Zpracování deklarací a přidělování paměti

- **Účel deklarací**
  -pojmenování objektů
  -umístění objektů v paměti

- **Tabulka symbolů**
  -uchovává informace o objektech
  -umožňuje kontextové kontroly
  -umožňuje operace
    1. **inicializaci informace pro standardní jména**
    2. **vyhledání jména**
    3. **doplnění informace ke jménu**
    4. **přidání položky pro nové jméno**
    5. **vypuštění položky či skupiny položek**

Struktura tabulky symbolů

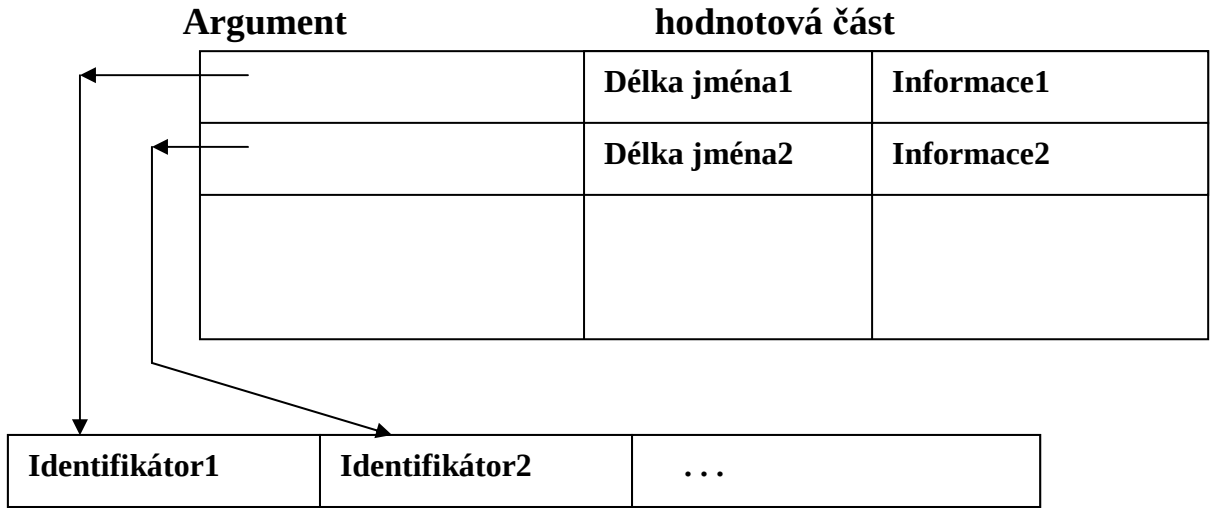
- **s jednoduchou strukturou**
- **s oddělenou tabulkou identifikátorů**
- **s oddělenou tabulkou informací**
- **uspořádané do podoby zásobníku**
- **s blokovou strukturou**

Tabulka symbolů:

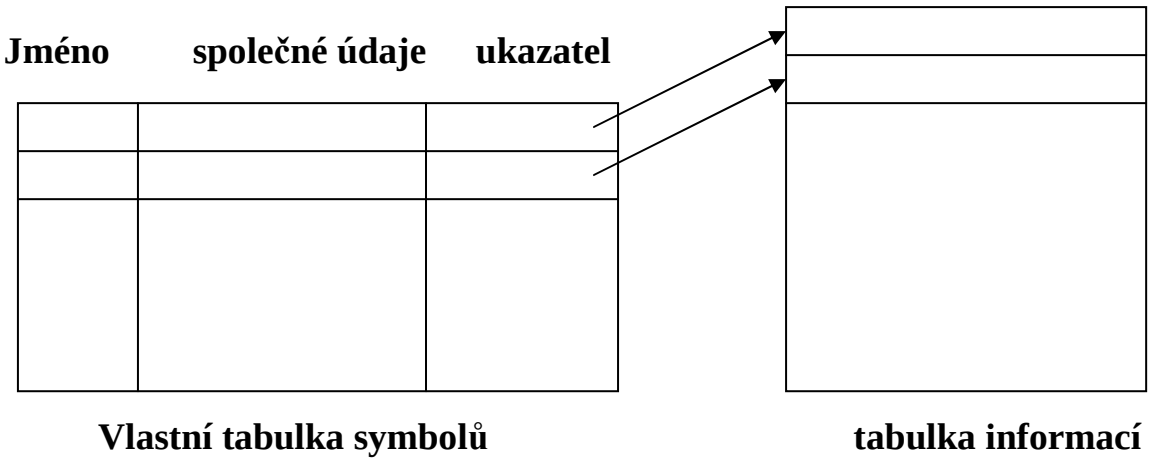
Argument= jméno hodnotová část= atributy

