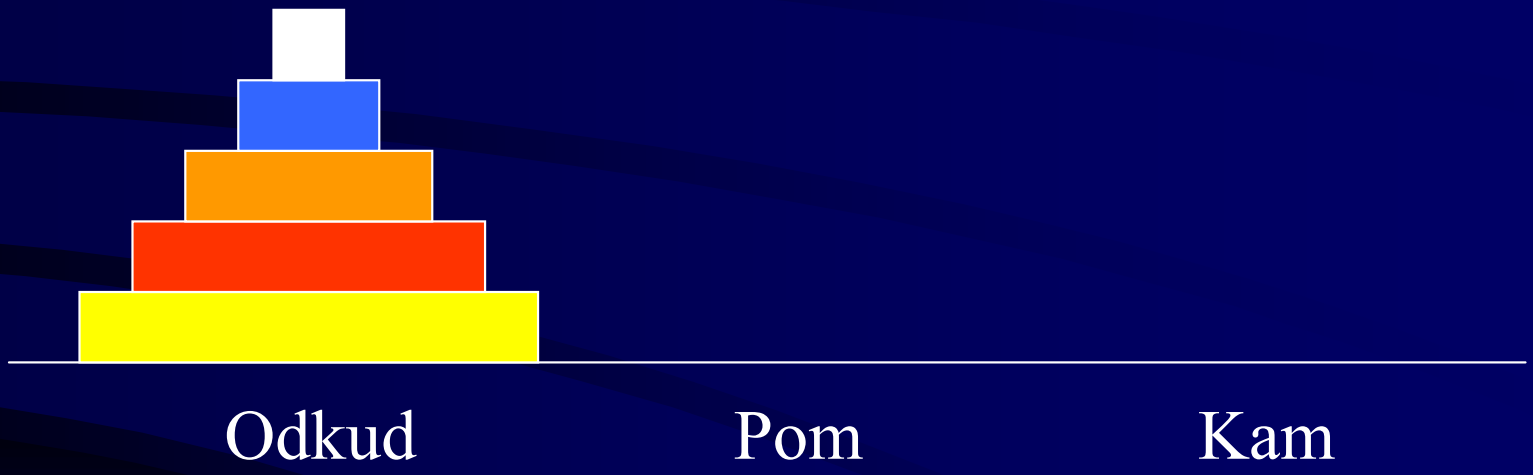
- Fibonacciho posloupnost:
  - $fib_1 = fib_2 = 1$
  - $fib_{n+2} = fib_{n+1} + fib_n$ pro $n > 0$
  - použití rekurze je naprosto nevhodné protože pro $n > 1$ počet volání roste exponenciálně

Rekurze (7)

- Pravidlo:
  - je-li možné jednoduše použít iterace (zápisu pomocí cyklu), pak se rekurzi vyhneme
  - rekurzi použijeme v případě, že daný problém je definován rekurzivně
- Platí:
  - každý iterativní algoritmus lze napsat rekurzivně a naopak

Rekurze (8)

- Problém Hanoiská věž:



- Tento problém lze vhodně řešit pomocí rekurze (stromové)

