DUM č. 5 v sadě

28. Inf-4 Jednoduchá hra Had ve Flashi (ActionScript)

Autor: Robert Havlásek

Datum: 26.02.2013

Ročník: 5AV

Anotace DUMu: Flash - teorie: Jednoduchá podmínka. Větvení s více možnostmi.

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Jednoduchá podmínka

Používá se v případech, kdy podle hodnoty nějaké podmínky (přesněji: podle hodnoty výrazu typu Boolean) provedeme či neprovedeme nějaký kus kódu.
Používá se zápis:
`if (podmínka) {blok_příkazů}`, případně `if (podmínka) příkaz`
Závorky okolo podmínky či okolo složitějšího výrazu typu Boolean zapsaného za if jsou nezbytně nutné, jinak Flash křičí syntaktickou chybu.

Typicky například:
```
if (i<0) trace("zaporne cislo");
```
nebo
```
absolutni=i;
podminka=absolutni<0;
if (podminka) {absolutni=-absolutni;
               trace("Otocili jsme znamenko…");
               }
```
Druhý příklad používá prom. `absolutni` a `i` typu Number a prom. `podminka` typu Boolean.
Pozn.: Osobně bych v druhém řádku psal spíše `podminka=(absolutni<0);` *, je to čitelnější.*

Porovnání čísel, řetězců pomocí operátorů

Operátory, které lze pro porovnání hodnot dvou proměnných používat, jsou:

`<` např. `0<3` odpovídá logické hodnotě `true`

`<=` např. `4<=4` odpovídá `true`

`>` např. `'3'>12` odpovídá `false` (řetězec se převede na číslo 3)

`>=` např. `'3'>='12'` odpovídá `true` (dva řetězce se porovnávají lexikograficky, podle prvních znaků, podobně jako 'Bedřich' >= 'Adam' nebo třeba 'Bedřich'>='Bartoloměj')

`==` např. `4=='4'` odpovídá `true` (převod řetězce na číslo je u operátoru == povolen)

`!=` např. `(4>3) != false` odpovídá `true`, ve vnitřních závorkách je pravda, která se vážně nerovná nepravdě.

`===` např. `4==='4'` odpovídá `false` (převody se nepovolují, kontrolují se i hodnoty i typy)

`!==` např. `5!==7` odpovídá `true`; při kontrole hodnot i typů se zjistilo, že neodpovídají hodnoty, proto neplatí `5===7`, tudíž platí `5!==7`. Podobně `5!=='5'` odpovídá `true`.

Pedagogická poznámka: Uvádíme zde pouze porovnávací operátory, ostatní (např. početní) vůbec neučím, pro studenty „vyplynou ze situace“, případně je vysvětlím, až je potřebujeme.

Po výkladu porovnávacích operátorů udělám na dataprojektoru společný příklad:
```
i=7-7;
if (i=0) {trace ("Vyšlo nulové číslo.");}
```
Nefunguje. Otázka: Co je na tomhle příkladu špatně?
Správná odpověď: Operátor = neznamená porovnání, ale přiřazení. Pro správnou funkci musíme napsat `if (i==0)`… Vedlejším efektem operátoru = je, že vrátí přiřazenou hodnotu, tedy závorka `(i=0)` má hodnotu `0`, což se převede na logickou hodnotu `false`. Pokud bychom měli třeba závorku `(i=3)` (resp. libovolné jiné číslo kromě nuly), převedla by se na `true`.
V kontextu přiřazení funguje dokonce i=j=k=5 (všem proměnným nastaví hodnotu 5).

Spojování logických výrazů pomocí operátorů

Operátory, které lze při spojování více logických hodnot používat, jsou:
`&&` též `and` např. `(0<3) and (true)` odpovídá logické hodnotě `true` (platí oboje zaráz)
`||` též `or` např. `(0<3) || (0>3)` odpovídá logické hodnotě `true` (platí aspoň jeden)
`!` též `not` např. !(0==3) odpovídá true.
Poznámka: V případě podmíněného příkazu je nutná závorka i před negujícím vykřičníkem, tedy `if !(x==0)…` *fungovat nebude, musí být* `if (!(x==3))….`

Podmínka s dvěma větvemi

Používá se zápis:
`if (`*`podmínka`*`) {`*`blok_příkazů`*`} else {`*`blok_příkazů`*`}`,
místo kteréhokoliv bloku příkazů může být jeden příkaz, který není nutné uzavírat do `{}`.

Typicky například:
```
if (i<0) trace("zaporne cislo")
    else trace("nezaporne cislo");
```

Namísto bloku příkazů je možné psát opět podmínku, například:
```
if (i<0) trace("zaporne cislo");
    else if (i>0) trace("kladne cislo");
             else trace("nulove cislo");
```

Pozn.: Středníky za jednotlivými řádky nejsou nutné, ostatně jako nikde ve Flashi.
Pozn.: Studenti, povšimněte si, jak píšu jednotlivé části podmínek (logicky) pod sebe.

Větvení s více možnostmi

V případě, že jednu proměnnou (nebo výraz daného typu) potřebujeme porovnat s více variantami (jako bychom vícekrát používali ==), můžeme místo množství podmínek napsat jedno větvení, které má zápis:
```
switch (výraz) {
  case hodnota: {blok_příkazů}
  case hodnota2: {blok_příkazů2}
```
atd.
```
  default: {blok_příkazů_pro_výše_nejmenované_možnosti}
  }
```
Pokud v bloku příkazů nepoužijeme jako poslední příkaz `break`, pokračuje Flash příkazy v dalším bloku (koncem větve `case` provádění neskončí).
Potřebujeme-li zapsat pro víc alternativ stejný blok příkazů, napíšeme prostě:
```
switch (výraz) {
  …
  case alt1:
  case alt2:
  case alt3: {blok_příkazů}
  …
  }
```

Praktický úkol: Zjistěte (buď změnami systémového data a výpisem nebo dohledáním přes F1 v manuálu), jaká čísla pro různé dny vrací `datum = new Date();` `trace(datum.getDay())` *a vypište buď „všední den“ nebo „sobota“ nebo „neděle“.*

Řešení:

```
datum = new Date();
switch (datum.getDay()) {
  case 0: trace("neděle"); break;
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: trace("všední den"); break;
  case 6: trace("sobota"); break;  // tento break zde byt nemusi
  }
```

Elegantnější řešení:

```
datum = new Date();
switch (datum.getDay()) {
  case 0: trace("neděle"); break;
  case 6: trace("sobota"); break;
  default: trace("všední den");
  }
```

Toto řešení předpokládá „kvalitní vstup“, neřeší případy jiných čísel než 0, 1, 2, 3, 4 a 5 (je to pro ně taky všední den). Zatímco řešení z předchozí strany nám na jiné případy než 0, 1, 2, 3, 4 a 5 odpoví mlčením.

Zde si můžeme „kvalitní vstup“ dovolit předpokládat (Flash žádnou jinou hodnotou opravdu neodpoví), obecně je ale vhodnější nekorektní varianty zadání separovat zvlášť, např. mlčením.