1.položka
2.položka
.
.
.
n-tá položka

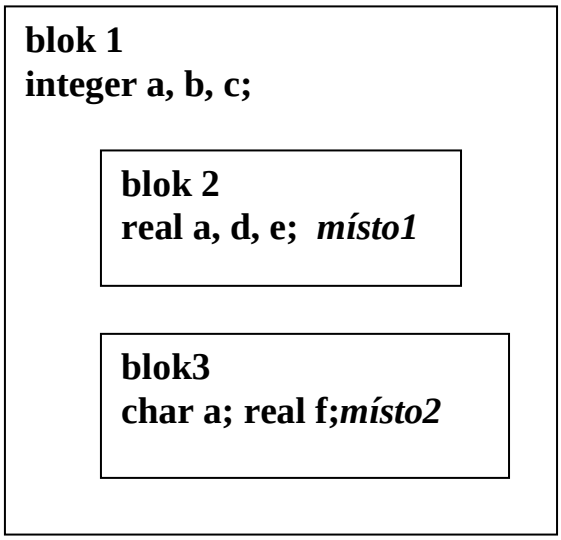
Tabulka symbolů s oddělenou tabulkou identifikátorů (neomezená jména)



Tabulka symbolů s oddělenou tabulkou informací



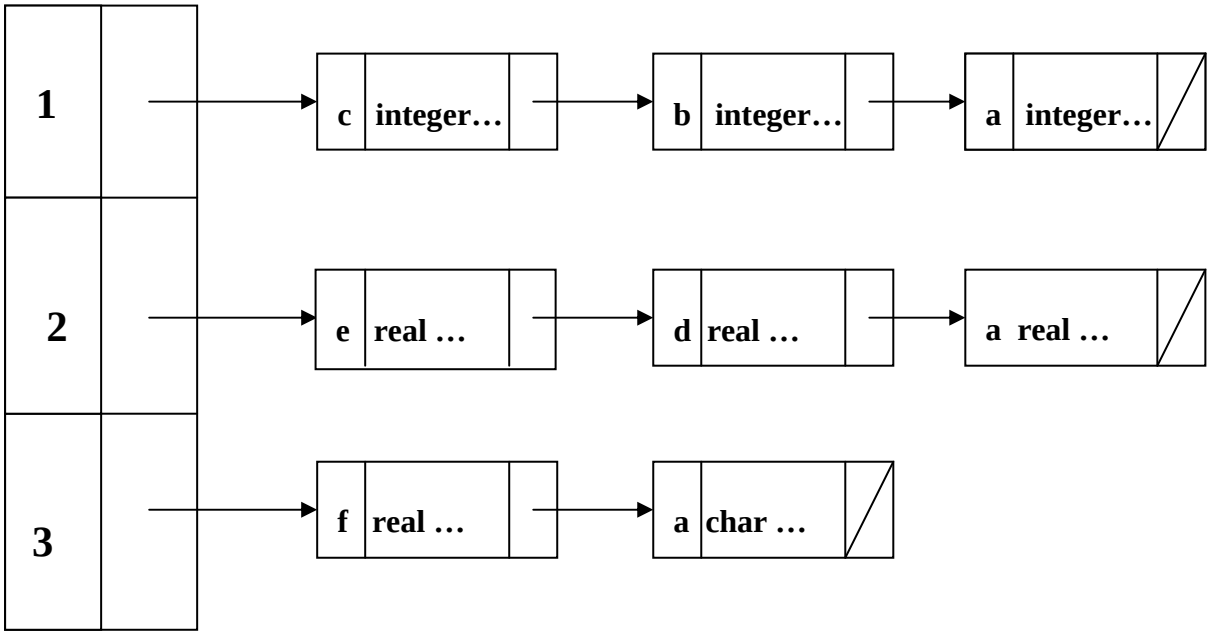
Tabulka symbolů uspořádaná do podoby zásobníku (pro jazyky s blokovou strukturou).
Rozsahová jednotka je blok, modul, funkce, balík,…
Respektuje zásady lokality

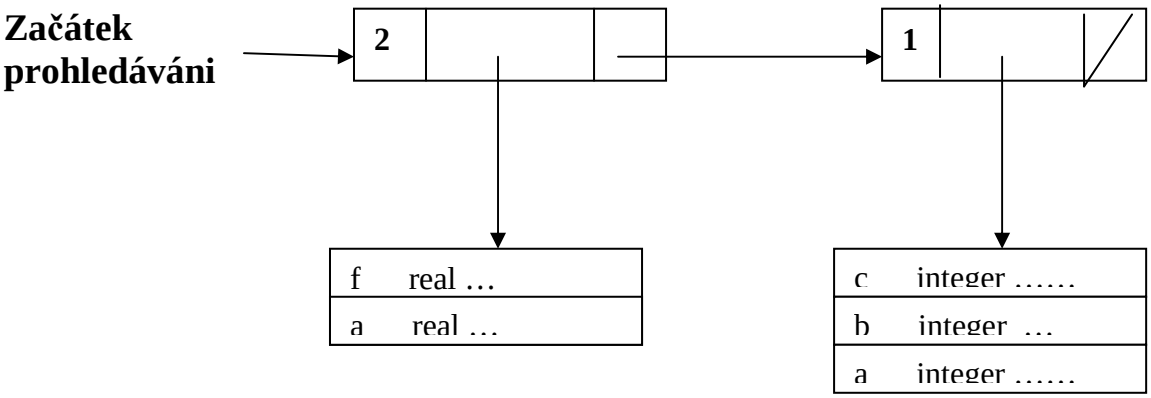


Vrchol →	**e**	**real, …**	
	d	**real, …**	
	a	**real, …**	**směr prohledávání**
	c	**integer, …**	
	b	**integer, …**	
	a	**integer, …**	

Vrchol →	**f**	**real, …**	
	a	**character, …**	
	c	**integer, …**	**směr plnění**
	b	**integer, …**	
	a	**integer, …**	

