

Zpracování deklarácí a přidělování paměti

- **Účel deklarácí**
 - 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

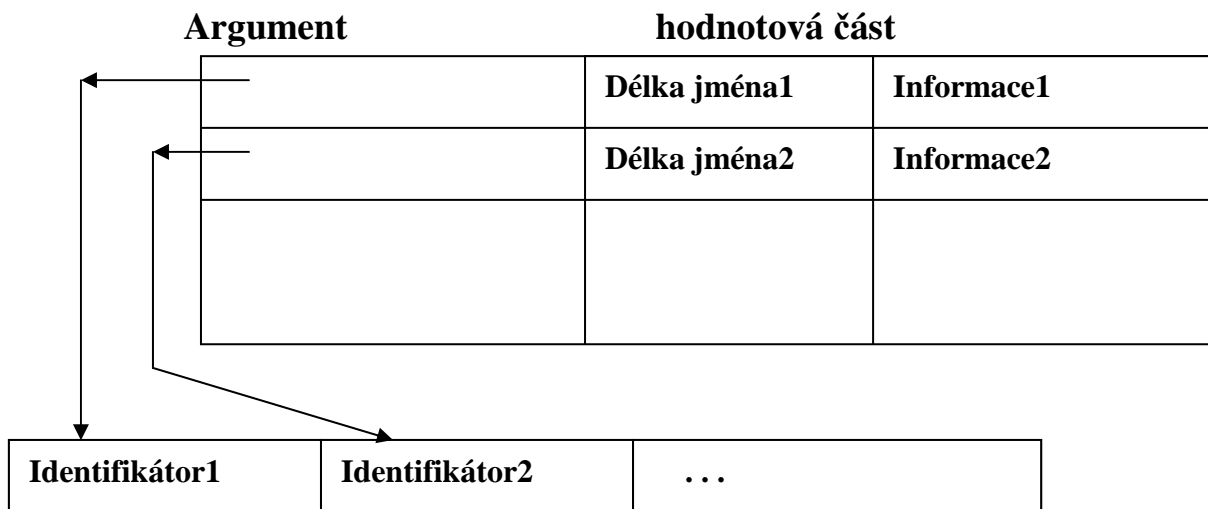
- s jednoduchou strukturou
- s oddělenou tabulkou identifikátorů
- s oddělenou tabulkou informací
- uspořádané do podoby zásobníku
- s blokovou strukturou

Tabulka symbolů:

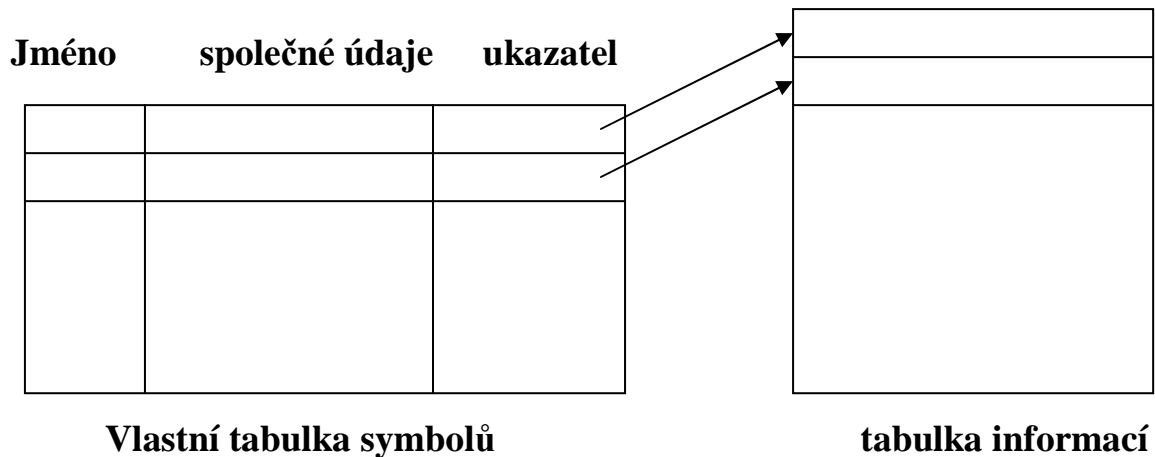
Argument= jméno | hodnotová část= atributy

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

Tabulka symbolů s oddělenou tabulkou identifikátorů (neomezená jména)