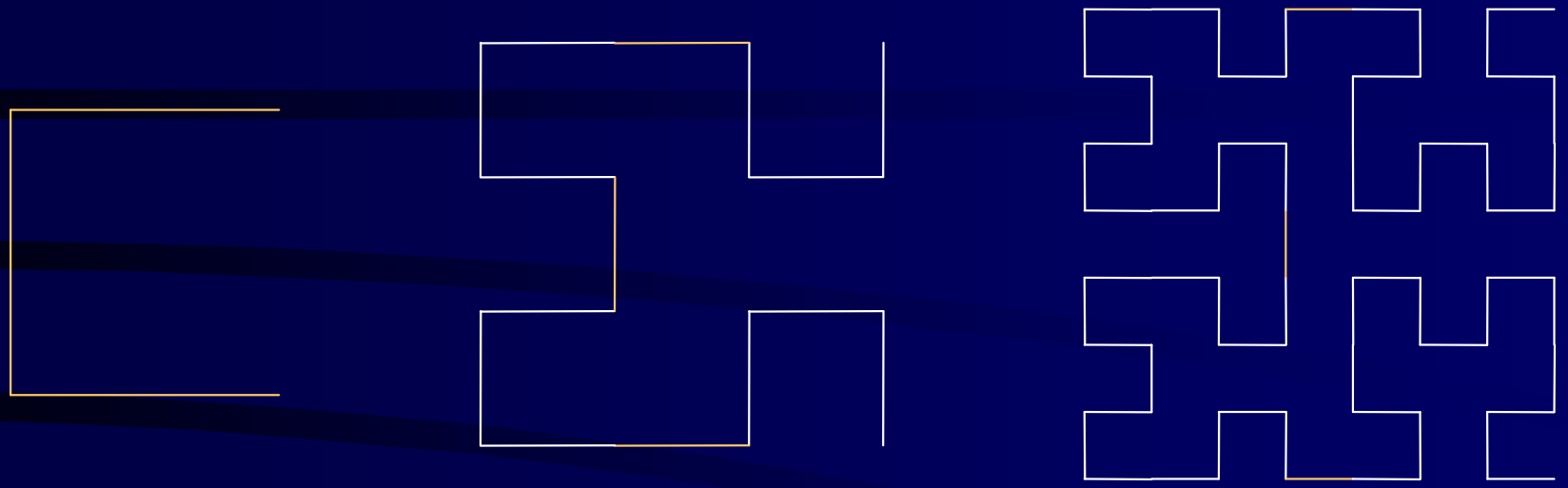
Rekurze (9)

- Při použití nepřímé rekurze je nutné, aby jedna z procedur (funkcí) byla volána dříve než je uvedena její deklarace
- V takovém případě musíme ještě dříve než použijme vyvolání dosud nedeklarované procedury (funkce) provést zápis deklarace její hlavičky s direktivou forward

Rekurze (10)

```
procedure Q (a:real; var b:integer); forward;
procedure P (x:intger);
begin
    Q (10.2, i);
end;
procedure Q;
begin
    P (8);
end;
```

Hilbertovy křivky (1)



H_1 H_2 H_3

- H_i - Hilbertova křivka i-tého řádu
- H_{i+1} dostaneme kompozicí čtyř křivek H_i poloviční velikosti, jejich pootočením a spojením třemi přímkami

Hilbertovy křivky (2)

- Označme čtyři části křivky H_i (Hilbertovy křivky nižšího řádu H_{i-1}) symboly A, B, C, D a spojující přímky šipkou odpovídajícího směru
- Dostáváme rekurzivní schéma:

⊏ A: D ← A ↑ A → B

⊔ B: C ↓ B → B ↑ A

⊐ C: B → C ↓ C ← D

⊓ D: A ↑ D ← D ↓ C

Typ množina (set) (1)

- Definice typu množina je tvaru:

  type T = set of T_z;

  kde T: jméno nově definovaného typu množina

  T_z: označení ordinálního typu (identifikátor nebo popis typu), kterému se říká bázový typ

- Hodnotami proměnných typu T jsou všechny podmnožiny bázového typu T_z a prázdná množina

Typ množina (set) (2)

- Např.:

```
type zakladnibarva = (cervena, zelena, modra);
     barva = set of zakladnibarva;
```

  – Hodnotami proměnných typu barva jsou:
  [cervena, zelena, modra], [cervena, zelena], [cervena, modra], [zelena, modra], [cervena], [zelena], [modra], []

Typ množina (set) (3)

- Konstrukce hodnoty množina spočívá ve výčtu množinových elementů, které jsou odděleny čárkami a uzavřeny do hranatých závorek
- Elementy mohou být dány výrazem bázového typu nebo ve tvaru m..n, který představuje množinu všech prvků i bázového typu takových, že $m \leq i \leq n$

Typ množina (set) (4)

- [] je prázdná množina stejně jako [m..n], kde m > n.
- Příklady:

