Vlastnosti jazykových konstrukcí pro statický a pro dynamický způsob přidělování paměti

Thursday, May 30, 2013 8:43 AM

Přímo podle Ježka:

Pro jazykovou konstrukci, která umožňuje jen statické přidělování, je možné řešit alokovanou paměť už během překladu. U těch dynamických není předem známo, kolik paměti je třeba vyhradit, je nutné to provádět až během vykonávání programu.

Jsou rekurze (nevíme předem, kolikrát se zanoříme)
Dynamické proměnné (new)
Dynamické typy (pole, jejichž velikost je daná výrazem)

Řeknu si, že chci udělat vlastní programovací jazyk a chci vědět, jak ho mám navrhnout - je třeba si uvědomit, co od něj budu potřebovat. Pokud potřebuju např. rekurzi, musím ho navrhnout tak, aby podporoval dynamické typy.

U statického přidělování paměti si mohu dovolit dělat statické proměnné, metody bez rekurze apod.
U dynamického pak vše ostatní jako rekurzi apod.

http://service.felk.cvut.cz/courses/X36PJP/Skripta_prednasky.pdf !!!

Důležitá hlediska jazykových konstrukcí:

- Dynamické typy
- Dynamické proměnné
- Rekurze
- Konstrukce pro paralelní výpočty

Podstatný je rovněž způsob:

- Omezování existence entit v programu
- Předávání parametrů funkce (hodnotou, odkazem)
- Určování přístupu k nelokálním entitám
    - na základě statického vnořování rozsahových jednotek,
    - na základě dynamického vnoření rozsahových jednotek.

Statické přidělování paměti:

- Globální proměnné
- Statické proměnné
- Proměnné jazyka bez rekurze (i s blokovou strukturou) (možno staticky na zásobníku)

Dynamické přidělování paměti (na hromadě):

- Dynamické typy a proměnné (překladač neví v době překladu kolik jich bude potřebovat)
- Proměnné předávané odkazem
- Pointery v C pro dynamicky alokované proměnné

Předávání parametrů podprogramům

- hodnotou (C, C++, Java, C#) formální parametr podprogramu je lokální proměnnou (tohoto podprogramu) do níž se předá hodnota (proměnná je zkopírována do zásobníku podprogramu)
- odkazem (C, C++ je-li parametrem pointer, objektové parametry Javy, C# parametry označené ref) předá informaci o umístění skutečného parametru (předána adresa na proměnnou) - u Javy se všechno předává prostřednictvím hodnoty (pass-by-value) tj. v případě instancí tříd (objektů) dochází k předávání **adresy** objektu ???
- výsledkem - formální parametr je lokální proměnnou z níž se předá hodnota do skutečného parametru před návratem z podprogramu

Dynamické přidělování v zásobníku

Aktivační záznam obsahuje místo pro:

- Lokální proměnné
- Parametry
- Návratovou adresu
- Funkční hodnotu (je-li podpr. funkcí)
- Pomocné proměnné (pro mezivýsledky)
- Další informace potřebné k uspořádání aktivačních záznamů

Statická typová kontrola – referenční prostředí podprogramů je definováno staticky, tj. při překladu zdrojového programu. Pro každou deklaraci je staticky vymezen rozsah platnosti, tj. část zdrojového kódu, ve kterém lze deklarované jméno použít. V podprogramu pak kromě lokálních jmen lze použít ta nelokání jména, do jejichž rozsahu platnosti je definice podprogramu vnořena. Statické referenční prostředí je často založeno na blokové struktuře programu.

Statické referenční prostředí je při překladu reprezentováno tabulkou symbolů. Překlad deklarace znamená rozšíření tabulky symbolů o nový záznam, při dosažení místa konce platnosti deklarace se záznam odstraní nebo skryje. Při překladu těla podprogramu překladač na základě tabulky symbolů pro každé jméno rozhodne, zda je či není v daném místě použitelné a jaký datový objekt (nebo jiný programový prvek) označuje. Tím se dosáhne vyšší bezpečnosti programu (nepoužitelné jméno je odhaleno již při překladu) i vyšší efektivity cílového programu.

Dynamická typová kontrola – např. Lisp, referenční prostředí podprogramů je definováno dynamicky. Dynamicky definované referenční prostředí **se nevytváří ani nekontroluje při překladu, ale až při provádění programu**. Při spuštění programu se vytvoří referenční prostředí tvořené vazbami jmen definovaných jazykem. Při každém vstupu do podprogramu se referenční prostředí rozšíří o vazby lokálních jmen podprogramu, při návratu z podprogramu se jeho lokální prostředí odstraní. Při provádění příkazů se pro každé jméno hledá jeho vazba.

Dyn. Definované ref. Prostředí snižuje bezpečnost programu i jeho efektivitu (význam jména se hledá při provádění programu). Jeho výhodou je jednoduchá sémantika – nenajde-li se jméno v lokálním prostředí podprogramu A, hledá se v lokálním prostředí podprogramu B, ze kterého byl podprogram A vyvolán, případně v lokálním prostředí podprogramu C, z čehož byl vyvolán podprogram B apod.