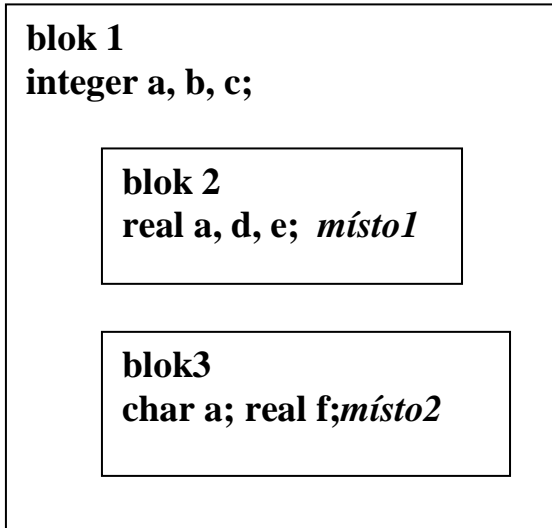
Tabulka symbolů s oddělenou tabulkou informací



Tabulka symbolů uspořádaná do podoby zásobníku (pro jazyky s blokovou strukturou).

Rozsahová jednotka je blok, modul, funkce, balík,...

Respektuje zásady lokality



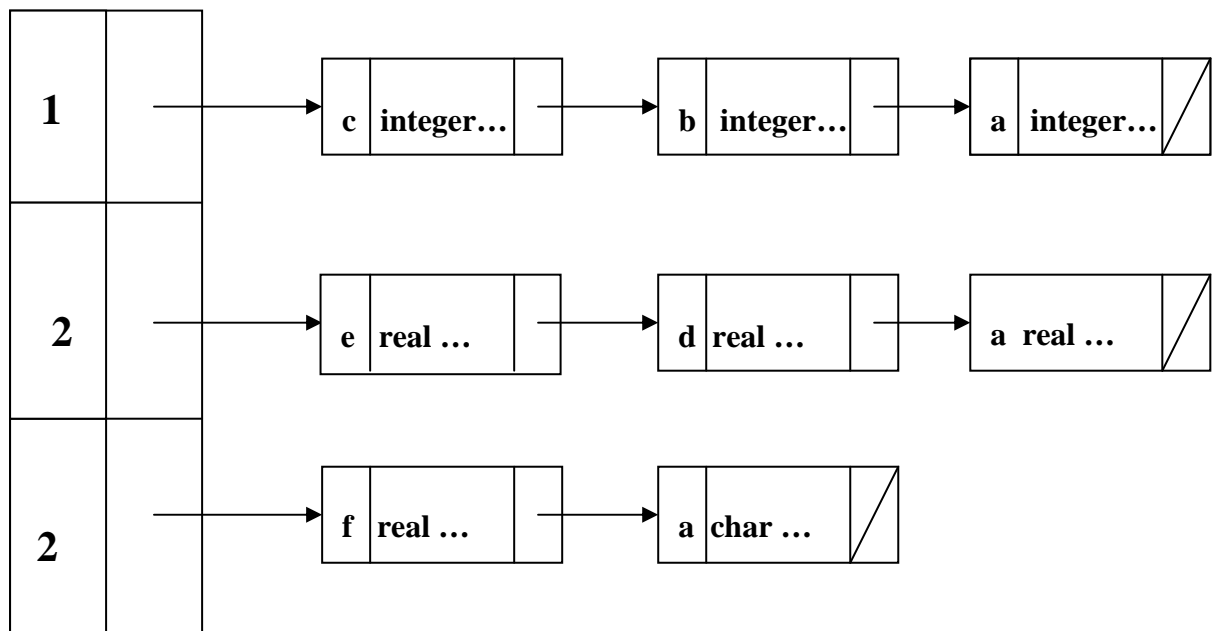
Vrchol → e real, ...
d real, ...
a real, ...
c integer, ...
b integer, ...
a integer, ...