Tabulka symbolů s blokovou strukturou





Informace v tabulce symbolů
(závisí na jazyce i způsobu překladu)

Př. pro jazyk s blokovou strukturou

1. **DRUH**
   - **-návěští**
   - **-konstanta**
   - **-typ i příp. objektový**
   - **-proměnná**
   - **-procedura**
   - **-funkce**
   - **-metoda**

2. **Hladina popisu = úroveň vnoření**

3. **Adresa**
   - **-funkcí, procedur, metod**
   - **-proměnných (statická nebo offset)**
   - **-konstant**

4. **Použití byla/nebyla použita**

5. **Typ**
   - **-údaj o standardním jednoduchém typu**
   - **-údaj o typu definovaném uživatelem**
   - **-údaj o strukturovaném typu**
   - **-údaj o objektovém typu**

6. **Formální parametr je/není to formální parametr**

7. **Druhy formálních parametrů kolik jich má a kde jsou v TS**

8. **Způsob volání hodnotou/odkazem**

9. **Hodnota jen u konstant**

Implementace tabulky symbolů

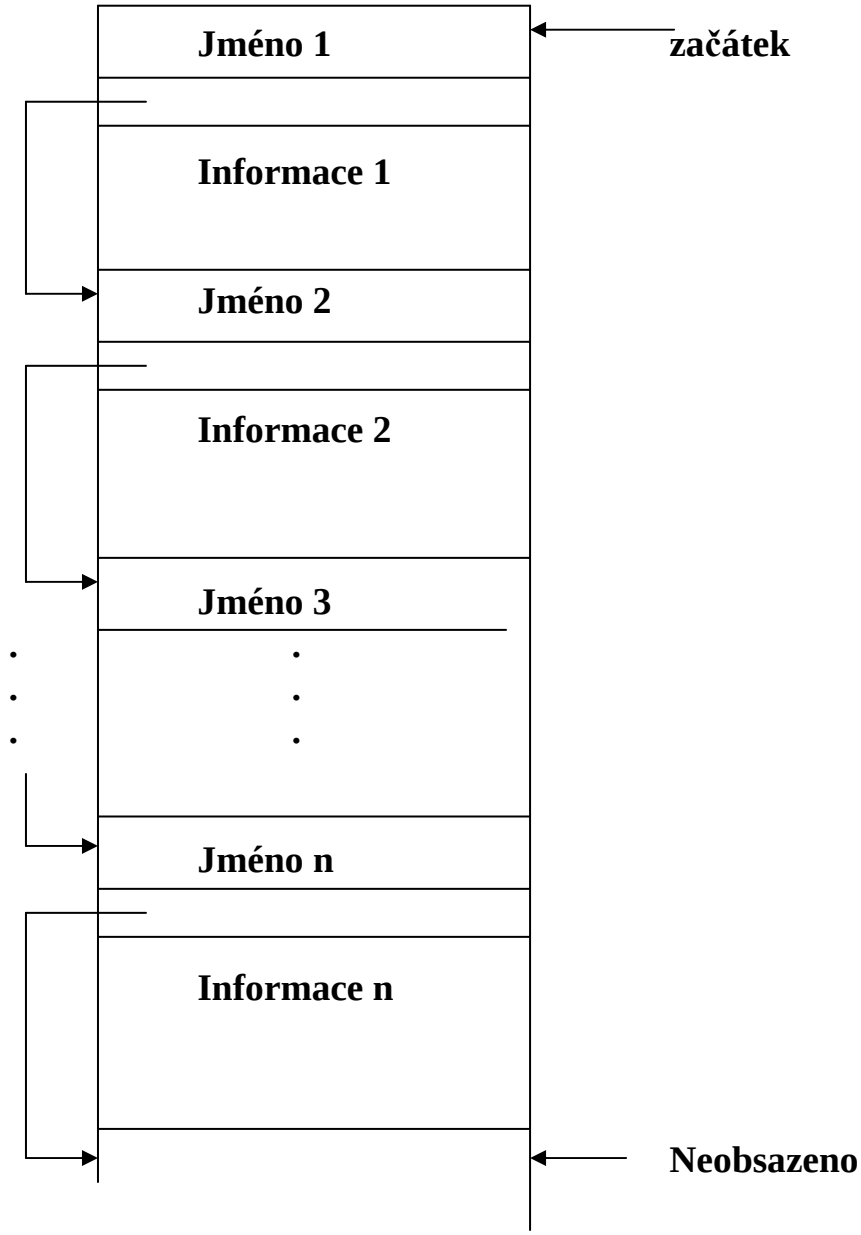
- **Vyhledávací netříděné tabulky (jen pro krátké programy)**
  -prostá struktura
  -lineární seznam

- **Vyhledávací setříděné tabulky**
  -průběžné setřiďování
  -setřídění po zaplnění

