# Programátorská súťaž ACM
## 21.-22. 10. 2016 v Bratislave

acm International Collegiate Programming Contest

IBM event sponsor

Max. 3-členné družstvá pošlite mail na adresu
**acm.icpc@fiit.stuba.sk do stredy 19.10.2016 do 18:00 hod.**

Ako odpoveď na Váš mail dostanete potvrdenie účasti a ďalšie pokyny k dokončeniu registrácie.

**Počet tímov je obmedzený!**

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

# Základy procedurálneho programovania 1

## Prednáška č. 2

# Čím žijú programy?

## (pokračovanie)

26. 9. 2016

zimný semester 2016/2017

# I Na predchádzajúcej prednáške

- Počítač nástroj, ktorý vykonáva programy
- **Program** je postupnosť inštrukcií, ktoré povedia počítaču, ako vykonať úlohu
- **Programovacie jazyky** sprostredkujú inštrukcie počítaču, ktorý ich vykonáva
- **Programovanie** je
  1. vymyslenie a navrhnutie postupu riešenia úlohy, a
  2. zapísanie riešenia v programovacom jazyku

# Na predchádzajúcej prednáške

- **Premenná** je pomenovaný priestor v pamäti pre uloženie dát
- **Priradenie** naplní hodnotu do premennej (do pamäti vyhradenej pre premennú) **vek ← 24**
- **Výraz (aritmetický)**
  - Vyhodnotenie výrazov
  - Operátory (+, -, =, ... ) a operandy (premenné, konštanty)
- **Logický výraz**
  - AND, OR, NOT, ...
  - Skrátené vyhodnocovanie
- **Vetvenie (if / else)**

```
if (priestupnyRok)
    pocetDniRoku=366;
else
    pocetDniRoku=365;
```

# 2 Dátové typy

- Rozličné dáta, ktoré ukladáme do pamäti zaberajú v pamäti rozlične veľa miesta
- Koľko máme pamäte? Napr. 8GiB – 8*10^9 byte-ov
- 1 byte je 8 bitov
- Koľko rozdielnych stavov vieme uschovať v 1 bit-e? dva (0 a 1)
- Koľko rozdielnych stavov vieme uschovať v 1 byte? 2^8 = 256 (teda čísla: 0, 1, 2, ..., 254, 255)
- Do pamäti pristupujeme po celých byte-och
- **Adresa pamäte** je poradie (číslo) byte-u od začiatku pamäte

# 2 Dátové typy (2)

- **Char** (1 byte = 8 bitov) dokáže uschovať najviac 256 rôznych stavov (čísel).
  - Ak nepotrebujeme záporné čísla:
    **unsigned char:** 0,1,..,255 ... (občas sa nazýva **byte**)
  - Ak chceme využiť aj záporné čísla:
    (signed) **char:** -128, ... -1,0,1,...127
- Čo ak chceme v premennej uschovať **menej ako 256 rôznych stavov**?
  - Napr. 1 stav ... Čo použijeme?
    Nič. Nemá zmysel mať premennú vždy v rovnakom stave.
  - 2 rôzne stavy ... Premenná typu **bool** (alebo **boolean**) hodnoty 0/1 resp. false/true, aj tak zaberá 1 byte (lebo menej sa nedá v pamäti adresovať)

# 2 Dátové typy (3)

- Čo ak chceme v premennej uschovať **viac ako 256 rôznych stavov**?
  - Môžeme využiť 2 byte ... 65536 stavov
    (signed) **short:** -32 768, ..., 0, ..., 32 767
    **unsigned short:** 0, ..., 65 535
  - Štandardný názov pre celé číslo je **int**, resp. **unsigned int**.
    Závisí aj od architektúry (32bit vs. 64bit)
    Môžeme využiť 4 byte ... 4,294,967,295 stavov
    (signed) **int:** -2 147 483 648, ..., 0, ... 2 147 483 647
    **unsigned int:** 0, ..., 4 294 967 295
  - „Väčšie“ celé číslo sa nazýva **long**, resp. **unsigned long**.
    Môžeme využiť 8 bytes.

