

Algoritmizácia a programovanie

2. prednáška

Ján Grman



Obsah prednášky

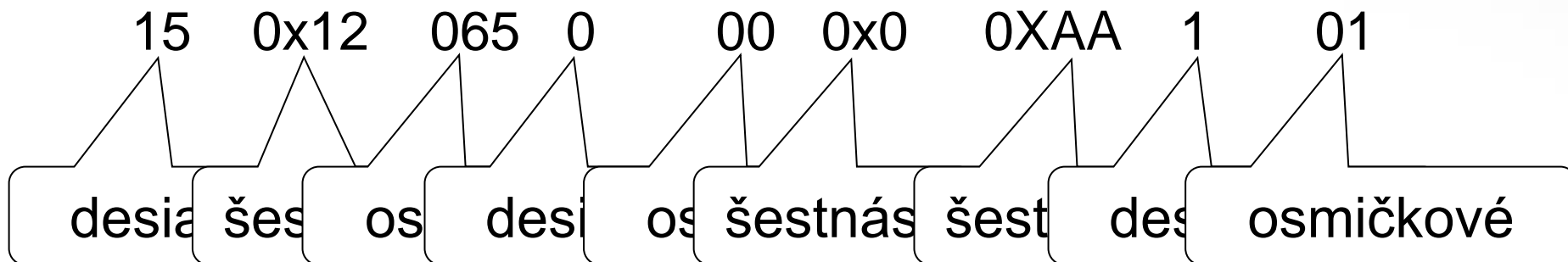


1. Konštanty a operátory
2. Formátovaný vstup-výstup
3. Príklady (2)
4. Príkazy vetvenia (`?:`, `if-else`)

Celočíselné konštanty



- celočíselné konštanty
 - desiatkové: postupnosť číslíc, na prvom mieste nesmie byť 0 (iba, ak je samotná nula)
 - osmičkové (oktalové): číslica 0 nasledovaná postupnosťou osmičkových číslíc (0 - 7)
 - šestnástkové (hexadecimálne): číslica 0 nasledovaná znkom x (alebo X) a postupnosťí hexadecimálnych číslíc (0 - 9, a - f, A - F)



Celočíselné konštanty



- typ konštanty
 - určený implicitne - podľa veľkosti konštanty
 - explicitne - použitím prípony L (alebo l) pre long , napr. 12345678**L**
- unsigned
 - explicitné určenie, či je konštanta unsigned - použitím prípony U (alebo u), napr. 129**u**, 123456**LU**
- záporné konštatny
 - určené znamienkom mínus (-), napr. -56

Celočíselné konstanty: příklad



```
int main()
{
    int i, j = 0xAA;
    unsigned int u;

    i = 017;
    j = j + 2 * i;
    u = 145u;

    return 0;
}
```

konstanta: 170

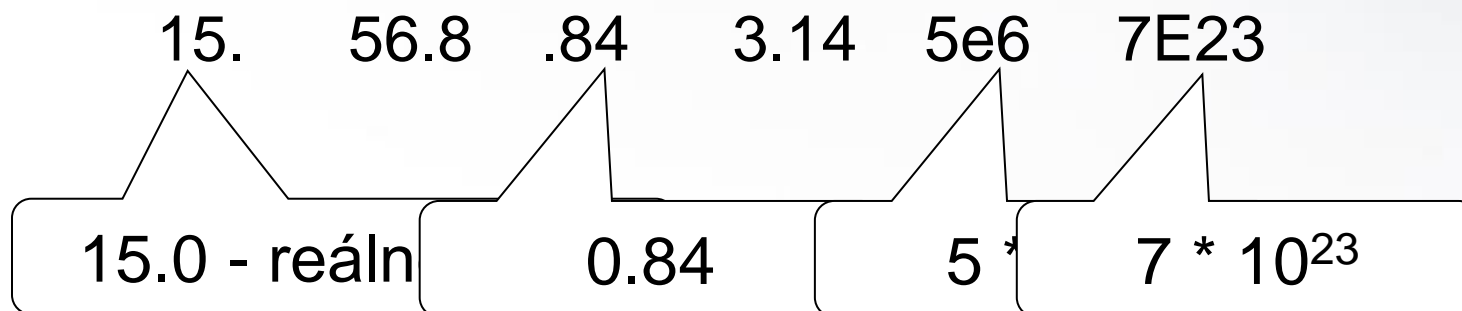
konstanta: 15

konstanta: 145 (ako unsigned)