- **Frekvenčně uspořádané tabulky**

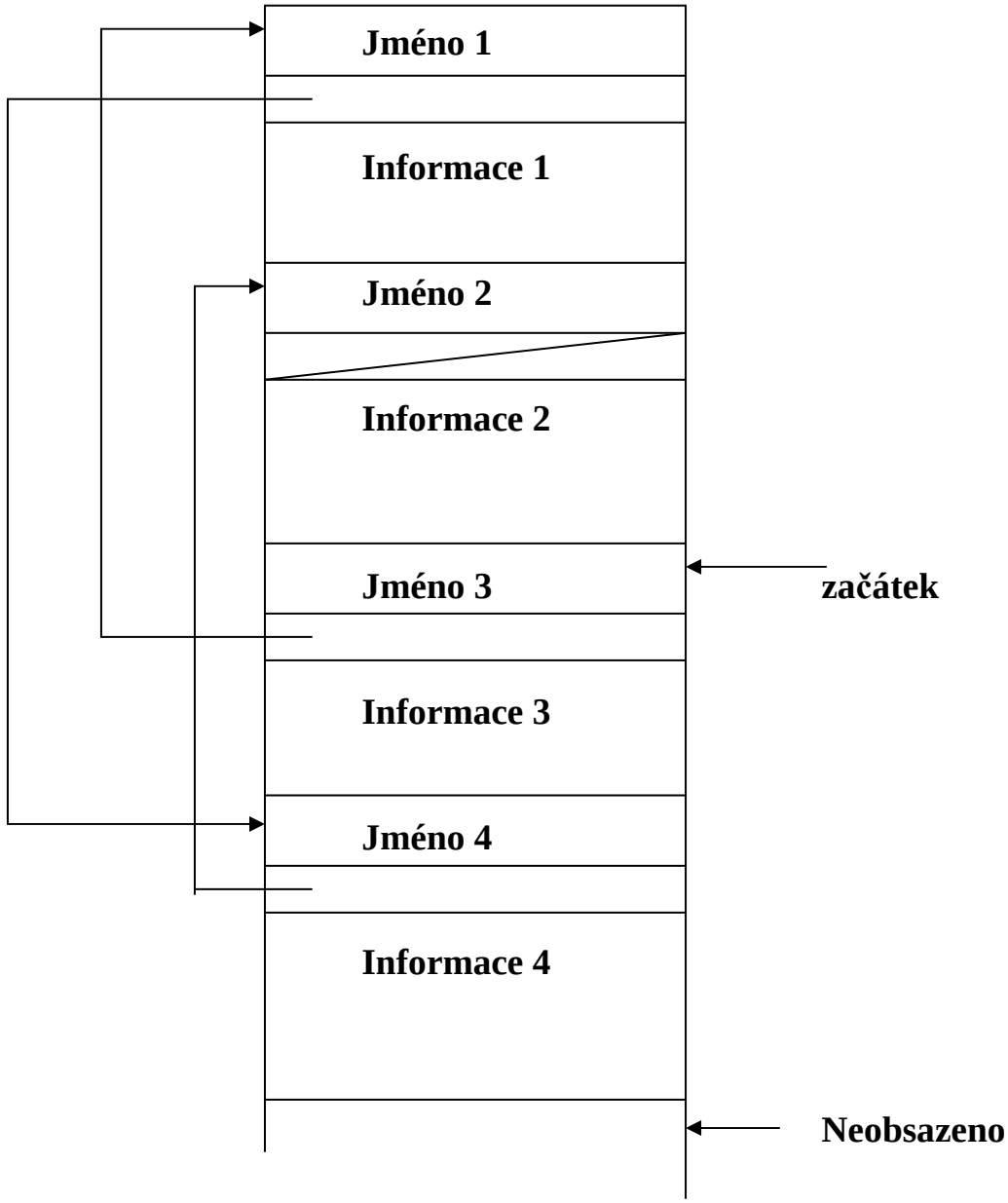
- **Binární vyhledávací stromy**

- **Tabulky s rozptýlenými položkami**



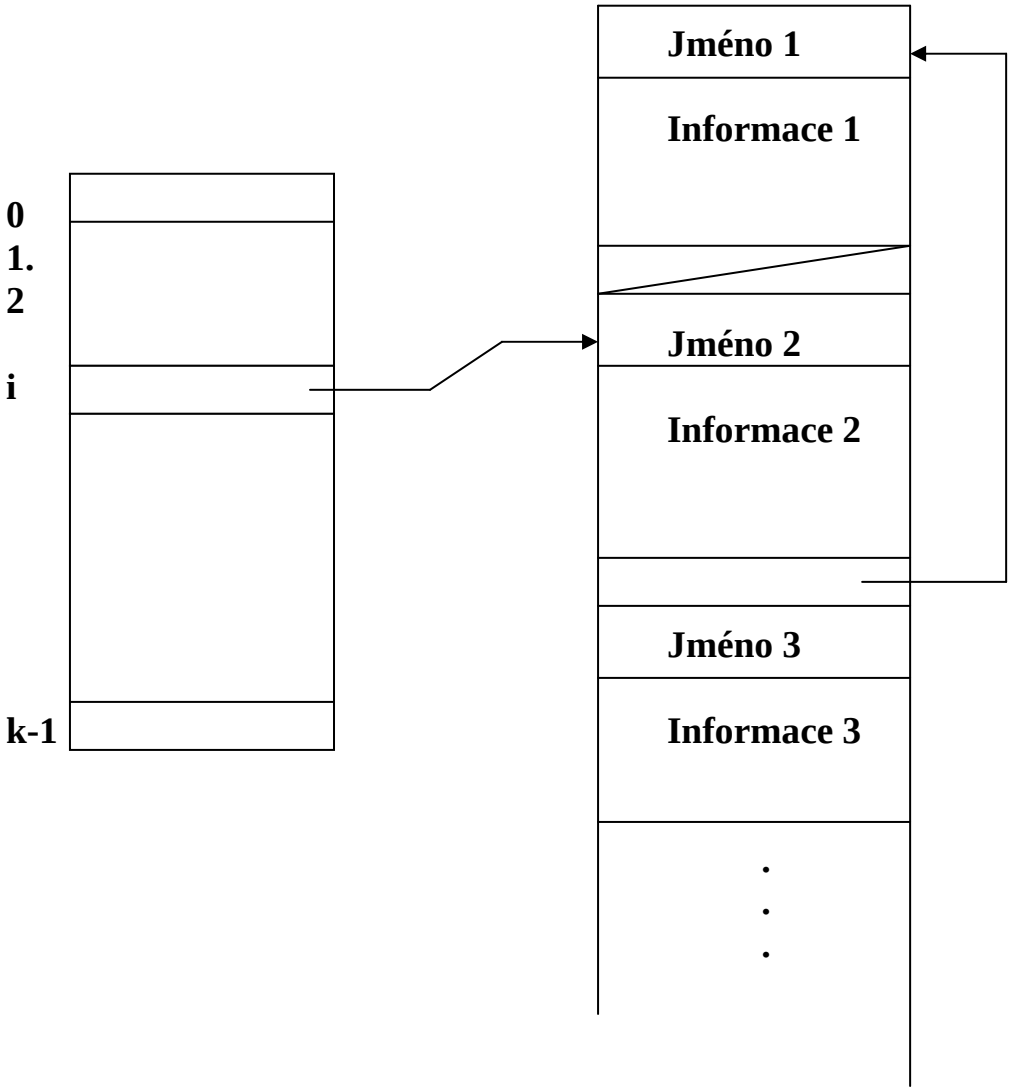
Tabulka symbolů organizovaná jako lineární seznam

? Jaká je časová náročnost v závislosti na počtu položek?



