**Typy**

**Proměnná**

pojmenované je to místo v paměti počítače, kam si můžeme uložit nějaká data.

Toto místo má podle typu proměnné také vyhrazenou určitou velikost, kterou proměnná nesmí přesáhnout.

Proměnná má vždy nějaký **datový typ**

Většinou musíme před prací s proměnnou tuto proměnnou nejdříve tzv. deklarovat (pojmenovat) a jakého datového typu bude. Jazyk ji v paměti založí a teprve potom s ní můžeme pracovat.

**Konstanta**

Podobná proměnné, ale nelze ji v průběhu programu změnit

**Datový typ**

Definuje velikost místa v paměti a způsob uložení informací

Určuje hodnoty, které proměnné a konstanta může obsahovat a také operace, které můžou být prováděny

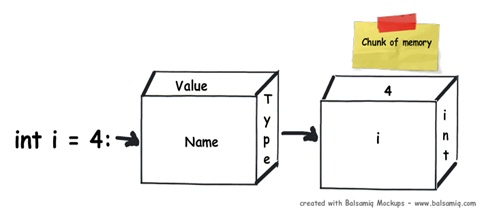
O každém datovém typu jazyk ví, kolik v paměti zabírá místa a jak s tímto kusem paměti pracovat.

Typový systém

* Dynamický: nepíše se k proměnné datový typ, vnitřně ho má. Proměnné se nemusí deklarovat, jakmile do nějaké proměnné něco uložíme a jazyk zjistí, že nebyla nikdy deklarována, sám ji založí. Do stejné datové proměnné zle ukládat různé hodnoty datových typů (čísla, text, objekty), jazyk sám mění datový typ proměnné. Výhody: méně kódu. Např: Python a Ruby



* Statický: vyžaduje datový typ proměnné a tento datový typ je neměnný. Jakmile proměnnou jednou deklarujeme, není možné její datový typ změnit. Např: C# a Java Výhoda oproti dynamickému: kód je možno kontrolovat, c# nedovolí kód ani zkompilovat a na chybu upozorní, u dynamického se na to přijde až v běhu programu



**Rozdělení proměnných**

* Globální proměnné se vytvoří při spuštění programu a existují po celou dobu jeho běhu. To znamená, že mají v programu stálou adresu (např. atributy tříd nebo proměnné static).
* Lokální proměnné se vytvářejí během běhu programu v závislosti na zdrojovém kódu a existují do konce bloku, ve kterém jsou deklarovány. Jsou to všechny vstupní parametry metod. Vznikají v okamžiku volání funkce, při ukončení funkce zanikají. Vznikají v okamžiku deklarace a zanikají při ukončení funkce (např. funkce Main). Proměnné deklarované uvnitř cyklu nebo podmíněného příkazu zanikají po cyklu
* Dynamické proměnné vznikají za běhu programu na základě příkazu new. Prostor pro ně čerpá program z volné paměti. O jejich zánik se stará Garbage Collector nebo jsou rušeny voláním příkazu pro zrušení.

**Rozdělení podle způsobu deklarace**

Předdefinované – jsou součástí syntaxe daného programovacího jazyka (int,long,bool)

Definované uživatelem – uživatel musí sám deklarovat (enum,pole)

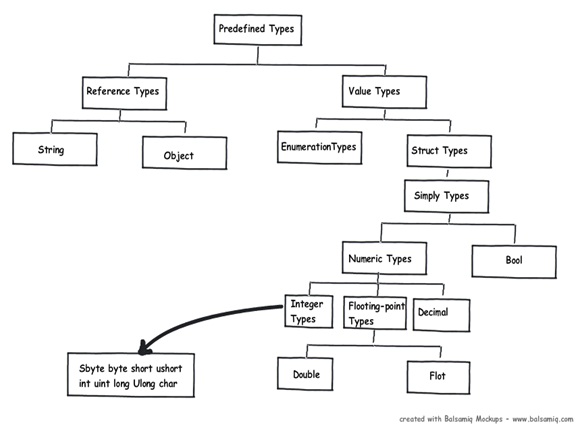
**Rozdělení podle počtu hodnot**

Jednoduché – obsahují jednu hodnotu

Strukturované – obsahují více hodnot

**Druhy datových typů**

Hodnotové (value) a referenční (reference)