Reálne konštanty



- podľa bežných zvyklostí
 - môžu obsahovať desatinnú bodku na začiatku aj na konci



- typ
 - float - pomocou prípony F (alebo f), napr. 3.14f
 - long double - pomocou prípony L (alebo l), napr. 12e3L

Reálne konštanty: príklad



```
int main()  
{  
    long i = 25L;  
    float f_1, f_2;  
  
    f_1 = .25;  
    f_2 = 80.;  
  
    return 0;  
}
```

konštanty: 0.25 a 80.0

Znakové konštanty



- znak uzatvorený v apostrofoch, napr. `'a'`, `'*'`, `'4'`
- hodnota (ordinárne číslo) je odvodená od ASCII tabuľky
- veľkosť znakovkej konštanty je typu `int`, nie `char`!
- znaková konštanta neviditeľného znaku:
 - `\ddd`, kde `ddd` je kód znaku - zložený z troch oktalových číslíc, napr. `\012`, `\007`
 - `\0XHH`, napr. `\0x0A`, `\0XD`, `\0X1f`
- znak `\`, nazývaný escape character - mení význam
 - napr. `012` nie je znak (len jeden znak), ale `\012` je znak

Escape sekvencie



- niektoré escape sekvenie majú okrem numerického kódu aj znakový ekvivalent:

\n	0x0A	nový riadok (new line, line feed)
\r	0x0D	návrat na začiatok riadku (carriage return)
\f	0x0C	nová stránka (formfeed)
\t	0x09	tabulátor (tab)
\b	0x08	posun doľava (backspace)
\a	0x07	písknutie (allert)
\\	0x5C	spätné lomítko (backslash)
\'	0x2C	apostrof (single quote)
\0	0x00	nulový znak (null character)

Znakové konštanty: príklad



```
int main()
{
    long i = 25L;
    float f_1 = .25, f_2 = 92E-1;
    char c, ch;

    c = 'a';
    ch = '\n';

    return 0;
}
```

konštanty: 'a' a nový
riadok

Reťazcové konštanty



- reťazec uzatvorený do úvodzoviek
 - napr. `"Toto je reťazcova konstanta"`
- ANSI C umožňuje zreťazovanie dlhých reťazcových konštánt (kvôli sprehl'adneniu)
 - napr. `"Takto vyzera velmi dlhy reťazec"`
 - `"Takto vyzera " "velmi dlhy reťazec"`
 - `"Takto vyzera " "velmi "`
`"dlhy reťazec"`

Špeciálne unárne operátory



- inkrement ($++$) ($\approx + 1$) `i++;`
- dekrement ($--$) ($\approx - 1$) `j--;`
- oba operátory sa dajú použiť ako:
 1. prefix: **`++vyraz`** `++i;` `--j;`
 - inkrementovanie **pred** použitím
 - **vyraz** je zvýšený o jednotku a potom je táto nová hodnota vrátená ako hodnota výrazu
 2. postfix: **`vyraz--`** `i--;` `j++;`
 - dekrementovanie **po** použití
 - je vrátená pôvodná hodnota **vyraz**-u a potom je výraz zväčšený o jednotku

Špeciálne unárne operátory



- výraz musí byť l-hodnota

~~45++;~~

++i;

~~--(i + j);~~

- použitie operátorov ++ a -- :

```
int i = 5, j = 1,  
k;
```

```
i++;
```

```
j = ++i;
```

```
j = i++;
```

```
k = --j + 2;
```

i bude 6

j bude 7, i bude 7

j bude 7, i bude 8

k bude 8, j bude 6, i bude 8

Operátory priradenia



- okrem jednoduchého priradenia =
- rozšírené priraďovacie operátory:

– namiesto

```
x = x operator vyraz;
```

kde **x** je l-hodnota, sa použije:

```
x operator= vyraz;
```

```
x += vyraz
```

```
x -= vyraz
```

```
x *= vyraz
```

```
x /= vyraz
```

```
x %= vyraz
```

nedávať medzeru medzi
operátor a =

a ďalšie, odvodené z iných operátorov

Opakovanie



hlavná funkcia programu: nemá argumenty a vracia hodnotu

```
int main()
{
    int i, j = 1, k;

    j++;
    ++j;
    k = i = 2 * j;
    j = --i;
    k = i++;
    k = --i + 2;

    j += i++;
    k *= 3;
    return 0;
}
```

definovali sme 3 celočíselné premenné: *i*, *j*, *k* a premennú *j* sme inicializovali na 1

j = 2

j = 3

k = 6, *i* = 6

i = 5, *j* = 5

k = 5, *i* = 6

i = 5, *k* = 7

j = 10, *i* = 6

k = 21

Opakovanie



```
int main()
{
    int i;
    float r = .25;
    char c;

    c = 'a';
    c++;
    c = '\n';

    i = 2;
    r *= i;
    return 0;
}
```

definovali sme:

- celočíselnú premennú `i`,
- reálnu premennú `r`, ktorú sme inicializovali na `0.25`
- premennú pre znak `c`

`c = 'a'`

`c = 'b'`

`c = '\n'`

`i = 2`

`r = 0.5`

Terminálový vstup a výstup



- Načítanie a výpis znaku

- vstup: `int getchar()`

- výstup: `void putchar(int c)`

pracujú s premennými typu
int

- Načítanie a výpis s využitím formátovacieho reťazca

- vstup: `scanf("...", ...)`

- výstup: `printf("...", ...)`

"..." - formátovací
reťazec,
... - premenné

Špecifikácie riadiaceho reťazca



- za znakom %:

ak načítavame len jeden znak, potom `zn = getchar()` ; je lepšie ako `scanf ("%c", &zn)` ;

c	znak
d	desiatkové číslo typu signed int
ld	desiatkové číslo typu signed long
u	desiatkové číslo typu unsigned int
lu	desiatkové číslo typu unsigned long
f	float (pre <code>printf()</code> tiež double)
Lf	long double
lf	double
x	hexadecimálne číslo malými písmenami
X	hexadecimálne číslo veľkými písmenami
o	osmičkové číslo
s	reťazec

pre `printf()` aj double

L musí byť veľké

niekedy sa nedá použiť aj pre `printf()`

Špecifikácie riadiaceho reťazca: príklady



```
printf("Znak '%c' ma ASCII kod %d (%XH)", c, c, c);
```

Pre `c='A'` vypíše: Znak 'A' ma ASCII kod 65 (14H)

```
printf("Znak '%c' ma ASCII kod %d (%XH)\n",  
      '*', '*', '*');
```

vypíše: Znak '*' ma ASCII kod 42 (2AH)

Špecifikácie riadiaceho reťazca: príklady



```
printf("Je presne %2d:%2d\n", hodiny, minuty);
```

za % môžeme určiť formát čísla
(počet cifier)

vypíše: Je presne 1:12
alebo napr.: Je presne 13: 3

```
printf("Utratili sme: %6.2f SKK.\n", pocet * cena);
```

vypíše: Utratili sme: 13.60 SKK.

```
printf("Kolko stalo %s pivo?\n", "jedno");
```

vypíše: Kolko stalo jedno pivo?

Príklad 1



```
#include <stdio.h>

int main()
{
    int i, j = 2;

    printf("Zadajte cislo: ");
    scanf("%d", &i);

    i *= 5 * (j + 1);

    printf("\nVysledok: %d\n", i);

    return 0;
}
```

Čo program vypíše pre $i=1$?

výsledok je: 15

Príklad 2



Doplňte chýbajúce vynechané príkazy označené komentárom

```
#include <stdio.h>

int main()
{
    char c;
    float f;

    printf("Zadajte oddeľovac (znak): ");
    c = getchar();          /* načítanie znaku */
    printf("Zadajte reálne číslo: ");
    scanf("%f", &f);        /* načítanie r. čísla */
    printf("číslo %c %.2f %c\n", c, f, c);

    return 0;
}
```

Príklad 3



program načíta malé písmeno,
skonvertuje ho na veľké písmeno a
vypíše ho

```
#include <stdio.h>

int main()
{
    int c;

    printf("Zadajte male pismeno: ");
    c = getchar() - 'a' + 'A';

    printf("Male pismeno: %c\n", c);
    return 0;
}
```