↓
směr prohledávání

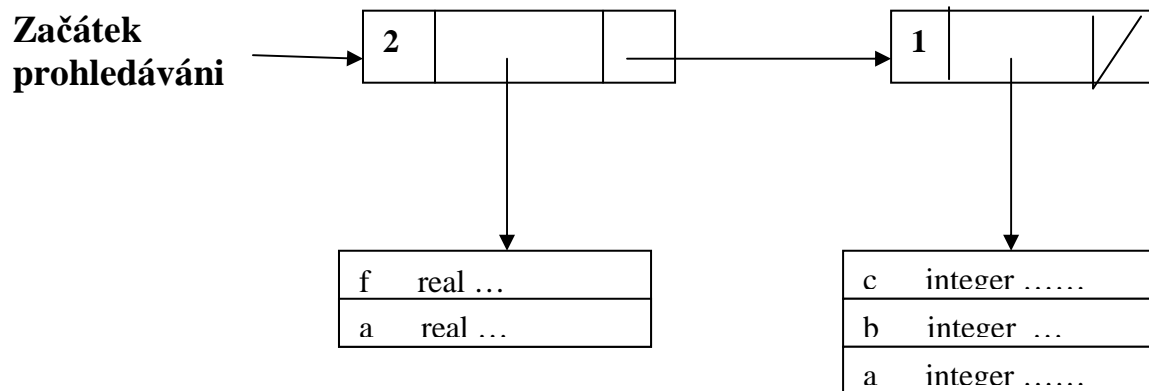
Vrchol → f real, ...
a character, ...
c integer, ...
b integer, ...
a integer, ...

↑
směr plnění

Tabulka symbolů s blokovou strukturou



Překládá-li se uvnitř bloku 3:



Diskutujte

- ? jak řešit případ, kdy jazyk dovoluje použít jména před jejich deklarací
- ? jak řešit případ, kdy jazyk dovoluje přetěžování jmen

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**
- třída**

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

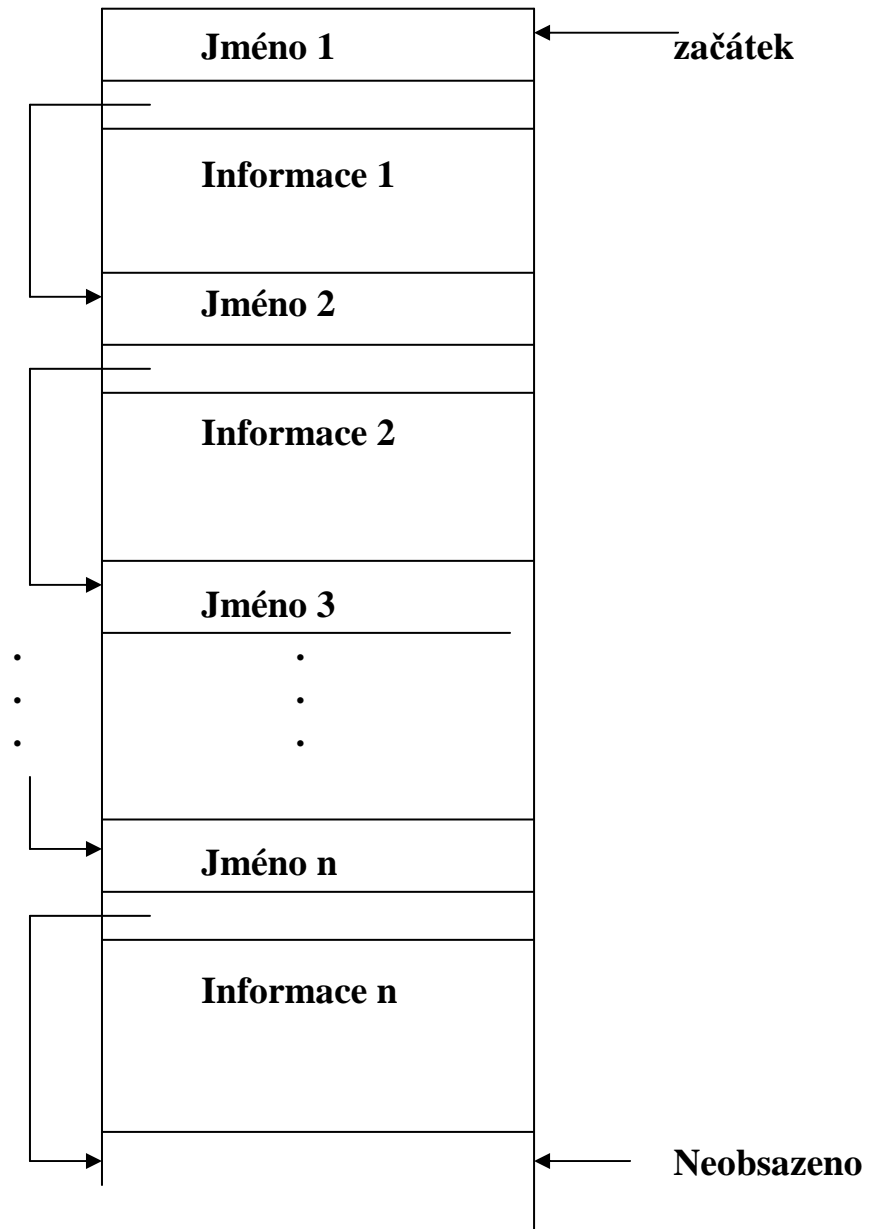
7. Druhy formálních parametrů kolik jich má a kde jsou v TS