Frekvenčně uspořádaná tabulka symbolů

? Jaká je časová náročnost v závislosti na počtu položek?



Rozptýlená organizace tabulky symbolů

-Rozptylovací funkce h(jméno)=zbytek po dělení (∑ord(znak))/k

-Řešení kolizí

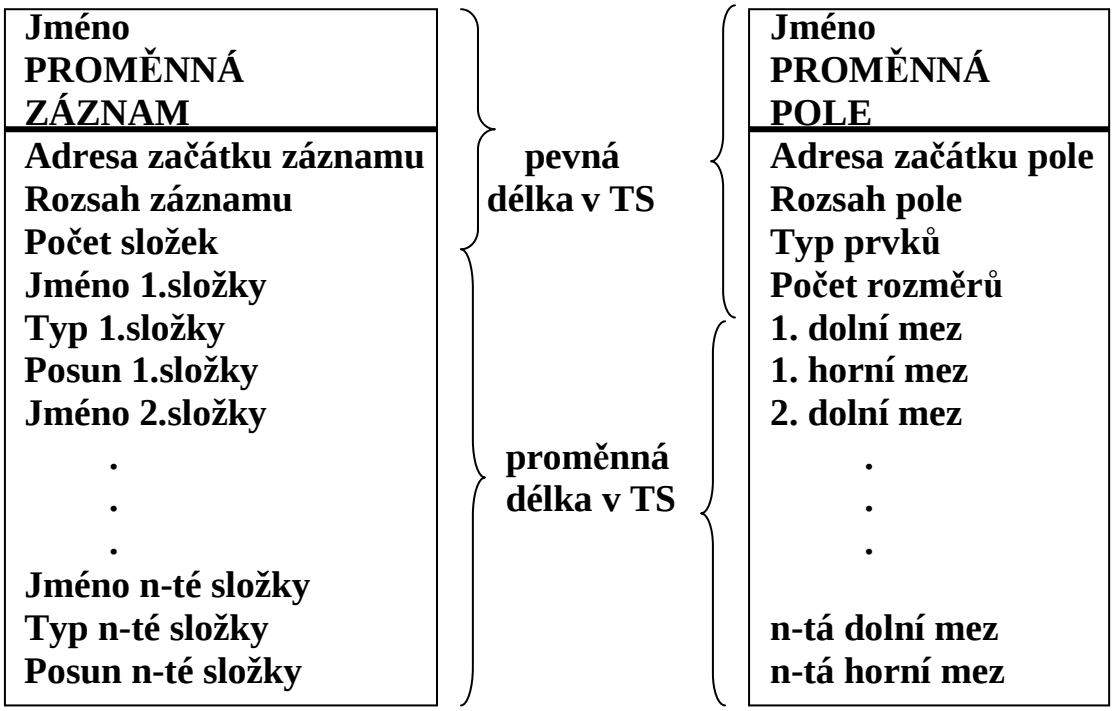
-Výpočtová složitost

? Jaká je časová náročnost v závislosti na počtu položek?