Príklad 4



program načíta 3 reálne čísla,
vypočíta a vypíše ich priemer

```
#include <stdio.h>

int main()
{
    double x, y, z;

    printf("Zadajte 3 realne cisla: ");
    scanf("%lf %lf %lf", &x, &y, &z);
    printf("Priemer: %.2f\n", (x + y + z) / 3);
    return 0;
}
```


Riadiace štruktúry – príkazy vetvenia



Booleovské výrazy



- nie je pre ne vytvorený špeciálny typ, používa sa `int`
 - FALSE: 0
 - TRUE: nenulová hodnota (najčastejšie 1)

<code>==</code>	rovnosť
<code>!=</code>	nerovnosť
<code><</code>	menší
<code><=</code>	menší alebo rovný
<code>></code>	väčší
<code>>=</code>	väčší alebo rovný
<code>&&</code>	logický súčin
<code> </code>	logický súčet
<code>!</code>	negácia

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

```
(x == 10)
```

1 (TRUE)

```
(y < x)
```

1 (TRUE)

```
(x != 10)
```

0 (FALSE)

1 (TRUE)

```
(y <= x) && (y > 2)
```

0 (FALSE)

```
(x < 10) || (y == 20)
```

Priradenie / porovnanie



x = 10

priradenie, **x** zmení
pôvodnú hodnotu na 10

x == 10

porovnanie, ak má **x**
hodnotu 10, výsledkom
výrazu je 1 (TRUE), inak
0 (FALSE)

Skrátené vyhodnocovanie logických výrazov



- v jazyku C sa logický súčet a súčin vyhodnocujú v skrátrenom vyhodnocovaní (*short circuit*)
 - argumenty sú vyhodnocované zľava a hneď ako je zrejmý konečný výsledok, vyhodnocovanie sa skončí

logický súčin: ak hodnota i-teho podvýrazu je 0, celý výraz je 0:
- ak `y == 0`, hodnota výrazu je FALSE a delenie nulou nenastane

napr.: `(y != 0) && (x / y < z)`

logický súčet: ak hodnota i-teho podvýrazu je 1, celý výraz je 1:
- ak `x > 10`, hodnota výrazu je TRUE a `x % 5` nenastane

`(x > 10) || (x % 5)`

Skrátené vyhodnocovanie logických výrazov



- Výhody:
 - Ak prvý podvýraz zisťuje, či je drahý výpočet potrebný

```
int a = 0;  
if (a && funckcia(b)) {  
    ...  
}
```

- Ak prvý podvýraz zabezpečuje, že druhý podvýraz neskončí s chybou (napr. delenie nulou, prístup na nesprávne miesto pamäte)

```
int delitel = 0;  
if (delitel && delenec/delitel) {  
    ...  
}
```

Priority vyhodnocovania logických výrazov



Operátor	smer vyhodnocovania
! ++ -- - + (<i>typ</i>)	zprava doľava
* / %	zľava doprava
+ -	zľava doprava
< <= >= >	zľava doprava
== !=	zľava doprava
&&	zľava doprava
	zľava doprava
? :	zprava doľava
= += -= *= ...	zľava doprava
,	zľava doprava

unárne operátory +
pretypovanie

`y != 0 && x / y < z`

`x > 10 || x % 5`

Priority vyhodnocovania logických výrazov



- aritmetické operátory a operátory porovnania majú väčšiu prioritu ako logické operátory

```
( (c >= 'A') && (c <= 'Z') )
```

Závierky tam nemusia byť, pretože `>=` a `<=` má väčšiu prioritu ako `&&`

- ak si nie ste istí, či zátvorky dať, radšej ich uveďte
- nezamieňajte `&&` za `&` a `||` za `|` - `&` a `|` sú bitové operácie

Priority vyhodnocovania logických výrazov: príklady



- pre všetky príklady platí:

