10.11.2024

3. Programovací jazyk – podprogramy

- klíčem strukturální strukturované programování
- rozdělují program na menší, přehlednější části ⟹ pro opakované použití

FUNKCE – vrací hodnotu
PROCEDURY – void

a) Prototyp funkce, návratová hodnota, parametry

- Prototyp funkce = – „Hlavička" funkce (– deklarace funkce)

```
návratový_typ název_funkce(typ_parametru1, typ_parametru2, ....);
```

↳ její název, návratový typ a seznam parametrů

– umožňuje kompilátoru ověřit správné volání funkce (s odpovídajícími parametry)

- návratová hodnota = jaký typ funkce vrátí např. int; char; uživatelsky definovaný typ; void (procedura)
- parametry = vstupní hodnoty, které funkce přijímá (CO DEKLARUJI) – argument = to co tam předáváme
  ↳ formální ↳ skutečný

b) Parametry: předávání hodnotou, odkazem

- předávání hodnotou = do funkce se předá KOPIE hodnoty ⟹ na původní proměnné se neprojeví změny

```
void zvyseJedna(int cislo) { cislo++; }
```

- předávání odkazem = pomocí UKAZATELŮ se předá přímo adresa proměnné ⟹ změny se projeví na původní proměnné

parametr Datový_typ *

```
void zvyseJedna(int *cislo) { *cislo++; } // Dereferencuji ukazatel a zvýšení hodnoty na adrese
```

c) Lokální a globální proměnné

statická = uvnitř funkce, ale je vidět vně
– např. přiřazení hodnoty bude funkce zvolená

- lokální = uvnitř funkce (hlavičky příkazu), na zásobníku, zaniká na konci BLOKU (scope) (funkce); AUTOMATICKÁ DYNAMICKÁ ALOKACE
- globální = vně funkce, v datové oblasti programu, zaniknou na konci programu STATICKÁ ALOKACE

d) Zásobník a fungování podprogramu

- Zásobník vyvolávání pro správné volání funkcí
- Při volání funkce se přidá ZÁZNAM na ZÁSOBNÍK

AKTIVAČNÍ ZÁZNAM
↳ hodnoty parametrů funkce + lokální proměnné

REKURZE

- datová struktura, která obsahuje odkaz na strukturu téhož typu (fronta, zásobník, lineární seznam, strom)
- funkce, která ve svém těle volá samu sebe ⟹ REKURZIVNÍ VOLÁNÍ (např. faktoriál, Quick sort, Merge sort, Hanojské věže)

```
long long faktorial(long long n) {
    if (n==0) { return 1; } // 0! = 1
    return n * faktorial(n-1); // n! = n*(n-1)!
```

2 fáze rekurze: Zanořování, Vynořování
– výpočet směřuje postupně jedna při zanořování, druhá při vynořování

Fáze zanořování (při volání funkce; vytvoření nové proměnné na zásobníku) Opakovaně!

Fáze vynořování
– při ukončení funkce
– uvolní se přidělený kus paměti na zásobníku (zaniknou lok. proměnné a parametry)
– do místa volání se předá výsledná funkční hodnota ! nezapomenu na RETURN !
– Rekurze se vynoří o jednu (!) úroveň

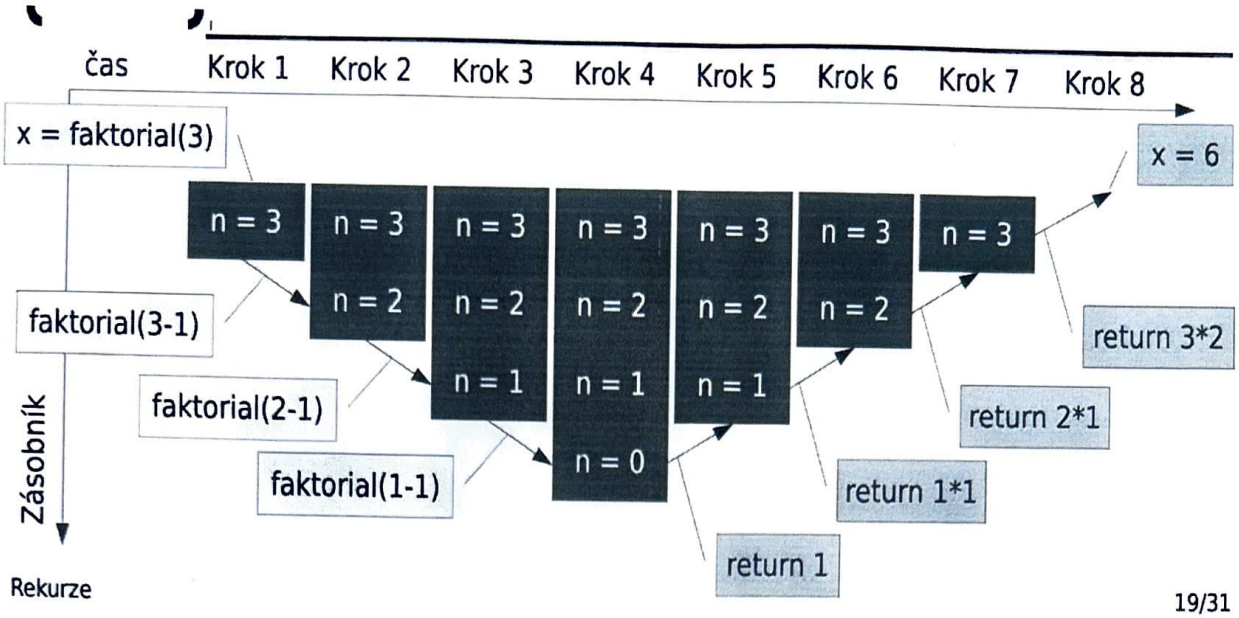
– Funkcionální jazyky takovou vytvoření na rekurzi
– Možné přidělení zásobníku
– Primitivní přístup ↳ proměnné
– Rekurzivní přístup ↳ volání funkce

! Každá rekurze jde případně do cyklu !

převedení rekurze na cyklus ⟹ explicitní zásobník
↳ konečná rekurze (jde) a když se ⟹ zásobník

volání všechny v sobě: homogenní věže, ukončená rekurze
↳ výsledek zůstane stejný bude
NEJDE zobrazit

– moderní OS má omezení na paměť, paměť jinak zabitá


čas
Krok 1
Krok 2
Krok 3
Krok 4
Krok 5
Krok 6
Krok 7
Krok 8
x = faktorial(3)
n = 3
n = 3
n = 2
n = 3
n = 2
n = 1
n = 3
n = 2
n = 1
n = 0
n = 3
n = 2
n = 1
n = 3
n = 2
n = 3
x = 6
faktorial(3-1)
faktorial(2-1)
faktorial(1-1)
return 1
return 1*1
return 2*1
return 3*2
Zásobník