8. Způsob volání hodnotou/odkazem

9. Hodnota jen u konstant

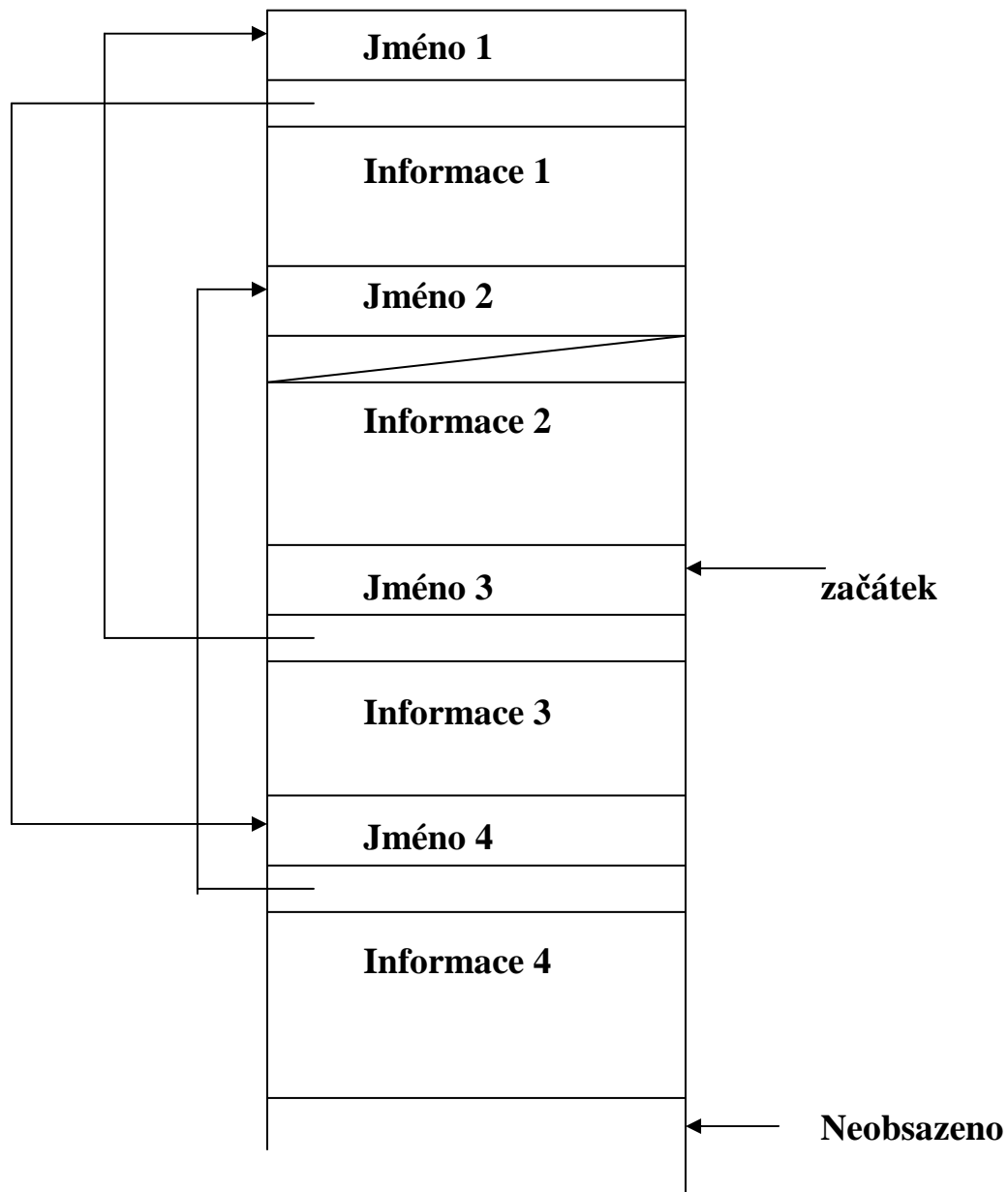
Implementace tabulky symbolů

- **Vyhledávací netříděné tabulky (jen pro krátké programy)**
 - prostá struktura
 - lineární seznam
- **Vyhledávací setříděné tabulky**
 - průběžné setřídování
 - setřídění po zaplnění
- **Frekvenčně uspořádané tabulky**
 - **Binární vyhledávací stromy**