From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx>

Printout

Monday, June 3, 2013 1:08 PM

Metody přidělování paměti

Základní způsoby: -Statické (přidělění paměti v čase překladu)
-Dynamické (přiděleno v run time) — v zásobníku / na haldě

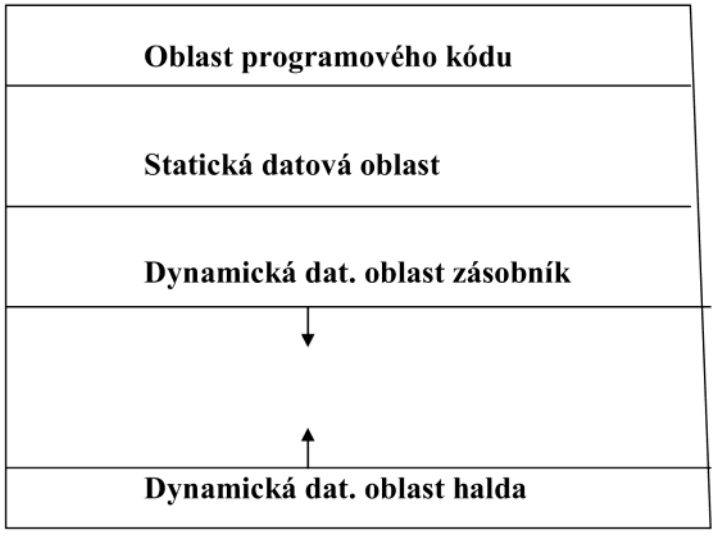
Důležitá hlediska jazykových konstrukcí:

- Dynamické typy
- Dynamické proměnné
- Rekurze
- Konstrukce pro paralelní výpočty

Podstatný je rovněž způsob:

- Omezování existence entit v programu (namespace, package, blok…)
- Určování přístupu k nelokálním entitám
  na základě statického vnořování rozsahových jednotek,
  na základě dynamického vnoření rozsahových jednotek.

Rozdělění paměti cílového programu



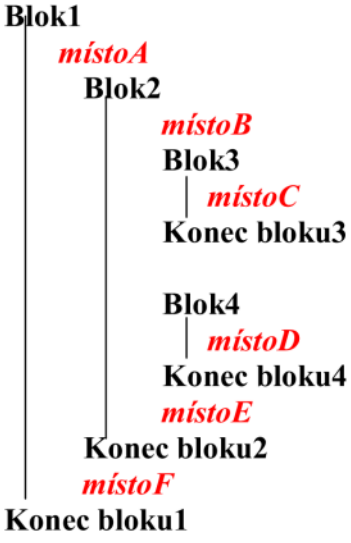
Page 2

Monday, June 3, 2013 1:08 PM

Statické přidělování paměti lze použít pro:

- **Globální proměnné**
- **Static proměnné**
- **Proměnné jazyka bez rekurze (i s vnořenou blokovou strukturou)**

Př.



Statické přidělování lze realizovat pomocí zásobníku

Ukažme obsah zásobníku v různých okamžicích výpočtu

Bloky

		3	4		
	2	2	2	2	
1	1	1	1	1	1
Místo A	**B**	**C**	**D**	**E**	**F**

Vnořování podprogramů (funkcí, metod) je ale složitější, viz dále.

Page 3

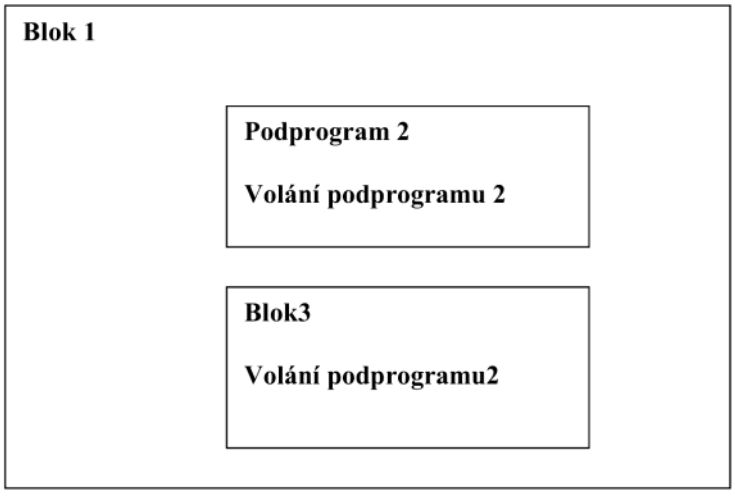
Monday, June 3, 2013 1:08 PM

Dynamické přidělování v zásobníku

Část paměti přidělovaná při vstupu výpočtu do rozsahové jednotky programu se nazývá Aktivační Záznam (AZ představuje lokální prostředí výpočtu). Obsahuje místo pro:

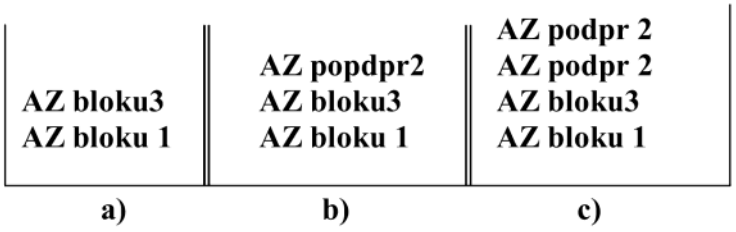
- **Lokální proměnné**
- **Parametry (je-li rozsahovou jednotkou podprogram či funkce)**
- **Návratovou adresu („ „)**
- **Funkční hodnotu (je-li rozsahová jednotka funkcí)**
- **Pomocné proměnné pro mezivýsledky (také možno v registrech)**
- **Další informace potřebné k uspořádání aktivačních záznamů**

