

# Datové typy strukturovaných jazyků

Datový typ definuje v programování druh nebo význam hodnot, kterých smí nabývat proměnná (nebo konstanta). Datový typ je určen oborem hodnot a zároveň výpočetními operacemi, které lze s hodnotami tohoto typu provádět

## Jednoduché datové typy:

Jednoduché (také elementární) datové typy jsou většinou definované přímo jazykem, jsou do něj zabudované, můžeme je dělit na ordinální a reálné.

### Ordinální typy

Hodnoty ordinálního typu tvoří lineárně uspořádanou množinu, kde pro každý prvek je přesně definovaný **předchůdce** i **následovník** (z posledního prvku ve většině jazyků dochází k tzv. přetečení na první).

**Integer** - nejtypičtější ordinální datový typ. Jedná se o celé číslo - můžeme tedy jasně definovat předchůdce (číslo o 1 menší) a následovníka (číslo o 1 větší). Integer má omezený rozsah, např. pokud je omezený na 8 bitů má rozsah -128 až 127 (Pokud k 8 bitovému integeru s hodnotou 127 přičteme 1, dojde k přetečení a získáme číslo -128)

V jazyce C existují tyto konkrétní typy:

- *int* = celé číslo, 4 byty
- *short* = celé číslo, 2 byty
- *long* = ještě více než *int*

**Char** - neboli znak = jednotka informace, která zpravidla odpovídá jednomu znaku v psané formě přirozeného jazyka. Tedy jednomu písmenku, číslici, symbolu, interpunkci,... např. 'a', 'A', 'g', ' ', '5', '%'.

**Výčtový typ** - jedná se o datový typ definovaný uživatelem - může nabývat hodnot ze specifikované konečné množiny. Např. měsíce v roce (leden, únor...), dny v týdnu (pondělí, úterý...)

v C ho definujeme takto:

```
typedef enum {
```

```
PRAHA, BRNO, OSTRAVA
```

```
} MESTA;
```

### Neordinální typy

U neordinálních datových typů není jednoznačně určen předchůdce a následovník každé hodnoty.

**Double, float, real** - reálné číslo nebo také číslo s plovoucí desetinnou čárkou, např. '1.58' nebo '0.0045'. V počítači je většinou reprezentováno jako *celé číslo* \*  $2^{\text{exponent}}$  kde exponent je také celé číslo. Mnohá desetinná čísla nelze v tomto formátu přesně reprezentovat. Např. číslo 0,1 má periodický dvojkový zápis  $(0,0001100)_2$ . Důsledkem je, že reálná čísla můžou v počítači vypadat a chovat se trochu jinak.

V jazyce C existují tyto konkrétní typy:

- *float* = reálné číslo, 4 byty
- *double* = reálné číslo se zvýšenou přesností, 8 bytů
- *long double* = ještě více zvýšená přesnost, 12 bytů

## Prázdný datový typ

**Void** - Tento typ nenabývá žádných hodnot, může sloužit např. pro deklaraci funkce, která nemá návratovou hodnotu, nebo označuje data nespécifikovaného typu.

## Složené typy:

Složené datové typy obsahují jeden nebo více prvků (např. 5 integerů). Můžeme říci, že jsou homogenní, když se skládají z prvků stejného typu, jinak jsou heterogenní.

**Pole - array** -  $\{[0] \Rightarrow 3; [1] \Rightarrow -2; [2] \Rightarrow 255\}$ , může být vícerozměrné (např. dvourozměrné označujeme jako matici). Jednotlivé prvky pole jsou dostupné pod číslem, které určuje jejich pořadí (tzv. index - v []). Nejčastější je konvence používaná v C-jazyce, kde indexy začínají číslem 0. Hlavní výhodou pole je možnost okamžitého přístupu ke kterékoli jeho položce přes index. Pole obsahuje proměnné stejného typu, takže je homogenní.

Velmi podobný je **textový řetězec - string**. Ke každému znaku se dostaneme podle čísla jeho pořadí. Např. string „liška“ -  $[2] \Rightarrow „š“$ ;  $[3] \Rightarrow „k“$ .

**Struktura - struct** - datový typ složený z jiných datových typů/složený z různých typů proměnných. Jedná se tedy o heterogenní datový typ. Nejlépe je pochopitelný přímo z ukázky deklarace v jazyce C:

```
struct account {
```

```
    int account_number;  
    char *first_name;  
    char *last_name;  
    float balance;
```

```
};
```

Když teď vytvoříme nový struct: 'struct account accountExample;' vytvoříme v podstatě „balíček“ výše deklarovaných proměnných, `account_number` tohoto konkrétního structu poté získáme např. 'accountExample.account\_number ...'

Speciální případ jsou **ukazatele - pointery** - jedná se o datový typ uchovávající adresu ukazující na nějaké místo do paměti počítače (ukazují tedy např. na nějakou jinou proměnnou, nebo na určitý

prvek pole)

# Staticky a dynamicky typované jazyky (+ silné a slabé typování)

Typová kontrola je v informatice proces ověřování datových typů ve zdrojovém kódu počítačového programu (pro operandy ve výrazech, parametry a návratové hodnoty funkcí i jinde). Může být prováděna buď překladačem při překladu programu (statická typová kontrola) nebo až běhovým prostředím za běhu programu (dynamická typová kontrola). Programovací jazyky, které vyžadují přísné uvádění datových typů, označujeme za **silně typované**, tyto jazyky provádí kontrolu za běhu programu a zjišťují jejich chybné použití. Jazyky neprovádějící kontrolu typů za běhu jsou **slabě typované**.

**Statická typová kontrola** ověřuje typy výrazů, zda odpovídají svým kontextům - např. požadavek na typ operandu určitého operátoru apod. Typová kontrola probíhá již v době kompilace - odhaluje pouze chyby vyhodnotitelné při kompilaci. Vynechání typové kontroly za běhu umožňuje efektivnější vykonání programu (rychlé či s menším nárokem na paměť) a další optimalizace. Při programování ve staticky typovaných jazycích je nutno explicitně deklarovat typy, v OOP obvykle typy proměnných. Mezi staticky neboli silně typované jazyky patří například C, C++, C#, Java.

**Dynamická typová kontrola** probíhá za běhu programu. Jazyky používající dynamické typování nepožadují specifikaci datového typu u proměnných a ty mohou tudíž odkazovat na hodnotu jakéhokoli typu. Mezi dynamicky neboli slabě typované jazyky patří JavaScript, PHP, Python, Ruby. Dynamické typování je pružnější, protože programy generují typy a funkcionalitu na základě běhových dat, ale také častější na běhové chyby, kdy hodnota nabude neočekávaného typu a ten se použije pro další zpracování.

From:

<http://wiki.gml.cz/> - GMLWiki

Permanent link:

<http://wiki.gml.cz/informatika:maturita:20a?rev=1428962781>

Last update: **14. 04. 2015, 00.06**