- **Tabulky s rozptýlenými položkami**



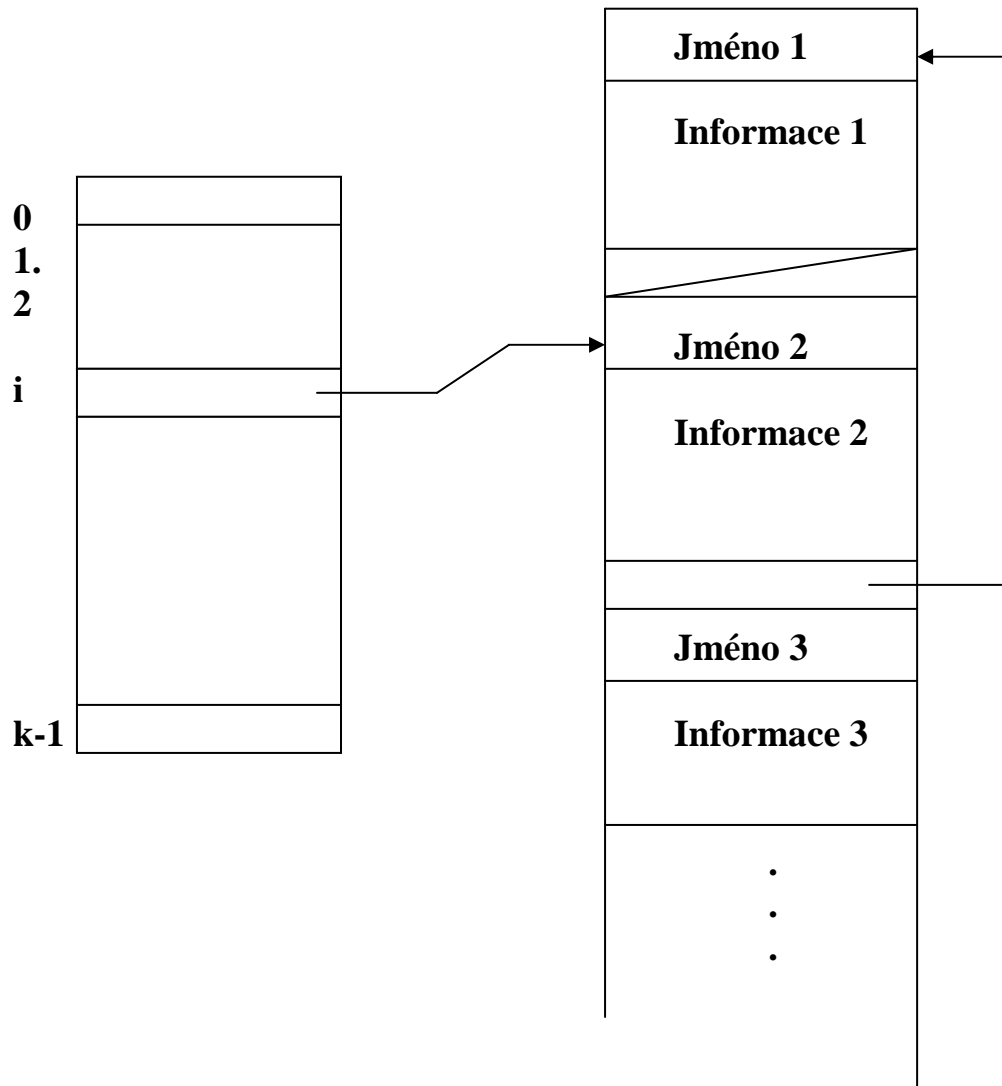
Tabulka symbolů organizovaná jako lineární seznam

? 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

- Jméno upravíme na n znaků
- Rozptylovací funkce $h(\text{jméno}) = \text{zbytek po dělení } (\sum \text{ord}(\text{znak}))/k$
- Řešení kolizí
- Výpočtová složitost

? Jaká je časová náročnost v závislosti na počtu položek?