Př.1



? stav výpočtového zásobníku v různých časech výpočtu

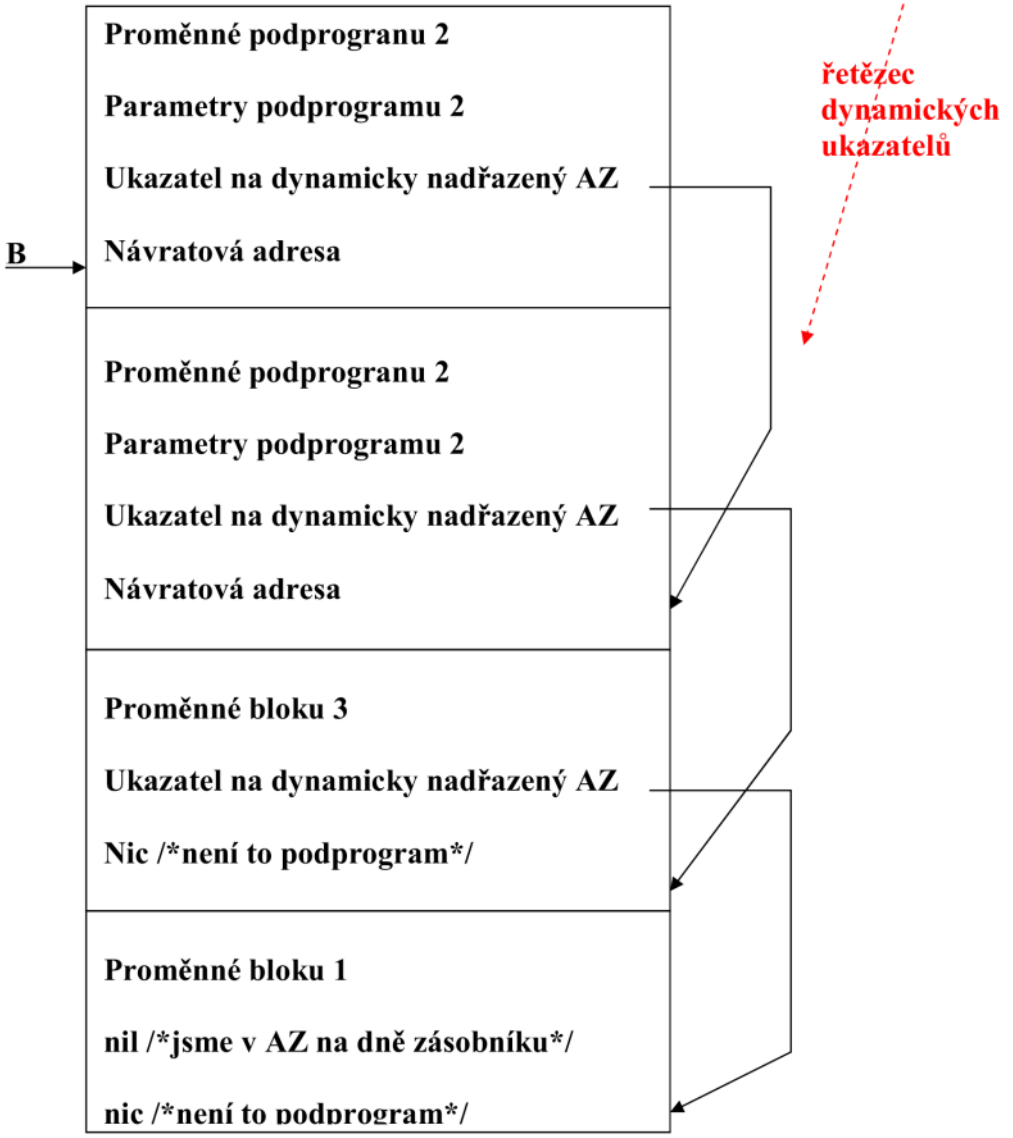
a) **Při vstupu do bloku 3**
b) **Při prvním volání podprogramu 2**
c) **Při rekurzivním vyvolání podprogramu 2**



Page 4

Monday, June 3, 2013 1:08 PM

Jaké bude uspořádání aktivačních záznamů při rekurz. vyvolání podpr.2 ?
Pro rušení AZ při výstupu z jednotky potřebujeme tzv dynamický ukazatel



