Podmienky `else-if`
Cykly `for`, `while`, `do-while`

Základy procedurálneho programovania 1, 2020

Ing. Marek Galinski, PhD.

Opakovanie

2

Celočíselné konštanty

- **Môžeme ich zapísať v rôznych číselných sústavách**

- **Desiatková** (dekadická) `15` `0` `1`
  - Zapisuje sa ako postupnosť číslic, na začiatku však nesmie byť nula (okrem hodnoty nula)
- **Osmičková** (oktalová) `017` `00` `01`
  - Číslica 0 nasledovaná osmičkovými číslicami (0-7)
- **Šestnástková** (hexadecimálna) `0xF` `0XF` `0x0` `0X1`
  - Reťazec 0x alebo 0X (znak nula) nasledovaný hexadecimálnymi číslicami (0-9,a-f,A-F)

Formátovanie: výpis čísel

```
float pi = 3.1415;

printf = "Hodnota Pi je: %.2f", pi);
>> Hodnota Pi je: 3.14

printf = "Hodnota Pi je: %g", pi);
>> Hodnota Pi je: 3.1415
```


4

Špeciálne unárne operátory

- **Inkrement ++**
- **Dekrement --**

```
i++;
j--;
```

- **Prefix zápis - ++hodnota**
  - **Inkrementuje sa pred použitím**

```
++i;
--j;
```

- **Postfix zápis – hodnota--**
  - **Dekrementuje sa až po použití**

```
i--;
j++;
```


5

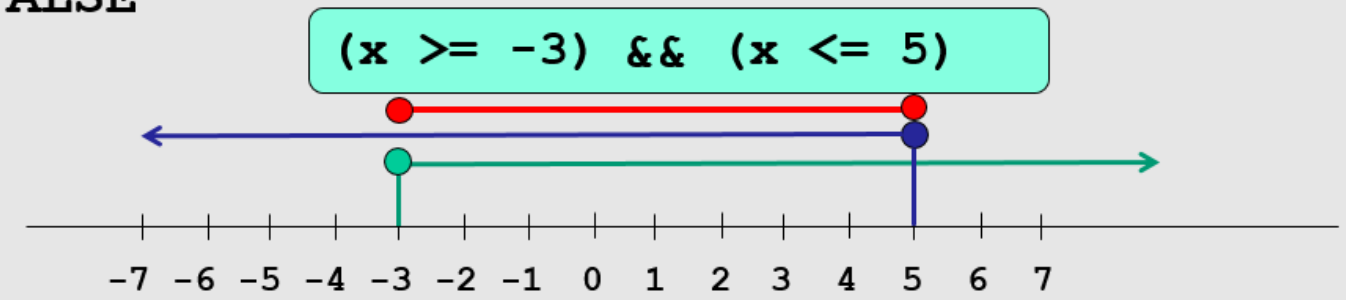
Logické operátory

```
int x = 10, y = 5;

(x == 10) // 1 (TRUE)
!(y < x)  // 0 (FALSE)
(x != 10) // 0 (FALSE)

(y <= x) && (y > 2)   // 1 (TRUE)
(x < 10) || (y == 20) // 0 (FALSE)
```