Překlad deklaráce datových struktur

Údaj o uspořádání strukturovaného typu se nazývá deskriptor

Statické struktury – deskriptor lze celý vytvořit a dát do TS při překladu

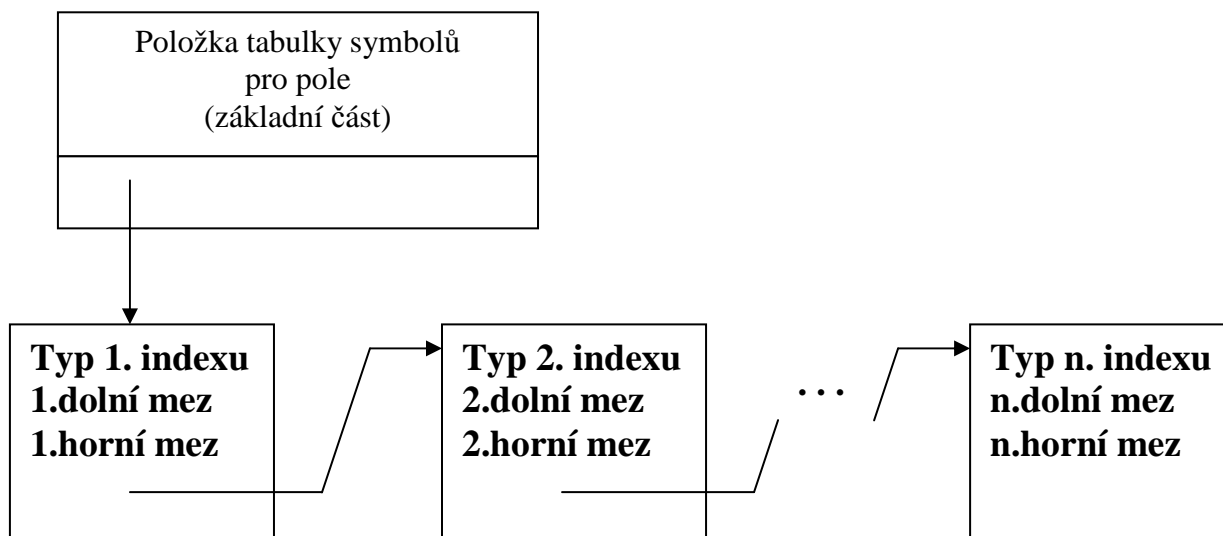
Dynamické struktury – hodnoty v deskriptoru se plní v čase výpočtu

Jméno PROMĚNNÁ ZÁZNAM	}	pevná délka v TS	}	Jméno PROMĚNNÁ POLE
Adresa začátku záznamu Rozsah záznamu Počet složek Jméno 1.složky Typ 1.složky Posun 1.složky Jméno 2.složky . . . Jméno n-té složky Typ n-té složky Posun n-té složky				Adresa začátku pole Rozsah pole Typ prvků Počet rozměrů 1. dolní mez 1. horní mez 2. dolní mez . . . n-tá dolní mez n-tá horní mez

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: $\text{array}[D1..H1, D2..H2, \dots, Dn..Hn]$ of T

Adresa prvku $A[i_1, i_2, \dots, i_n]$ = adresa začátku pole
+ hodnota mapovací fce

Adresa začátku pole = adresa prvku s nejnižšími indexy

Mapovací fce = posun prvku vzhledem k začátku pole

$$\text{hod.map.fce.prvku } [i_1, i_2, \dots, i_n] = \sum_{k=1}^n (i_k - D_k) * K_k$$

konstanty=koef.indexů
dolní meze indexů

pro $\text{pm}(T)$ = počet míst paměti k uložení prvku typu T platí:

a) při uložení pole po řádcích

$$K_n = \text{pm}(T)$$

$$K_k = (H_{k+1} - D_{k+1} + 1) * K_{k+1} \quad \text{pro } k = n-1, \dots, 2, 1$$

b) při uložení pole po sloupcích

$$K_1 = \text{pm}(T)$$

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

Mapovací funkci lze rozdělit

$$\text{Mapovací funkce} = \sum_{k=1}^n (i_k * K_k) - \sum_{k=1}^n (D_k * K_k)$$

konstantní částí je $(\text{adresa_začátku pole} - \sum_{k=1}^n (D_k * K_k))$
?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