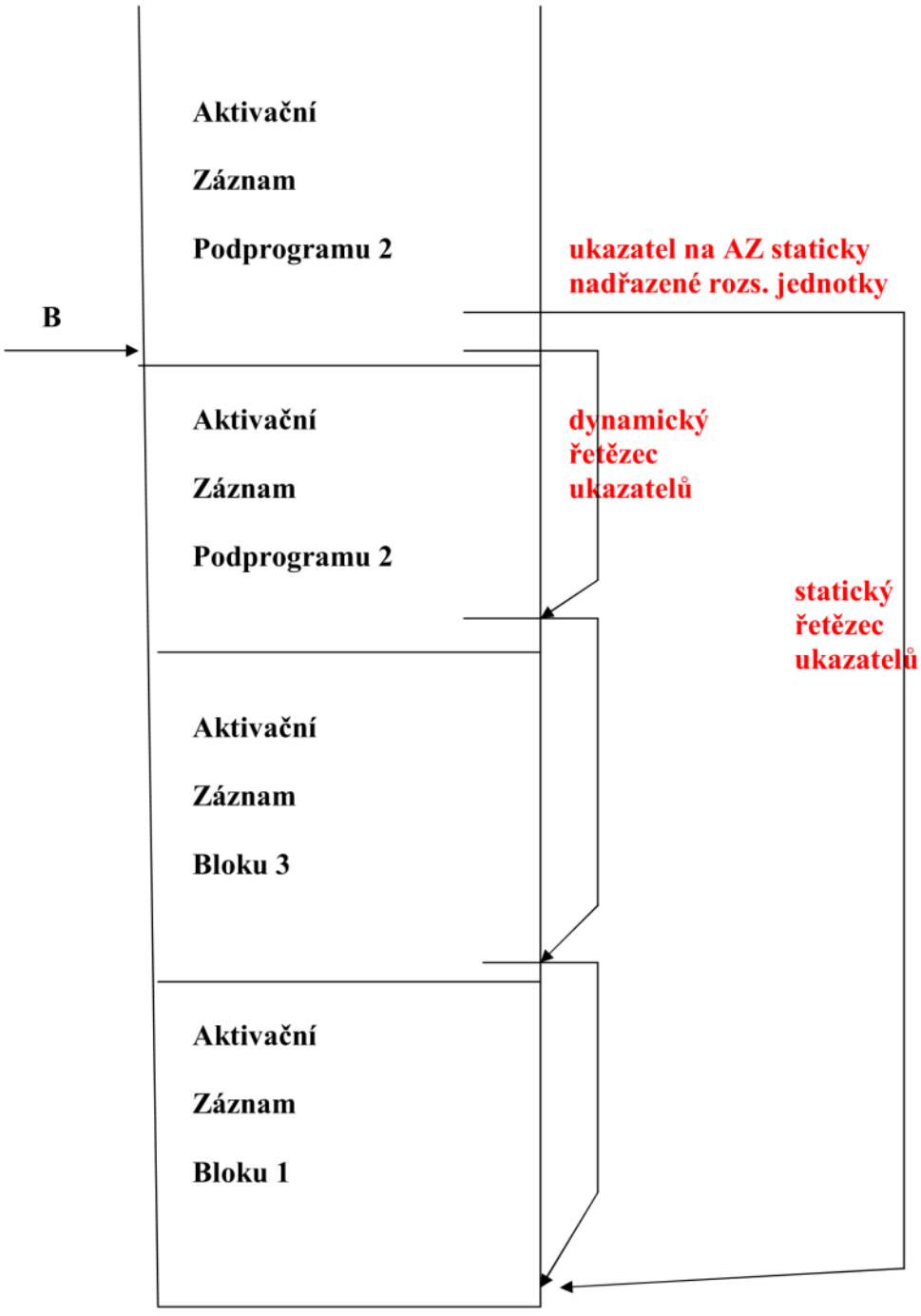
Obr. Zásobník při rekurzivním volání podprogramu 2

B je registr ukazující na vrcholový AZ

Potřebujeme ještě vyřešit přístup k nelokálním proměnným při statickém = lexikálním rozsahu platnosti jmen. To řeší tzv. řetězec statických ukazatelů

Page 5

Monday, June 3, 2013 1:08 PM



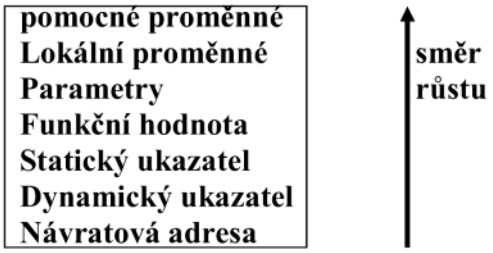
Obr. Zásobník se statickým (ukazuje na lexikálně nadřazený AZ) a dynamickým řetězcem ukazatelů při rekurzivním volání podprogramu 2
Pozn.: Statický uk. je nakreslen (pro přehlednost) jen u AZ rek. volání podpr.2

Page 6

Monday, June 3, 2013 1:08 PM

Vytváření řetězců ukazatelů

Nechť AZ má tvar:



Uvažujme zásobník Z, s vrcholem (nejvyšší zabranou adresou) T,
n je hladina deklarace , m je hladina volání
Při vstupu do rozsahové jednotky (vyvolání podprogramu nebo vstupu výpočtu do bloku = Aktivace rozsahové jednotky):

A1) Z[T + 1] ← návratová adresa /* pouze u podprogramů*/
A2) Z[T + 2] ← B /*nastavení dynamického ukazatele*/
A3) Z[T + 3] ← B
For i ← 1 to m – n do Z[T + 3] ← Z[Z[T + 3] + 2] /*nastavení statického ukazatele*/
A4) B ← T + 1 /*nastavení bázového registru*/
A5) T ← T + velikost aktivačního záznamu
A6) skok na první instrukci podprogramu a uložení do Z údajů o skutečných parametrech /*pouze u podprogramů*/

Pozn. Je-li podprogram překládán odděleně (neznámá velikost jeho AZ), pak je úprava T provedena až na začátku volaného podprogramu.

