Rekurze

Z FAV wiki

Rekurzí rozumíme sebevolání, tedy kód, který volá sám sebe či svojí část (přímá rekurze), případně volá kód, který posléze volá kód volající (nepřímá rekurze).

Rekurze je vhodná pro řešení rekurzivních problémů (například výpočet faktoriálu nebo Fibonacciho posloupnosti, fraktály), je však paměťově velmi náročný (každé rekurzivní volání vyžaduje vlastní paměťový prostor), doporučuje se tedy pouze pro vhodné problémy (rekurzivní výpis řetězce na obrazovku je nesmysl).

Rekurze musí mít následující podmínky:

- musí být konečná, tedy musí existovat vstup, pro který rekurzivní funkce nevolá sama sebe, tedy musí být testována tato konečnost před voláním sebe sama
- problém se musí rekurzí zjednodušovat, aby rekurze ke koncové podmínce dospěla (případně můžeme definovat maximální počet vnoření jako pojistku)

Typy rekurze

- **Přímá** - metoda volá sama sebe
- **Nepřímá** - metoda volá metodu, která volá zpětně ji

- **Lineární** - metoda volá pouze jednu svoji kopii (faktorial)
- **Stromová** - metoda volá více svých kopií (Fibonacci, průchod adresářovou strukturou, často Divide & Conquer algoritmy)

Rekurze však často umožňuje zjednodušení kódu a ušetření času při programování. Často lze rekurzivní problémy řešit také čistě iteračním algoritmem, nebo použitím zásobníku.

Příklad: Procházení adresářové struktury. Vytvoříme metodu ProjdiAdresář(string cesta);

Iteračně (nesmysl?):

1. Vytvoříme ukazatel na položku adresářové struktury ukazující "před?" výchozí adresář a uložíme tuto hodnotu do vedlejší proměnné
2. Posuneme ukazatel na další položku adresáře
3. Je-li pod ukazatelem označená položka, nebo prázdno, vrátíme se na předchozí položku a zpět na 2, nebo ukončíme, pokud jsme na uložené položce "před?"
4. Je-li pod ukazatelem adresář, označíme jej jako zpracovaný (i když je otázka jak) a posuneme ukazatel na jeho první položku
5. Je-li pod ukazatelem soubor, [otevře, vypíše název, zpracuje, přidá do vlastní datové struktury,...], a označíme soubor jako zpracovaný
6. Zpět na bod 2

Zásobníkem:

1. Načte obsah adresáře 'cesta' do zásobníku
2. Prochází položky zásobníku:
    1. Je - li položka soubor, [otevře, vypíše název, zpracuje, přidá do vlastní datové struktury,...]
    2. Je - li položka adresář, načte jeho obsah do zásobníku
3. Opakuje body 2 a 3 dokud není zásobník prázdný

Rekurzivně:

1. Načíst obsah adresáře 'cesta' a prochází jednotlivé položky
2. Narazí - li na soubor, [otevře, vypíše název, zpracuje, přidá do vlastní datové struktury,...]
3. Narazí - li na adresář, volá sama sebe na jeho cestu

Vidíme, že pro rekurzi neimplementujeme vlastní zásobník (viz však níže), a kód je přehlednější (Rekurze cca 5 řádek, iteračně mnohem více)

Pár slov k implementaci rekurze: V imperativním programování je většinou rekurze vnitřně řešena jak jinak než na zásobníku, jelikož je stav volající funkce uložen na vnitřní zásobník. Technicky vzato je tedy rekurze nahrazena zásobníkovým způsobem. Pri chybě rekurze (nesplnění koncové podmínky) většina jazyků vyhodí Stack Overflow (přetečení zásobníku) případně Access violation (Chyba přístupu do chráněné paměti), záleží na způsobu vnitřní implementace zásobníku.

// http://www.abclinuxu.cz/clanky/ruzne/komiks-xkcd-244-stolni-rpg pro ty, to ví co je Dungeons & Dragons

Příklady

Výpočet faktoriálu:

```
int faktorial(int f) {
  if (f <= 1)
    return 1;
  else
    return f * faktorial(f - 1);
}
```

Fibonacciho posloupnost:

```
int fibonacci(int n) {
  if (n <= 2)
   return 1;
 else
   return fibonacci(n - 1) + fibonacci(n - 2);
}
```


Kategorie: Fav-kiv-bzinf