17. Základní datové typy, datové struktury

Význam datových typů a struktur; dynamický typový systém, statický typový systém

Datový typ:

- Definuje druh nebo význam hodnot, kterých smí nabývat proměnná
- Datový typ je určen oborem hodnot a zároveň výpočetními operacemi
- Např. Integer - Jakých hodnot nabývá? Čísla, až 2^32, každý jazyk to má ale jinak
- Jaké operace s ním můžeme provádět? Sčítání a odčítání, násobení a dělení atd.

Datová struktura:

- Umožňuje uchovávat a zpracovávat množinu dat stejného typu nebo různorodých, ale logicky souvisejících dat
- Tyto data lze reprezentovat různými datovými typy

Dynamický typový systém

- Především v OOP programovacích jazycích
- Většinový přístup je takový, že množina zpráv, které daná skupina objektů rozumí, je určena pomocí třídy, jež definuje jejich sdílené chování
- Objekty se chovají jako uzavřené entity

Statický typový systém

- U každé deklarované proměnné či parametru je nutné uvádět typ
- Typ pak definuje množinu hodnot, kterých může proměnná nabývat, a množinu operací, které mohou být s danou proměnnou provedeny
- V každém okamžiku pak musí být naprosto jasné, jakého typu je proměnná, s níž právě pracujeme
- Statická typová kontrola se vyznačuje velikou bezpečností vygenerovaných programů
- Již v době překladu můžeme prohlásit, že výsledná aplikace nemá žádnou typovou chybu, tedy že se nesnažíme pracovat s daty, která mají nepovolený rozsah hodnot a ani se nad nimi nesnažíme vykonávat nepovolené operace
- Máme definován celočíselný typ s rozsahem 0 až 255. Statická typová kontrola nám v tom případě samozřejmě zakáže napsat:
- var = 256;
- Ovšem většinou už jí nebude vadit:
- var = 255;
- var = + 1;

Základní rozdělení datových typů

Ordinální datové typy

- **Ordinální datové typy** můžeme velikostně uspořádat, můžeme jednotlivé entity tohoto typu mezi sebou porovnat a určit jaký je větší/menší
- **Logická hodnota:** Typ boolean, který smí nabývat hodnot true nebo false. 1 = true, 0 = false

- **Celé číslo**: Ve většině jazyků mají celá čísla omezený rozsah. Pokud je celé znaménkové číslo omezeno například na 16 bitů, tak bude mít rozsah -32768 až +32767, což je dané kódováním ve dvojkovém doplňkovém kódu (kódování ve dvojkové soustavě)
- **Znak:** Pro znak se obvykle používá označení char – např. '%'. Ve skutečnosti je znak v počítači reprezentován pomocí celého čísla. Pro kódování znaků se většinou používá znaková sada ASCII a její národní rozšíření, nebo znaková sada Unicode.

Neordinální datové typy

- Např. Nemůžeme říct, že pravda je větší/menší než lež
- **Reálné číslo:** Reálná čísla, která lze vyjádřit nekonečně dlouhým desetinným rozvojem jsou představována desetinnými čísly.
- Double, float a real – Real může obsahovat hodnotu o velikosti 4 bajtů nebo také 7 míst za desetinou čárkou, zatímco float až 15 desetinných míst nebo také 8 bajtů. Double je podobný jako float ale umožňují mnohem větší čísla.

Prázdný datový typ

- **Void:** Tento typ nenabývá žádných hodnot, může sloužit pro deklaraci funkce, která nemá návratovou hodnotu.
- V některých jazycích existuje rovněž prázdná hodnota ošetřující neplatný výsledek **– null**

Složené datové typy

- **Pole:** Array, může být vícerozměrné
- **Textový řetězec:** String, uložení konečné posloupnosti čísel
- **Seznam:** List, obdoba pole, na rozdíl od pole nelze seznam přímo adresovat pomocí indexu.
- Seznam je tedy možné procházet pouze postupně, od začátku do konce, sekvenčně.
- Výhodou seznamů proti polím je, že je možné snadno přidávat nebo odebírat i prvky nacházející se uprostřed seznamu.

Abstraktní datové typy

FIFO – First In First Out

- Je to např. Fronta na oběd, je to taková struktura seznamu, kde první prvek opouští danou strukturu, není možné předbíhat.
- Prvky vkládáme na jeden konec seznamu a z druhého konce odchází.
- Využívá se tehdy, pokud potřebujeme dočasně uložit nějaké údaje a později je průběžně zpracovat a záleží nám na jejich pořadí.

LIFO – Last in First Out

- Je to např. Zásobník, přesný opak FIFO
- Např. Pokud máme zboží, prvně vyskladňujeme zboží nejnovější, které přišlo jako poslední

STROM

- Představuje stromovou strukturu s propojenými uzly
- **Uzel stromu může obsahovat:**
- Hodnotu, podmínku, reprezentovat strukturu oddělených dat, vlastní strom
- Uzly jsou navzájem spojeny hranami. Neexistuje osamocený uzel, ke kterému by nevedla žádná hrana (s výjimkou stromu s pouze jedním uzlem).

Kořen stromu, Uzel a Listy

- **Listy –** Uzly bez následníka. Není k nim připojen žádný podstrom.

- **Kořen –** Uzel bez předchůdce = kořen. Existuje právě 1.
- **Vnitřní uzly –** Uzly, které nejsou listem ani kořenem.
- Patří mezi rekurzivní datové struktury: Každý uzel je současně kořenem stromu a zároveň listem stromu vyšší úrovně.
- Pro každý uzel platí, že všechny údaje v levém podstromu jsou menší než U a všechny údaje v pravém podstromu větší než U.

Využití v praxi

- Stromy, a zejména jejich některé konkrétní vyhledávací varianty, nacházejí široké uplatnění v oblastech, kde je třeba řešit ukládání a vyhledávání dat, zejména tam, kde je kritickou omezující podmínkou vyhledání dat s co nejmenší úrovní složitostí a při co nejméně přístupů čtení.
- Pravděpodobně nejpoužívanější v praxi jsou aplikace B+ stromů, kde nejčastější použití je u souborových systémů a většiny databází.

Datové struktury v OOP

- Objekt si “pamatuje” svůj stav (v podobě dat čili atributů) a zveřejněním některých svých operací (metod) poskytuje rozhraní, jak s ním pracovat.
- Při používání objektu nás zajímá, jaké operace (služby) poskytuje, ale ne, jakým způsobem to provádí – princip zapouzdření.
- Jestli to provádí sám nebo využije služeb jiných objektů, je celkem jedno.
- Vlastní implementaci pak můžeme změnit (např. zefektivnit), aniž by se to dotklo všech, kteří objekt používají.