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

```
j = j && (i = 2);
```

i bude 2, j bude 1, pretože `j == 1` a výraz `(i = 2)` má hodnotu 2

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

j bude 0, pretože `i == 1`

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

j bude 1, pretože `(i / 2)` nezáleží

```
j = !j && (i = i + 1);
```

j bude 0, i bude 1, i sa neinkrementuje, pretože `!j` je FALSE

Podmienený výraz



- ternárny operátor

```
podmienka ? vyraz_1 : vyraz_2
```

má význam:

ak podmienka tak vyraz_1, inak vyraz_2

- príklad

```
int i, k, j = 2;  
  
i = (j == 2) ? 1 : 3;  
k = (i > j) ? i : j;
```

i bude 1, pretože `j == 2`

k bude maximum z `i` a `j`

Použitie podmieneného výrazu



- používa sa len na jednoduché výrazy
- väčšinou sa nepoužíva, v C je čitateľnejšia konštrukcia `if-else`
- napr. konverzia znaku v premennej `c` na malé písmeno

```
MALE = (c >= 'A' && c <= 'Z') ? c + 'a' - 'A' : c;
```

zátvorky sa uvádzajú len kvôli čitateľnosti

Podmienený výraz: príklad



program načíta znak z klávesnice a
ak je to veľké písmeno, zmení ho na
malé a naopak, výsledný znak vypíše

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int c;
```

```
    c = getchar();
```

```
    c = (c >= 'A' && c <= 'Z') ? c + 'a' - 'A' :
```

```
    c + 'A' - 'a';
```

```
    putchar(c);
```

```
    return 0;
```

```
}
```

vložíme popis knižnice
na vstup a výstup

vytvoríme funkciu main

definujeme premennú

načítame znak

zmeníme znak

vypíšeme znak

Operátor čiarky



- tak ako logický súčin (`&&`), logický súčet (`||`), ternárny operátor (`? :`) - už len operátor čiarky (`,`) vyhodnocuje ľavý operand pred pravým
- syntax: `vyraz_1, vyraz_2`
- spracováva sa tak, že sa vyhodnotí **vyraz_1**, jeho výsledok sa zabudne a vyhodnotí sa **vyraz_2**

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

toto nie je operátor čiarky

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

i bude 3 a j bude -1

Operátor čiarky



- je vhodné používať ho len v riadiacich príkazoch
for a while

Poradie vyhodnocovania



- len operátory:
 - logický súčin (&&)
 - logický súčet (||)
 - ternárny operátor (? :)
 - operátor čiarky (,)

zaručujú poradie vyhodnocovania, iné operátory
nezaručujú žiadne poradie vyhodnocovania

```
int j, i = 1;  
j = ++i - (i = 3);
```

nie je dané, čo sa vyhodnotí skôr,
ak ľavý operand: i bude 3, j bude -1,
ak pravý operand: i bude 4, j bude 1

Príklad



program načíta z klávesnice dva znaky a vypíše znak s menším ordinálnym číslom

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int c, d;
```

```
    c = getchar();
```

```
    d = getchar();
```

```
    putchar(c < d ? c : d);
```

```
    return 0;
```

```
}
```

vložíme popis knižnice na vstup a výstup a vytvoríme funkciu **main()**

definujeme dve

ak je hodnota premennej **c** menšia ako hodnota premennej **c**, tak vypíšeme ju, inak **d**

Príklad: trochu inak



program načíta z klávesnice dva znaky a vypíše znak s menším ordinálnym číslom

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int c, d;
```

```
    c = getchar();
```

```
    putchar(c < (d = getchar()) ? c : d);
```

```
    return 0;
```

```
}
```

načítanie znaku do premennej **d** priamo v príkaze

Príkaz `if`



- jeden z najpoužívannejších príkazov
- syntax:

```
if (podmienka)  
    prikaz
```

zátvorky sú nevyhnutné, odporúča sa dať za `if` medzeru

- ak platí **podmienka**, vykoná sa **prikaz**

Príkaz if: príklad



program načíta znak z klávesnice a ak je to veľké písmeno, vypíše jeho ordinálne číslo

