II. Efektivita programu

- Efektivní progamy x čitelné programy
- Výkonnost hardware v současnosti převyšuje požadavky běžného software -> při vývoji SW je proto potřeba spíše dbát na efektivitu práce (čitelnost programu)
- Tlak na efektivitu programu u „malého“ levného hardware (malé jednodeskové PC, jednočipové mikropočítače)
- Znalost procesu kompilace a činnosti přeloženého programu napomáhá používání takových konstrukcí ve vyšším programovacím jazyce, které po přeložení pracují maximálně efektivně

6. Mechanismus přístupu k datům

- typy paměti používané programem pro ukládání dat (statická paměť, zásobník, halda)
- lokální proměnné (zásobník)
  - *adresa vcholu zásobníku* + *offset*
- globální proměnné (statická paměť)
  - *adresa ve statické paměti*
  - existují programy pouze s glob. proměnnými (bez lokálních a bez parametrů funkcí)
- registrové proměnné (procesor)
- jedno- a vícerozměrné pole
  - velikost známá při překladu – umístění (zásobník x halda)
  - *adresa pole* + *index* * *velikost prvku*
  - *adresa pole* + *index1* * *velikost řádku* + *index2* * *velikost prvku*
- speciální třídy (Vector, ArrayList, HashMap, Hashtable, String)
  - pole polí, hashovací metoda (objekt -> index)
- struktury, třídy, volání metod, virtuální metody
  - *adresa struktury* + *offset*
- halda (heap) (spousta různých implementací, většinou pomocí zřetězených seznamů)
  - objekty na haldě jsou referencovány pointery
  - operace alokování prostoru (včetně vyhledání ideálního volného prostoru)
  - operace uvolnění prostoru (včetně scelování)
  - operace „setřepání“ - nelze v každém prog. Jazyce
  - některé jazyky hlídají, jestli je prostor na haldě referencován
  - garbage collector (hledá nereferencované objekty na haldě, případně volá destruktor)
  - některé jazyky se bez haldy neobejdou (Java): výkonnost
- typ množina
  - s výčtem prvků známým při kompilaci
  - dynamická (HashMap)

7. Implementace programových struktur

- mechanismus volání funkce

<dno zásobníku>
...
Lokální proměné volající funkce
Parametry volané funkce
Návratová adresa
Lokální proměnné volané funkce
<vrchol zásobníku>

- parametry funkcí
- rekurzivní funkce
- for-cyklus

```
for (I = 0; I < 10; I++) {opakovaný kód }
```

Zkompiluje se jako:

```
I = 0
začátek cyklu:
if (I >= 10) goto konec cyklu

opakovaný kód

I++
Goto začátek cyklu
Konec cyklu:
```

- vícenásobné větvení (switch)
  - po sobě jdoucí hodnoty: lookup table adres
  - jinak lookup table hodnot + adres
  - hodnoty nejsou známy při kompilaci: kompiluje se jako posloupnost if

8. Rozdíl v interpretovaných a překládaných jazycích

- překládané jazyky – rychlejší, typová omezení (v dobrém slova smyslu)
- interpretované jazyky – pomalejší, typová volnost, volání funkcí nebo odkaz na proměnnou názvem
- JIT (v Javě od 1.3 – cca 6x rychlejší, od 1.4 – cca 12x rychlejší, v .NET od počátku)