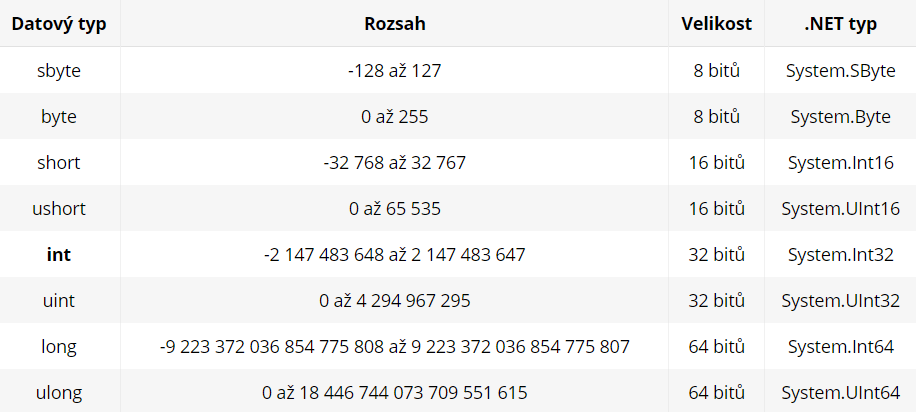
**Hodnotové**

do této skupiny patří všechny jednoduché datové typy i některé datové typy strukturované (struct,enum).

U těchto jednoduchých typů se hodnota proměnných ukládá přímo do stacku.

Pokud přiřadíme proměnnou hodnotového typu jiné proměnné hodnotového typu, zkopíruje se obsah zdrojové proměnné do cílové proměnné a vytvoří se úplná kopie instance.

**Celočíselné datové typy (integer types)**



Záleží na jeho velikosti

Čím větší číslo, tím více spotřebuje paměti.

Když začíná název typu na “u“, tak nemůže obsahovat záporné čísla, takže kladnou část mohou uložit 2x vyšší hodnotu, říká se jim “unsigned“, ostatním “signed“. Protože ten znaménkový bit je použit místo znaménka hodnotu čísla..

Sbyte:

-128 = 10000000

127 = 01111111

Byte:

127 = 01111111

128 = 10000000

255 = 11111111

= 7 bitů na hodnotu, 1. Je znaménkový, záporné vznikne: negace kladného, přičtení 1

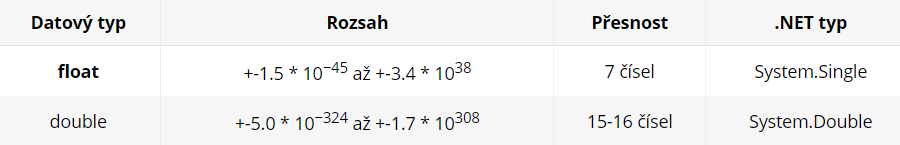
Int64 vs long – není rozdíl, hodnotové typy je zvyk pojmenovat malým písmenem. .NET nabízí aliasy těchto typů



Nejde protože, výsledek sčítání je 32bitový

 nebo 

#### Desetinná čísla (floating point a decimal)



Float – 32bit, double dvojnásobně co float, takže 64bit

Decimal – 128bit, vnitřně pracující v desítkové soustavě, dobrý na finance, zaokrouhluje přesně, ale pracuje pomaleji

\*\*Vzhledem k tomu, že desetinná čísla jsou v počítači uložena ve dvojkové soustavě, dochází k určité ztrátě přesnosti. Odchylka je sice téměř zanedbatelná…

Například nepoužívat u finačního systému.



U float se musí použít sufix F, u double ne (jeho výchoží typ je desetinný)

**Znak (Char) a logický (bool)**



Char

reprezentuje jeden znak, uložený v unicode znaku

C# chars are Unicode, which is 16-bit, while C only uses ASCII, which is actually only 7 bits.

Bool

True nebo false

Je 8 bitový, protože CPU nemůže adresovat nic menšího něž bajt (8 bitů)

Tyto datové typy se dají vypsat



Protože C# provede implicitní konverzi a zavolá na čísle metodu ToString().

**Referenční**

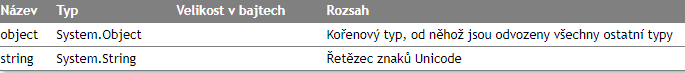
do této skupiny patří datový typ System.String a všechny třídy.

Na rozdíl od proměnných hodnotových datových typů se jejich hodnota uloží do haldy.

Do proměnné v stacku se uloží pouze reference - adresa paměti na haldě, kde je hodnota uložena.

Pokud přiřadíme proměnnou referenčního typu jiné proměnné referenčního typu, bude výsledkem nový odkaz na stejnou adresu na haldě.

String, objekty, třídy, rozhraní, pole, delegáty

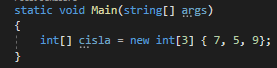


String je uložen na haldě a obsahuje adresu na pole charů

Obsahuje hodně metod: Contains(), Replace()

Má neomezenou délku

Třeba příklad pole



V zásobníku je prostor 32bit pro adresu, která ukazuje na pole na haldě

Na haldě je 3\*32bit prostor pro uložení čísel 7,5,9

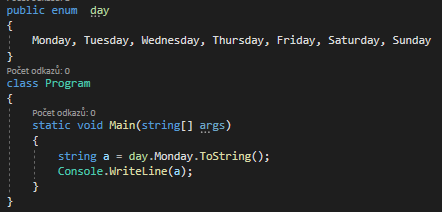
**Výčtové datové typy – enum**

Je uložen na stacku.