U statických je součástí tabulky symbolů

Ukládání záznamů

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

Přístup k položce p záznamu Z

$$\text{adresa}(Z.p) = \text{adresa_začátku}(Z) + \text{posun}(p)$$

Pro posun položky p_i platí:

$$\text{Posun}(p_i) = \sum_{j=1}^{i-1} \text{rozsah}(T_j)$$

Překlad objektových konstrukcí

Deklarace třídy:

```
class JMENOTRIDY extends SUPERCLASSJMENO {
    datovapolozky /* jako záznamy*/
    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 offset
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;
/* při výpočtu vytvoří na haldě ClassInstanceRecord (CIR)*/
/* při překladau „ v TS jméno
typ objekt
odkaz na deskriptor tridy
*/
- b) deklarací: jméno_třidy jméno_instance; // alokuje se v zásobníku
/*při zpracování deklarace výpočtem se vytvoří CIR, jeho zásobníková
adresa a velikost byla určena při překladau
*/

Volání metody

objektpromenna . metoda (parametry) ;

-U statických hledá se při překladau 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 prvním 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, neděděné 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
překladač to zkontroluje podle údajů v tab.symbolů

Překlad metod - obdoba překladu funkcí

Statické metody

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

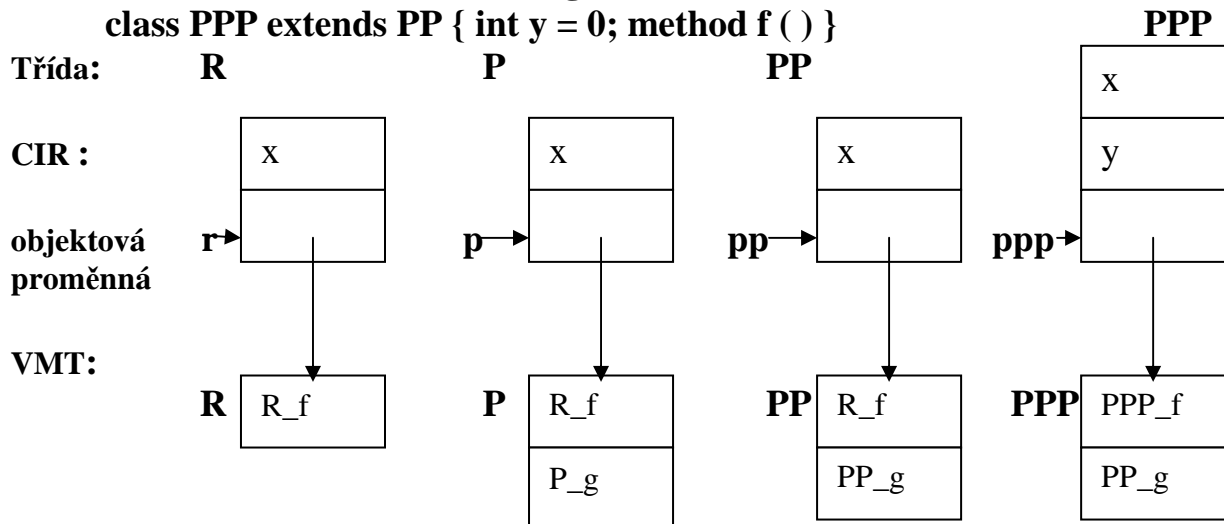
- překladač vyhledá třídu C objektu c
- v C hledá metodu f
- když jí nenajde, hledá jí v rodiči C (informaci má v deskriptoru)
- ...
- pokud program není chybný, v nějakém předkovi jí najde
- volání se přeloží do skoku na její vstupní bod.

Dynamické metody

Mohou být překryty ve třídě potomka.

V době překladač 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 () }
```



R_f označuje metodu **f** z třídy **R**, ...

Ukazatel na VMT se získá pro CIR pomocí deskriptoru třídy.