Při výstupu z rozsahové jednotky (Návrat z podprogramu nebo průchod koncem bloku):

N1) T ← B – 1
N2) B ← Z[B + 1]
N3) skok na adresu uloženou v Z[T + 1] /*pouze u podprogramů*/

Výstup z rozsahové jednotky nelokálním skokem (hladina n deklarace návěští je menší než hladina m místa s příkazem skoku)
Vždy platí n ≤ m

```
S1)  for i ← 1 to m – n do { Pom ← B
                             repeat     T ← B – 1
                                        B ← Z[ B + 1]
                             until  B ≠ Z[ POM + 2]
                           }
```

S2) skok na adresu, kterou návěští představuje

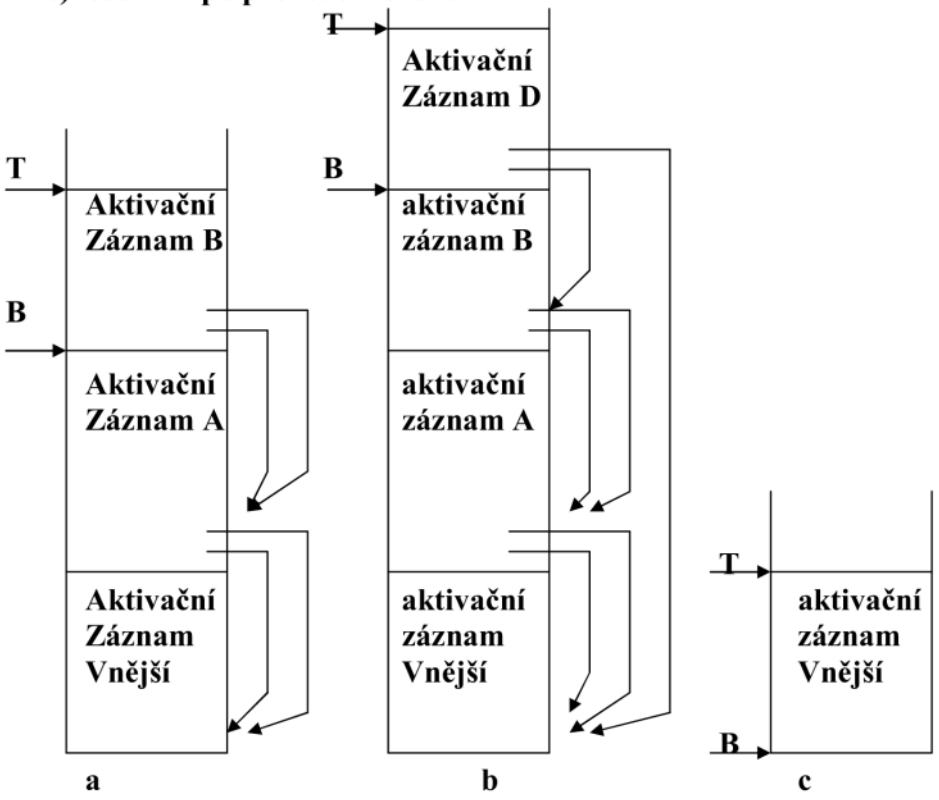
Page 7

Monday, June 3, 2013 1:08 PM

Př.2

```
Podprogram Vnější //označme n hladinu deklarace
Label 1                    // a m hladinu místa použití
        Podprogram A
                Podprogram B
                Volání D  //zde  n=1; m=3
        Volání B
        Podprogram D
        Goto 1        //zde n=1; m=2
Volání A
1:
```

a) obsah Z při provádění B, před voláním D,
b) obsah Z po vyvolání D, ped provedením nelokálního skoku,
c) obsah Z po provedení skoku.

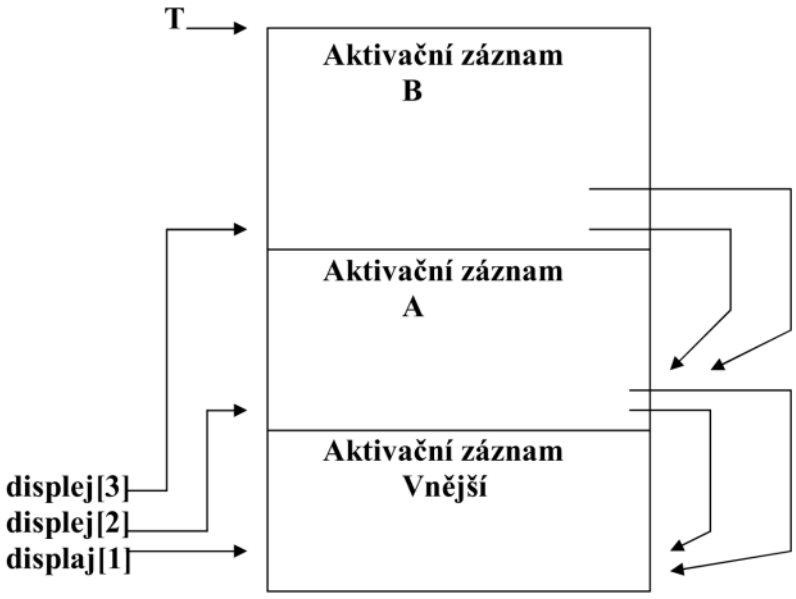


ad b) je to stav v okamžiku volání podpr. s hladinou deklarace 1, volaného v místě s hladinou 3
ad c) stav po výskoku z hladiny 2 do místa s hladinou 1