```
#include <stdio.h>
```

```
int main()  
{
```

```
    int c;
```

```
    c = getchar();
```

```
    if (c >= 'A' && c <= 'Z')
```

```
        printf("%d\n", c);
```

```
    return 0;
```

```
}
```

ak hodnota c je veľké písmeno
vypíš ho na obrazovku

Príkaz `if`: príklad: C-štýl



program načíta znak z klávesnice a
ak je to veľké písmeno, vypíše jeho
ordinálne číslo

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int c;
```

```
    if ((c = getchar()) >= 'A' && c <= 'Z')
```

```
        printf("%d\n", c);
```

```
    return 0;
```

```
}
```

využívame skutočnosť, že aj
priradenie je výraz

C-štýl - detailnejšie



```
if ( c = getchar()  >= 'A' && c <= 'Z' )  
    printf("%d\n:", c);
```

ak by sme neuzátvorkovali výraz `c = getchar()`, celý príkaz by fungoval takto: `getchar()` prečíta znak, porovná ho s 'A'.

- Ak je výsledok logická 0 (FALSE), test končí a do `c` je priradená 0
- Ak výsledok testu je logická 1, porovnáva sa ešte stále nedefinovaná hodnota premennej `c` s 'Z'. Výsledkom je buď logická 0, alebo 1, ktorá sa vynásobí s výsledkom z predošlého testu a do `c` sa priradí buď logická 0, alebo 1

Rozšírený príkaz `if`



- príkaz `if` môžeme rošíriť aj o časť `else`, ktorá sa vykoná, ak podmienka nie je splnená:

```
if (podmienka)
    prikaz_1
else
    prikaz_2
```

- ak platí `podmienka`, vykoná sa `prikaz_1` , inak sa vykoná `prikaz_2`
- ak je v sebe vnorených viac príkazov `if`, tak `else` patrí vždy k najbližšiemu `if`-u

Rozšířený příkaz `if`: příklady



```
if (i > 3)
    j = 5;
else
    j = 1;
```

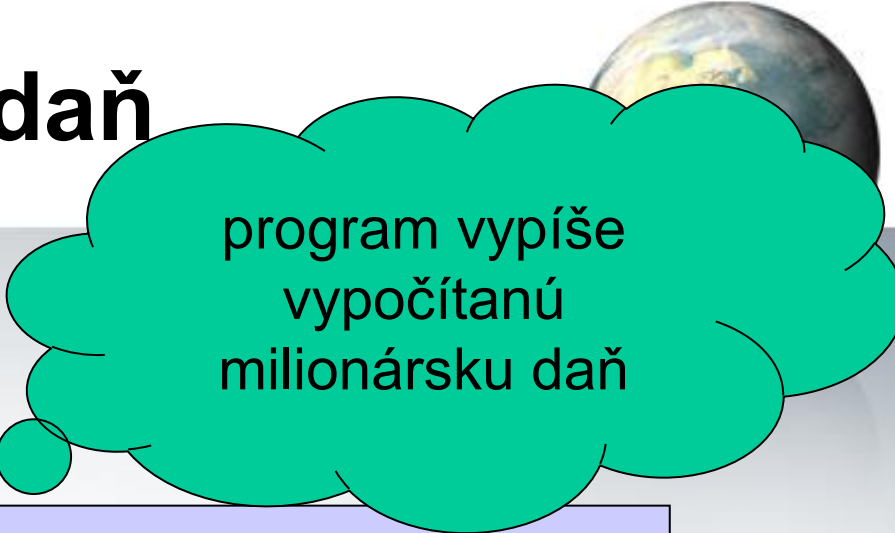
ak je `i > 3`, `j` bude 5, inak 1

```
if (i > 3) {
    j = 5;
    i = 7;
}
else
    j = 1;
```

zložený príkaz (vetva `if`) je uzatvorený v `{ }`

ak `i > 3`, `j` bude 0 a `i` bude 7, inak `j` bude 1 a `i` nezemní hodnotu

Príklad: milionárska daň



program vypíše
vypočítanú
milionársku daň

```
#include <stdio.h>

int main()
{
    double prijem;

    printf("Zadaj prijem: ");
    scanf("%lf", &prijem);

    if (prijem >= 1000000)
        printf("Milionarska dan: %lf", prijem * 0.3);
    else
        printf("Z prijmu sa neplati milionarska dan.")
    return 0;
}
```

Príklad: párne/nepárne čísla



```
#include <stdio.h>
```

```
int main()  
{
```

```
    int i;
```

```
    printf("Zadajte cislo: ");  
    scanf("%d", &i);
```

```
    printf("%s.\n", (i % 2)? "Neparne" : "Parne");
```

```
    return 0;
```

```
}
```

program zistí, či
načítané číslo je
párne, alebo
nepárne.