  [6], [i+j, i-j], [’0’..’9’], [’a’, ’b’, ’c’]
- Operátory aplikovatelné na operandy typu množina s kompatibilními bázovými typy:

  + sjednocení

Typ množina (set) (5)

*	průnik
-	množinový rozdíl
=, < >	test na rovnost (nerovnost)
<=, >=, <, >	test na, množinovou inkluzi
in	test na příslušnost k množině

- Příklad:

```
if (ch = 'a') or (ch = 'e') or (ch = 'i') or (ch = 'o')
   or (ch = 'u') or (ch = 'y') then P;
```

Typ množina (set) (6)

lze nahradit:

if ch in [’a’, ’e’, ’i’, ’o’, ’u’, ’y’] then P;

- Implementace Pascalu obvykle omezují rozsah bázového typu množina v závislosti na délce posloupností z 0 a 1, kterými jsou reprezentovány množiny
- Díky této reprezentaci však mohou probíhat množinové operace velmi rychle

Typ množina (set) (7)

- Omezení Borland (Turbo) Pascalu:
  - bázový typ nesmí mít více než 256 hodnot
  - ordinální hodnoty dolní a horní meze bázového typu musí ležet uvnitř intervalu 0..255
- Poznámka:
  Množina je statický a homogenní datový typ

Typ záznam (record) (1)

- Hodnoty typu záznam jsou tvořeny složkami různých typů $\Rightarrow$ záznam je heterogenní datový typ
- Tyto složky (jejich počet a typ) jsou pevně dány definicí typu záznam, popř. deklarací proměnné typu záznam $\Rightarrow$ záznam je statický datový typ

Typ záznam (record) (2)

- Typ záznam definujeme takto:

type T = record

$p_{11}, p_{12}, \ldots, p_{1n_1}: T_1;$

$p_{21}, p_{22}, \ldots, p_{2n_2}: T_2;$

$p_{m1}, p_{m2}, \ldots, p_{mn_m}: T_m;$

end;

Typ záznam (record) (3)

- p_{ij} značí identifikátory složek, kterým se říká položky
- Položky záznamu mohou být dále strukturovaného typu (např. pole, množina nebo opět záznam)

Typ záznam (record) (4)

- Příklad:

  na základě definic typů:

  type den = 1..31;

  mesic = (leden, unor, brezen, duben,
  kveten, cerven, cervenec,
  srpen, zari, rijen, listopad,
  prosinec);

  rok = 1900 .. 2100;

Typ záznam (record) (5)

můžeme definovat typ:

```
datum = record
            d: den;
            m: mesic;
            r: rok;
        end;
```

Typ záznam (record) (6)

- Výběr položky (přístup k položce) záznamu se realizuje zápisem, který je tvořen identifikátorem záznamu následovaným tečkou a identifikátorem příslušné položky
- Je-li proměnná deklarována jako:

  var prijezd:datum;

  pak se její složky označují:

  prijezd.d, prijezd.m, prijezd.r

Typ záznam (record) (7)

- Změna hodnoty proměnné se realizuje změnou jednotlivých položek:

  Např.:

```
if prijezd.m < > prosinec
   then prijezd.m := succ (prijezd.m)
   else begin prijezd.m := leden;
                  prijezd.r := prijezd.r+1;
        end;
```

Typ záznam (record) (8)

- Typ záznam může mít i tzv. variantní část:

```
type typvoz = (autobus, naklauto);
     Vozidlo = record
          cena:longint;
          case TV:typvoz of
               autobus: (pocosob:integer);
               naklauto: (nosnost: integer);
          end;
```

Příkaz with (1)

- Slouží ke zjednodušení přístupu k položkám proměnných typu záznam
- Obecný tvar:

  with $p_1, p_2, ..., p_n$ do P;

  kde p_i: proměnná typu záznam

  P: příkaz, v jehož rámci je možné pracovat přímo s položkou záznamu, aniž bychom museli uvádět proměnnou typu záznam následovanou tečkou

Příkaz with (2)

- Příklad:

```
with prijezd do
    if m < > prosinec then m := succ (m)
      else begin m := leden;
                 r := r+1;
           end;
```