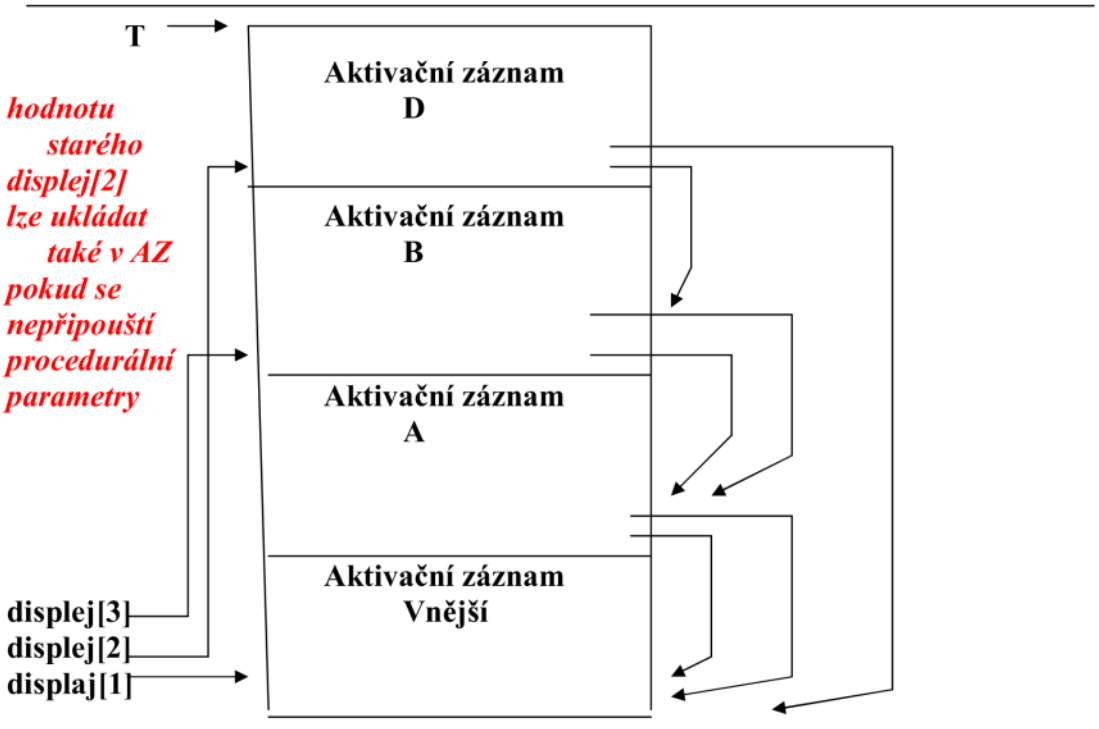
Page 8

Monday, June 3, 2013 1:08 PM

Zrychlení přístupu k nelokálním proměnným
(pomocí vektoru ukazatelů displej[i], kde i je hladina rozs. jedn.)



Obr. Stav Z při výpočtu B z př.2



Obr. Stav Z při výpočtu D z př.2

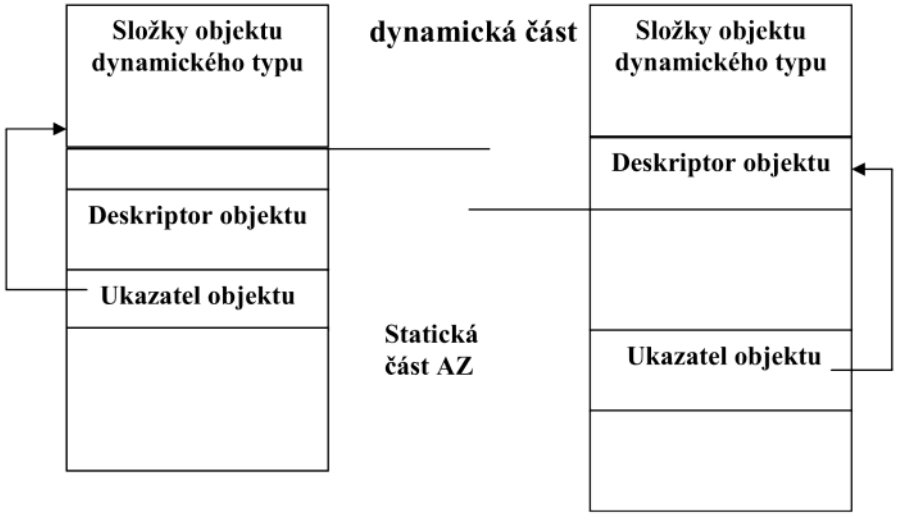
Dynamická adresa proměnné je dvojice (n, p) = displej[n] + p

Page 9

Monday, June 3, 2013 1:08 PM

Objekty s dynamickými typy (typicky pole s proměnnými mezemi)