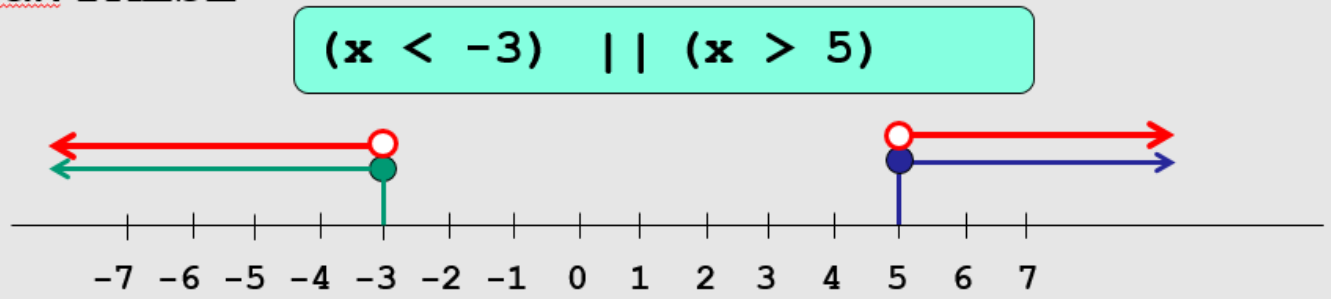

6

Zložené logické výrazy

- Podmienka je **TRUE**, ak číslo **x** je z intervalu <-3, 5>, inak **FALSE**



- Podmienka je **TRUE**, ak číslo **x nie** je z intervalu <-3, 5>, inak **FALSE**



7

Vyhodnocovanie logických výrazov

- **Výrazy sa vyhodnocujú zľava doprava**

```
A && B
```

  - **Ak A je TRUE, musíme zistiť či aj B je TRUE**
  - **Ak A je FALSE, nie je potrebné ďalej vyhodnocovať**

```
A || B
```

  - **Ak A je FALSE, musíme zistiť, či B je TRUE**
  - **Ak A je TRUE, nie je potrebné ďalej vyhodnocovať**

8

Vyhodnocovanie výrazov

9

Vyhodnocovanie výrazov

- **Čo platí a čo neplatí?**

```
int i = 1, j = 1, k;
```

```
k = i && (j == 3);
```

```
k = !i && j;
```

```
k = (i = 2) || j;
```

Vyhodnocovanie výrazov

- **Čo platí a čo neplatí?**

```
int i = 1, j = 1, k;
```

```
k = i && (j == 3);
```
NEPLATÍ, k bude 0

```
k = !i && j;
```
NEPLATÍ, k bude 0

```
k = (i = 2) || j;
```
PLATÍ, k bude 1

Vyhodnocovanie výrazov

- **Čo platí a čo neplatí?**

```
int i = 1, j = 1, k;
```

```
k = i && (j == 3);
```

NEPLATÍ, k bude 0

```
k = !i && j;
```

NEPLATÍ, k bude 0

SKRÁTENÉ VYHODNOCOVANIE

```
k = (i = 2) || j;
```

PLATÍ, k bude 1

SKRÁTENÉ VYHODNOCOVANIE

Vyhodnocovanie výrazov

- **Čo platí a čo neplatí?**

```
int i = 1, j = 1, k;
```

```
k = !i || (j = j - 1);
```

```
k = i || (j = j - 1);
```

```
k = i >= 0 && --i % 2;
```

Vyhodnocovanie výrazov

- **Čo platí a čo neplatí?**

```
int i = 1, j = 1, k;
```

```
k = !i || (j = j - 1);
```

NEPLATÍ, k bude 0

```
k = i || (j = j - 1);
```

PLATÍ, k bude 1

```
k = i >= 0 && --i % 2;
```

NEPLATÍ, k bude 0

Príkaz `else-if`

15

Príkaz `else-if`

- **Jednotlivé výrazy sa vyhodnocujú postupne**

- Prvý, ktorý sa vyhodnotí ako splnená podmienka, ten sa vykoná a končí

- Ak nie je splnená žiadna podmienka, vykoná sa príkaz za posledným `else`

```
if (podmienka_1)
    prikaz_1
else if (podmienka_2)
    prikaz_2
else if ...
    ...
else
    prikaz_n
```

Ternárny operátor

17

Ternárny operátor

- **Podmienený výraz – ternárny operator**

```
podmienka ? vyraz_1 : vyraz_2
```

má význam:

Ak **podmienka**, tak **vyraz_1**, inak **vyraz_2**

```
int i, j = 2, k;

i = (j == 2) ? 1 : 3;

k = (i > j) ? i : j;
```