Je-li proměnná **pp** ukazatel na objekt z třídy **PP** (může ukazovat na objekt třídy **PP** a jejích potomků), nelze určit při překladač 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

CIR musí ukazovat na deskriptor třídy, ten se uchovává i v run-time

Násobná dědičnost (C++, Smalltalk, Python)

- Python upřednostní dříve uvedené

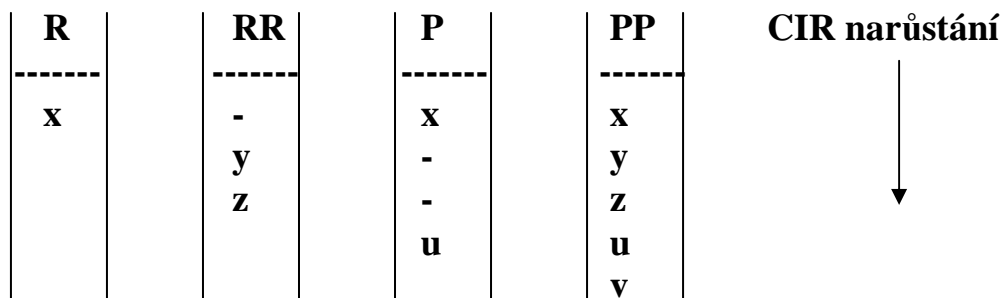
- Nelze vložit na začátek deskriptoru potomka jak položky rodiče R1, tak i rodiče R2, nutno zabránit opakovanému dědění.

Obtížné nalezení offsetu položek a metod

Řešitelné např. barvením grafu

- uzly jsou jména položek
- hrany spojují koexistující položky třídy
- barvy jsou offsety (0, 1, 2, ...)

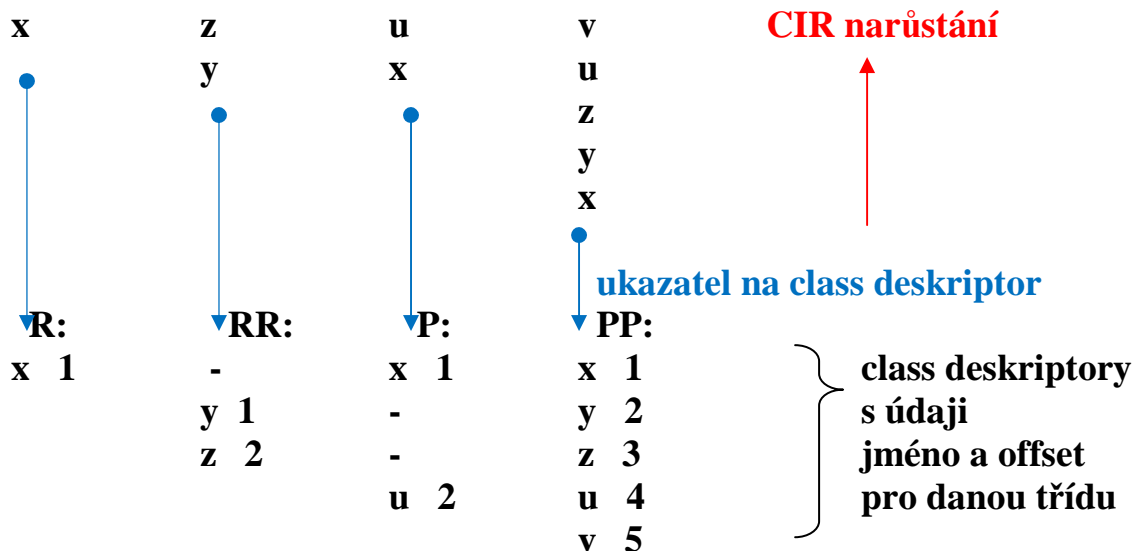
Př. `class R { int x=0 }`
`class RR { int y=0; int z=0 }`
`class P extends R { int u=0 }`
`class PP extends R, RR, P { int v=0 }` //tady existují všechny =5barev



Zůstávají pak prázdné sloty v objektech (v CIR)

Řešení je komplikované:

Zapakovat položky v CIR každého z objektů a zaznamenat offsety v class deskriptoru.



Ušetří paměť, ale potřebuje více operací:

- 1. Vybrat z CIR ukazatel na deskriptor**
- 2. Zjistit z deskriptoru offset**
- 3. Použit zjištěný offset k přístupu k položce**

Deskriptory tříd pak mají i nadále prázdné sloty, ale vlastní objekty ne.