Překlad deklarace datových struktur

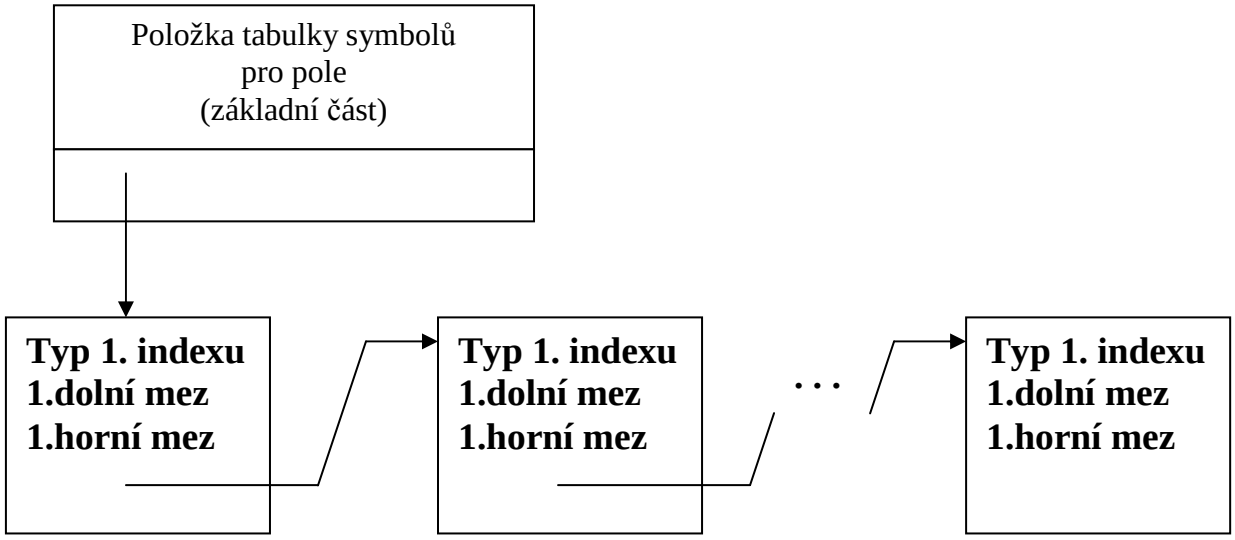
Údaj o uspořádání strukturovaného typu se nazývá deskriptor

Statické struktury – deskriptor lze celý vytvořit a dát do TS při překladu
Dynamické struktury – hodnoty v deskriptoru se plní v čase výpočtu



A) Položka TS a deskriptor záznamu **B) Položka TS a deskriptor pole**

Alternativní struktura položky TS pro pole
(údaje o dimenzích jsou řetězeny do seznamu)



Ukládání polí

Předp. deklaraci tvaru: **array[D1..H1, D2..H2, …, Dn..Hn] of T**

Adresa prvku A[i1, i2, …in] **=** **adresa začátku pole + hodnota mapovací fce**

Adresa začátku pole = adresa prvku s nejnižšími indexy
Mapovací fce = posun prvku vzhledem k začátku pole

$$\text{adr.prvku [i1,i2,…in]} = \sum_{k=1}^{n} (ik - Dk) * Kk$$

konstanty

dolní meze

pro pm(T) = počet míst paměti k uložení prvku typu T platí:

a) při uložení pole po řádcích

$K_n = pm(T)$

$K_k = (H_{k+1} - D_{k+1} + 1) * K_{k+1}$ **pro k= n–1, …, 2, 1**

b) při uložení pole po sloupcích

$K_1 = pm(T)$

$K_k = (H_{k-1} - D_{k-1} + 1) * K_{k-1}$ **pro k=2, 3, …, n**

Mapovací funkci lze rozdělit

$$\text{Mapovací funkce} = \sum_{k=1}^{n} (ik*Kk) - \sum_{k=1}^{n} (Dk*Kk)$$

konstantní částí je (adresa_začátku pole $- \sum_{k=1}^{n} (Dk*Kk)$ **)**

?co je to za adresu?

Deskriptor pole zahrnuje:
1)typ prvků, 2)počet rozměrů, 3)meze indexů, 4)koeficienty map.fce, 5)konstantní část map.fce, 6)velikost paměti pro pole
U dynamického pole se 3-6 dopočítají při běhu programu

Př.

Pole: **a21 a22 a23 a24 a25**
a31 a32 a33 a34 a35

nechť je: Pm(T) = 10
Začátek pole na 100

Uložení po řádcích: 100-109 110-119 … 140-149
150-159 160-169 … 190-199

D1=2, H1 =3
D2=1, H2 =5
K2=Pm(T)=10
K1=(H2-D2+1)xK2=(5-1+1)x10 = 50
Konstantní část = adr.začátku - (D1xK1 + D2xK2) =
100 -110 = -10

Např. a22 bude na adrese i1 x K1 + i2 x K2 + konst.část
=
=2 x 50 + 2 x 10 – 10 = <u>110</u>

Uložení po sloupních: 100-109 120-129 … 180-189
110-119 130-139 … 190-199

K1 = 10
K2 = (H1-D1 + 1) x K1 = (1+1) x 10 = 20
Konst. část = 100 – (2 x 10 + 1 x 20) = 60

Např a22 bude na adrese 2 x 10 + 2 x 20 + 60 = <u>120</u>

Ukládání záznamů

Předp. deklaraci tvaru: struct p1:T1, p2:T2, …, pn:Tn end;