Operátor čiarky

19

Operátor čiarky

- **Operátor čiarky vyhodnocuje ľavý operand pred pravým operandom**
- Syntax: `vyraz_1, vyraz_2`

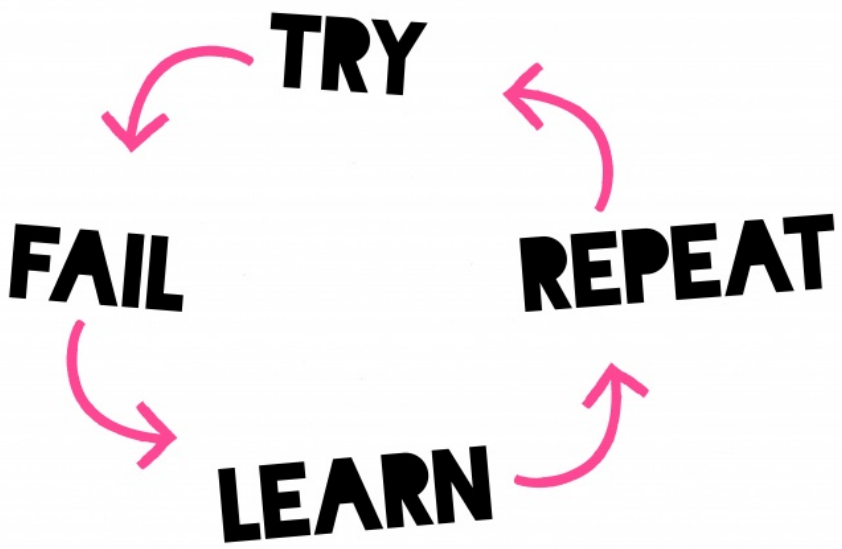
```
int i = 2, j = 4;

j =  ( i++ , i - j );
```

Cykly

21

Cykly

- **Keď potrebujeme, aby program tú istú operáciu vykonal opakovane**
- Cykly umožňujú opakovať príkaz, alebo celý blok príkazov
- Tri typy cyklov
  - `for`
  - `while`
  - `do-while`



Cyklus for

- **Používa sa vtedy, keď vieme dopredu počet prechodov cyklom**

```
for (vyraz_start; vyraz_stop; vyraz_iter)
   prikaz;
```

- Na začiatku sa vyhodnotí vyraz_start
- Otestuje sa, či výraz_stop platí, ak nie, cyklus končí
- Ak áno, vykoná sa prikaz a vyraz_iter
- Bodkočiarky sú povinné, výrazy nie, a nemusia spolu súvisieť

Cyklus `while`

- **Cyklus iteruje dovtedy, kým platí podmienka**

```
while (podmienka)
    prikaz;
```

- Podmienka sa testuje pred prechodom cyklu
- Nevieme, koľko krát cyklus zbehne, jeho ukončenie môže ovplyvniť niečo, čo sa nachádza v tele cyklu

Cyklus do-while

- **Cyklus testuje podmienku po prvej iterácii**

```
do {
   prikazy;
} while (podmienka)
```

- Aspoň 1 krát sa cyklus určite vykoná
- Pri nesplnenej podmienke cyklus končí
- Používa sa zriedka

Príkaz `break` a `continue`

- **Príkazy, ktoré je možné použiť vo všetkých typoch cyklov, menia štandardné správanie cyklu**

`break`	Ukončuje cyklus (ukončuje najvnútornejšiu slučku)
`continue`	Skáče na koniec slučky, v ktorej sa nachádza a vynucuje ďalšiu iteráciu

Footnotes

- **Prednáška je dostupná na YouTube:**

https://www.youtube.com/watch?v=P7LmN1S3cbY

V prednáške boli použité materiály zo slidov prednášok ZPrPr1 od Gabriely Grmanovej.

Q?

28