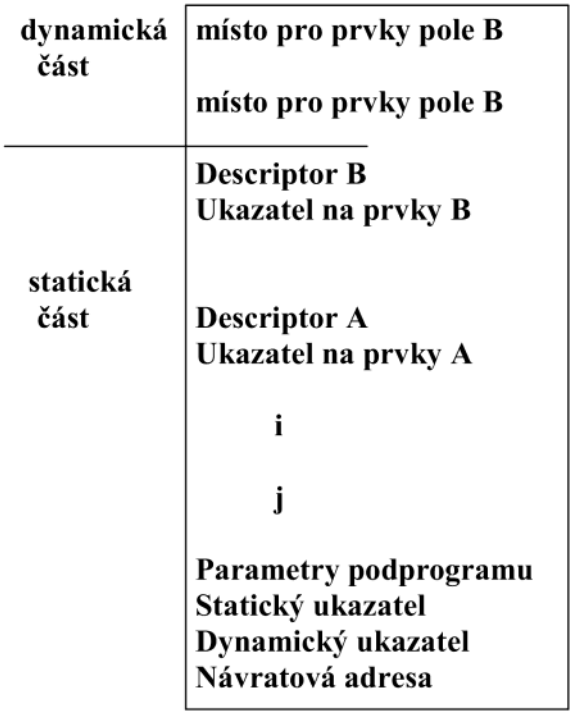
Možnosti struktury aktivačního záznamu s objektem dynamického typu



Deskriptor se vytvoří při překladu, uchovat se ale musí i při výpočtu

Př.3 Aktivačního záznamu s objekty dynamického typu

```
podprogram PRIKLAD;
        int i, j ;
        int A(m .. n);
        int B(p .. q, r .. s,  );
```



Page 10

Monday, June 3, 2013 1:08 PM

Předávání parametrů podprogramům

-hodnotou (C, C++, Java,C#) formální parametr je lokální proměnnou do níž se předá hodnota

-odkazem (C, C++ je-li parametrem pointer, objektové parametry Javy, C# označené ref) předá informaci o umístění skutečného parametru

-výsledkem - formální parametr je lokální proměnnou z níž se předá hodnota do skutečného parametru před návratem z podprogramu slouží jako výstupní parametr

-hodnotou výsledkem (novější Fortran) - kombinace

-jménem – má efekt textové substituce (jako historická zajímavost)

-v případě strukturovaných parametrů

-jsou-li to statické typy ⇒ předá se adresa prvého prvku

-jsou=li to dynamické typy ⇒ předá se ukazatel na descriptor

-je-li parametrem podprogram

-u jazyků nedovolujících hnízdění podprogramů ⇒předá se adresa začátku = pointer

- u jazyků dovolujících hnízdění podprogramů ⇒

spolu s adresou musí předdat i platné prostředí. Jsou různé možnosti co považovat za platné prostředí:

-**mělká vazba** ⇒ platné je prostředí v němž se nachází volání formálního podprogramu

-**hluboká vazba** ⇒platné je prostředí kde je předávaný podprogram definován

-**ad hoc vazba** ⇒platné je prostředí kde je vydán příkaz volání podprogramu jež má za parametr podprogram

Page 11

Monday, June 3, 2013 1:08 PM

Př.4

```
Podprogram  P1( ) {
        Prom x ;
        Podprogram P2 ( ) {
                Vytiskni (x) ; /*co se tady tiskne?*/
        };
        Podprogram P3 ( ) {
                Prom x ;
                x ← 3;
                P4(P2) ;
        };
        Podprgram P4( podprogram Px ) {
                Prom x ;
                x ← 4;
                call Px( );
        }
        x←1;
        P3( ) ;
}
```

prostředí, kde je předávaný podprogram definován

prostředí, kde je vydán příkaz volání s parametrem podprog.

prostředí, v němž je volán formální podprogram

Při mělké vazbě se tiskne … **?**

Při hluboké vazbě se tiskne … **?**

Při ad hoc vazbě se tiskne … **?**

Př.5

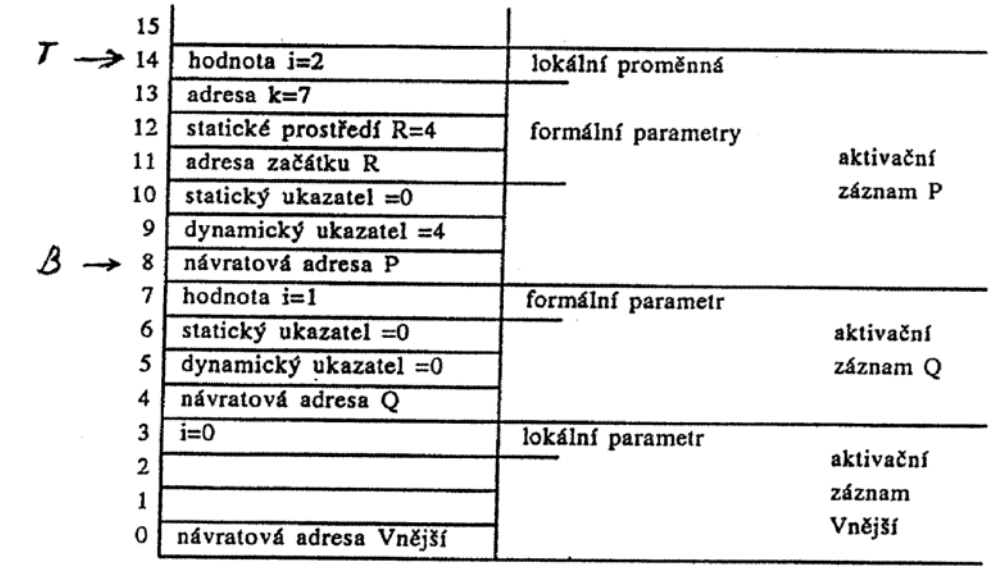
Předpokládejme hlubokou vazbu. Co se vytiskne po spuštění procedury Vnější?