Přístup k položce p záznamu Z

$$adresa(Z.p) = adresa\_začátku(Z) + posun(p)$$

Pro posun položky pi platí:

$$Posun(pi) = \sum_{j=1}^{i-1} rozsah(Tj)$$

Překlad objektových konstrukcí

Deklarace třídy:

```
class JMENOTRIDY extends SUPERCLASSJMENO {
        datovepolozky     /* jako zaznamy*/
        metody       /*maji implicitni parametr this typu JMENOTRIDY /*
}  /* pri prekladu se vytvori descriptor tridy v TS */
```

Deskriptor třídy obsahuje

- Ukazatel na deskriptor rodiče
- Seznam datových položek
  - jejich ofset
  - údaj o privat
- Seznam metod
  - jejich vstupní bod
  - údaj o static
  - údaj o privat

Vytvoření objektu včetně inicializace jeho datových prvků lze provést:

```
a) objektpromenna = new JMENO TRIDY;  // vytvoří object na haldě
        /*vytvoří se při výpočtu ClassInstanceRecord (CIR)*/
        /* „       při překladu      v TS       jméno
                                                 typ objekt
                                                 odkaz na deskriptor tridy
        */
b) deklarací:  jméno_třídy  jméno_instance; // alokuje se v zásobníku
        /*při zpracování deklarace výpočtem se vytvoří CIR, jeho zásobníková
              adresa a velikost byla určena při překladu
        */
```

Volání metody

```
objektpromenna . metoda (parametry) ;
```

-U statických hledá se při překladu vstupní bod v deskriptoru třídy, nenajde-li se, pak v deskriptoru nadtřídy, …

-U dynamických obsahuje descriptor třídy ještě ukazatel na tabulku virtuálních metod (VMT). Před prvým voláním metody je nutné vyvolat konstruktor. Ten propojí instanci volající konstruktor s VMT.

Odkaz na datovou položku

```
objektpromenna . datovapolozka    // stejný mechanismus jako záznamy
```

Jednoduchá dědičnost datových položek

Potomek má zděděné položky na začátku svého rekordu, v pořadí jako u rodiče, nedělené položky jsou uloženy za nimi.

Př. **class R { int x = 0;}**
class P1 extends R { int y = 0; int z = 0; int u = 0; }
class P11 extends P1 { int v = 0; }
class P2 extends R {int y = 0; }

R	P1	P2	P11
x	x	x	x
	y	y	y
	z		z
	u		u
			v

V podstatě jako záznam, s rozdílem

- **lze použít objekt typu potomka i tam, kde se očekává předek**

Překlad metod - obdoba překladu funkcí

Statické metody

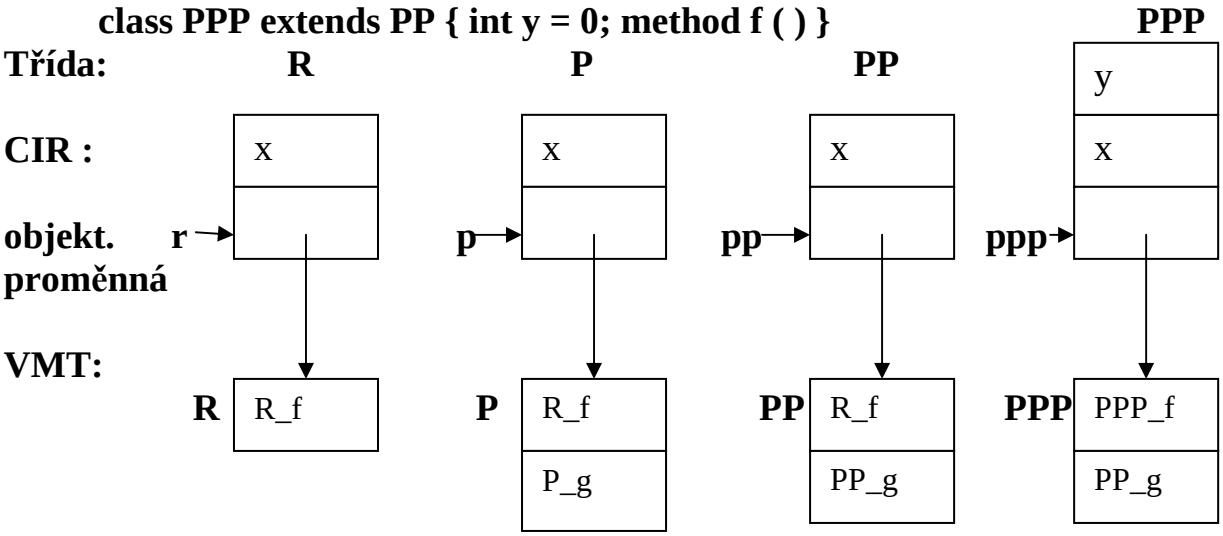
c . f( ) **překlad volání f závisí na typu objektové proměnné c**

-překladač vyhledá třídu C objektu c
-v C hledá metodu f
-když jí nenajde, hledá jí v rodiči C
-…
-pokud program není chybný, v nějakém předkovi jí najde
-volání se přeloží do skoku na její vstupní bod.