Je tvořen seznamem přípustných hodnot (pevně dané možností, které se nemění)

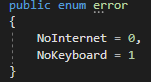
Lze dvěma položkám přiřadit stejnou hodnotu a C# si potom vybere jednu hodnotu jako primární (z důvodů reflexe a řetězcových konverzí). Výčtové typy mají následující omezení:

* nemohou definovat své vlastní metody
* nemohou implementovat rozhraní
* nemohou definovat své indexery nebo vlastnosti



Enum primárně implentuje int, dá se to změnit na short, long, celočíselné datové typy

Příklad



**Struktura**

Strukturovaný (má více položek)

heterogenní (položky mohou být různého typu)

hodnotový (ukládán na zásobník)

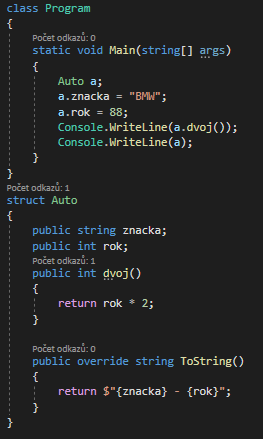
je uživatelem definovaný datový typ.

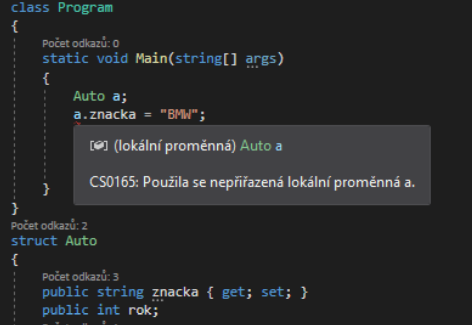
struktura může obsahovat i deklaraci metod.

Strukturu můžeme deklarovat jako součást jmenného prostoru nebo jako součást třídy. Strukturu nelze deklarovat uvnitř metody.

Je také možné deklarovat konstruktor a vytvářet proměnnou jeho voláním pomocí klíčového slova new.

Nemůžou dědit od jiných tříd, ani být děděny.





Takto se nepoužívají, protože stack má omezenou velikost (64KB) a struktury ho lehce zaplní.

**Generika**

* obecný datový typ
* generický kód – ten, který využívá parametrizované typy, Parametrizované typy jsou nahrazeny konkrétními typy v době použití kódu

uživatel má volnost zvolit si libovolný datový typ

Generický typ T se uvádí za název definovaného objektu (třídy, rozhraní, metody, struktury, delegáty)

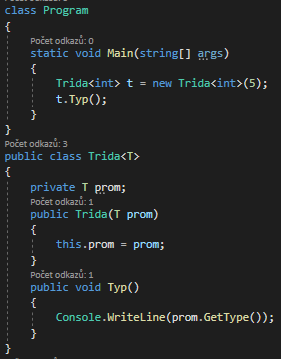
Výhody (související s řádkem nad):

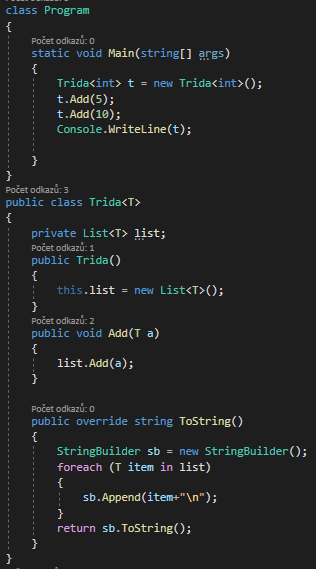
V prvé řadě dosáhneme silné typové kontroly v době překladu zdrojového kódu, a tedy možného nalezení všech případů, které by mohly způsobit havárii aplikace způsobenou nesprávným využitím použitých typů.

Omezení explicitních typových konverzí, které jsou nutné v době překladu, kdy neznáme typ objektu uloženého v univerzální kolekci.

Díky těmto výhodám dosáhneme nejen čistějšího, rychlejšího výsledného kódu, ale především náš kód bude hlavně bezpečnější.

používá se třeba při vstupních proměnných do metod, nebo při vytváření generických kolekcí





Obecně nezáleží, jaké písmeno do špičatých závorek zvolíme. Nejčastěji se používá T





**Ještě nějaké datové typy**

Dynamic

hodí se v situacích, kdy se neví datový typ proměnné

V proměnné tak může být naprosto cokoli a kdykoli se může měnit i její datový typ.

je typově dynamická



Var

Psát datové typy je občas zbytečné, např. v momentě, kdy víme, jaký datový typ dosazujeme

Nemůže měnit datový typ.

Je doporučená jen tehdy, pokud bude i s ní kód přehlednej