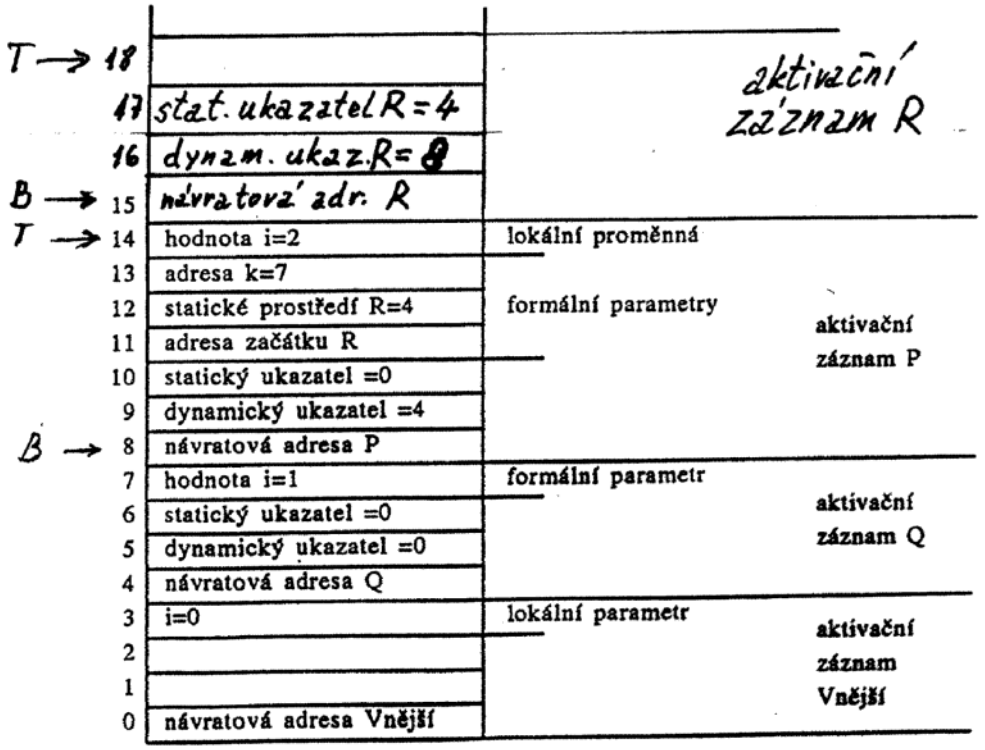
```
podprogram Vnejsi;  {
        prom i:int;
        podprogram P( podprogram FP; prom k:int;)  {
                prom i:int;
                i←k+1;  FP( ); tisk(i);
        }
        podprogram Q(i:int);
                podprogram R ( )  {
                        Tisk(i);
                }
                P(R,i);
        }
        i← 0; Q(i+1);
}
```

Page 12

Monday, June 3, 2013 1:08 PM

Stav před vyvoláním a po vyvolání formálního poprogramu FP z př.5





Page 13

Monday, June 3, 2013 1:08 PM

Přidělování paměti pro paralelní výpočty

Pro uložení AZ paralelního výpočtu nutno použít haldu nebo zobecněný zásobník

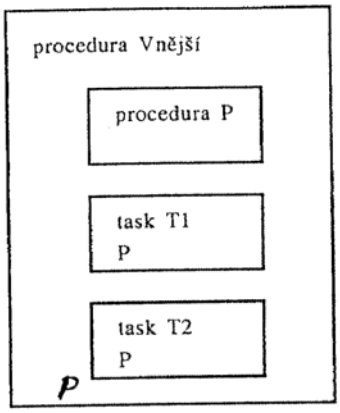
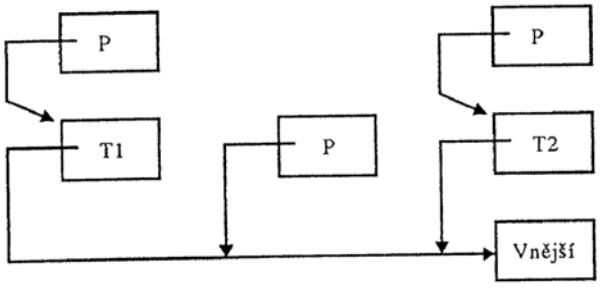
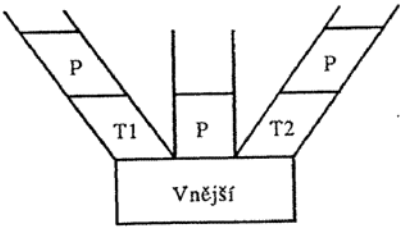


Schéma vnoření bloků programu



Struktura aktivačních záznamů při paralelním výpočtu



Uložení aktivačních záznamů ve zobecněném zásobníku

Page 14

Monday, June 3, 2013 1:08 PM

Př v javovském prostředí