Dynamické metody

Mohou být překryty ve třídě potomka.
V době překladu nelze určit zda se jedná o metodu předka či potomka

Př. **class R { int x = 0; method f ( ) }**
class P extends R { method g ( ) }
class PP extends R { method g ( ) }
class PPP extends PP { int y = 0; method f ( ) }



Je-li proměnná pp ukazatel na objekt z třídy PP (může ukazovat na objekt třídy PP a jejích potomků),
nelze určit při překladu zda volání pp . f ( )
je voláním R_f -ukazuje-li pp na objekt třídy PP
nebo voláním PPP_f -ukazuje-li pp na objekt třídy PPP

Řešení:

- **Deskriptor třídy musí obsahovat vektor s metodami pro každé ze jmen nestatických metod (VMT)**
- **Když třída P dědí z R, pak VMT pro P začíná se vstupními body všech metod platných pro R a pokračuje s novými metodami zavedenými v P**
- **Pokud třída PPP překryje metodu R_f, bude na místě R_f ve VMT pro PPP adresa PPP_f**
- **Při exekuci pp . f ( ) musí přeložený kód provést**
  1. **Zjistit deskriptor třídy objektu na který ukazuje pp,**
  2. **Z něj zjistit adresu vstupního bodu metody f**
  3. **Skok do podprogramu na adresu tohoto vstupního bodu**

Násobná dědičnost

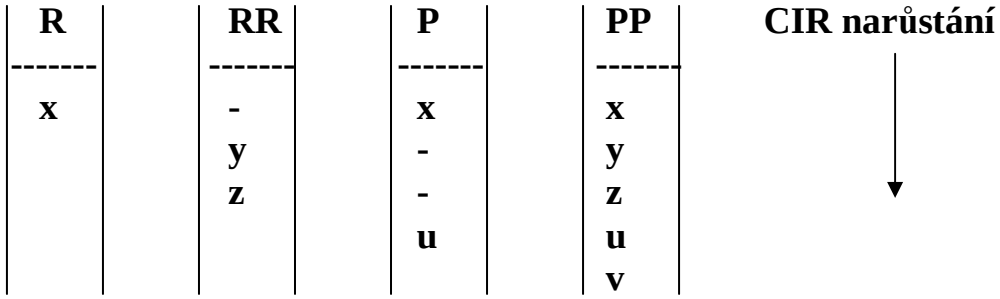
- Python upřednostní dříve uvedené

- Nelze vložit na začátek deskriptoru potomka jak položky rodiče R1, tak i rodiče R2.

Obtížné nalezení ofsetu položek a metod

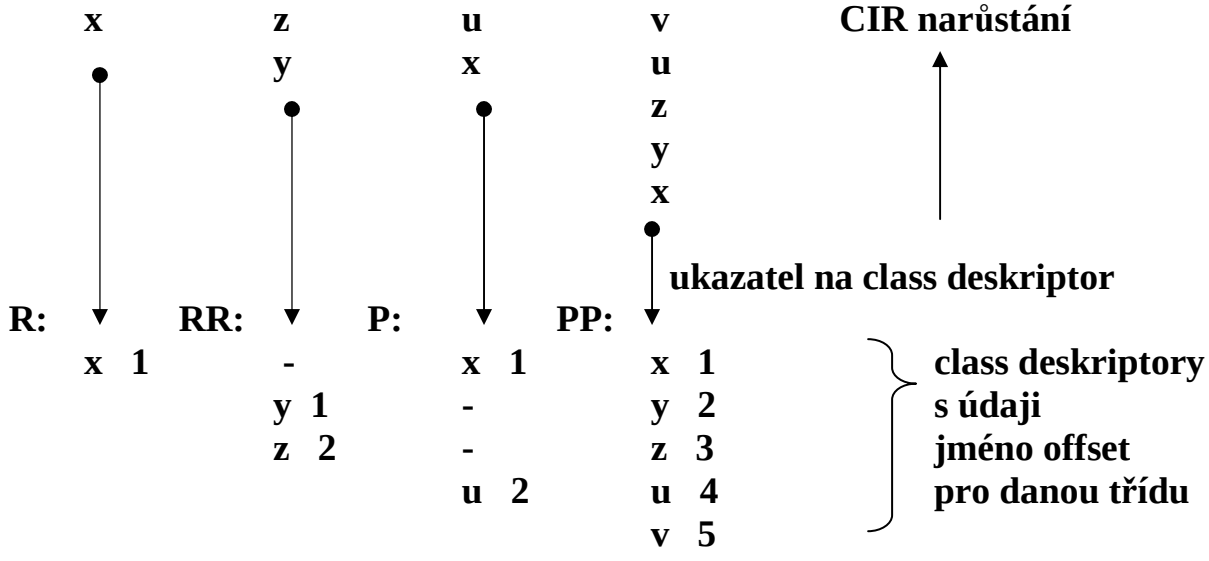
Řešitelné např. barvením grafu
- **uzly jsou jména položek**
- **hrany spojují koexistující položky třídy**
- **barvy jsou ofsety (0, 1, 2, …)**

Př. **class R { int x =0 }**
class RR { int y =0; int z = 0 }
class P extends R { int u = 0 }
class PP extends R, RR, P { int v = 0 }



Zůstávají prázdné sloty v objektech (v CIR)

Řešení je komplikované:
Zapakovat položky v CIR každého z objektů a zaznamenat ofsety v class deskriptoru.



Ušetří paměť, ale potřebuje více operací