Prepíšte if pomocou
ternárneho operátora


```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int a, b;
```

```
    char c;
```

```
    printf("Vyber:\n");
```

```
    printf("(+) Scitaj, (-) Odcitaj, (*) Nasob, (/) Del?\n");
```

```
    c = getchar();
```

```
    printf("Zadaj prve cislo: ");
```

```
    scanf("%d", &a);
```

```
    printf("Zadaj druhe cislo: ");
```

```
    scanf("%d", &b);
```

```
    if(c == '+') printf("%d", a+b);
```

```
    else if(c == '-') printf("%d", a-b);
```

```
    else if(c == '*') printf("%d", a*b);
```

```
    else if(c == '/')
```

```
        if(b != 0) printf("%d", a/b);
```

```
        else printf("Nulou sa nedeli.");
```

```
    else printf("Zadali ste nespravny znak.");
```

```
    return 0;
```

```
}
```

program zistí, akú
operáciu má vykonať a
podľa toho sčíta,
odčíta, vynásobí alebo
vydelí 2 čísla.

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main(void)
{
```

```
    int magicke;
```

```
    int tip;
```

```
    magicke = rand();
```

Vráti náhodné číslo.

```
    printf("Vas tip na magicke cislo: ");
```

```
    scanf("%d", &tip);
```

```
    if (tip == magicke) {
```

```
        printf("*** BINGO! ***");
```

```
        printf(" %d je magicke cislo.\n", magicke);
```

```
    }
```

```
    else {
```

```
        printf("Nespravne, ");
```

```
        if (tip > magicke) printf("tip je prilis vysoky.\n");
```

```
        else printf("tip je prilis nizky.\n");
```

```
    }
```

```
    return 0;
```

```
}
```

program náhodne
vyberie magické číslo a
zistí, či ho používateľ
tipol správne.

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    double x, y, z, max;
```

```
    printf("Zadaj tri realne cisla: ");
```

```
    scanf("%lf %lf %lf", &x, &y, &z);
```

```
    if (x > y) {
```

```
        if (x > z)
```

```
            max = x;
```

```
        else
```

```
            max = z;
```

```
    }
```

```
    else {
```

```
        if (y > z)
```

```
            max = y;
```

```
        else
```

```
            max = z;
```

```
    }
```

```
    printf("Najvacsie cislo je %lf \n", max);
```

```
    return 0;
```

```
}
```

program načíta 3
reálne čísla a vypíše
najväčšie z nich

prázdny riadok je len kvôli
prehľadnosti

zátvorky - kvôli prehľadnosti, tu
je jednoznačné, ktorý **else**
patrí ku ktorému **if**

Formátovací štýl



- Dobré formátovanie programu
 - Výrazne napomáha dobrej čitateľnosti programu
 - Na prvý pohľad sa dá programu rozumieť
- Kompilátor ignoruje biele znaky
 - Dobrý formátovací štýl je výhodný pre ľudí

```
#include <stdio.h>

int main()  /* hlavny program */
{
    double x, y, z, max;

    printf("Zadaj dve realne cisla: ");
    scanf("%lf %lf", &x, &y &z);

    if (x > y) {          /* max ak x > y */
        if (x > z)
            max = x;
        else
            max = z;
    }
    else {                /* max ak x <= y */
        if (y > z)
            max = y;
        else
            max = z;
    }

    printf("Najvacsie cislo je %lf \n", max);
    return 0;
}
```



Ukážka dobrého
formátovacieho
štýlu



```
#include <stdio.h>

int main()  /* hlavny program */ {double x, y, z, max;
printf
("Zadaj dve realne cisla: ");
scanf("%lf %lf", &x, &y &z);if (x > y) {
/* max ak x > y */
if (x > z) max = x; else
max = z;}
else {/* max ak x <= y */
if (y > z)
max = y;
else max = z;}
printf
("Najvacsie cislo je %lf \n", max);return 0;}
```

Ukážka zlého formátovacieho štýlu