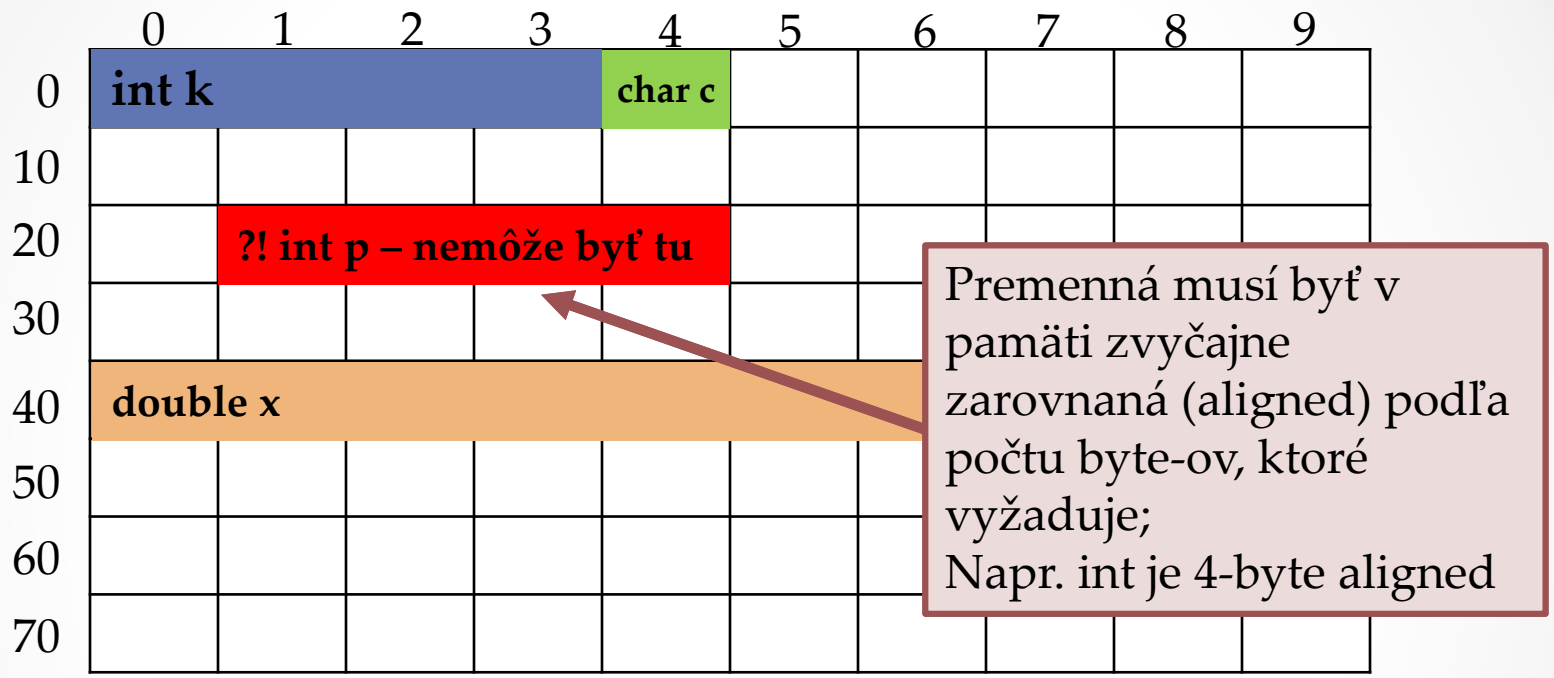
# 2 Dátové typy (3)

- Čo ak chceme v premennej uschovať reálne čísla (nekonečné množstvo stavov)?
  **Nie je to možné.**
- Môžeme uschovať desatinné čísla, ktoré počítajú s presnosťou na niekoľko desatinných miest.
  - **float (4 byte)** presnosť 6 desatinných miest
    hodnoty 1.2E-38 až 3.4E+38
    **používalo sa v minulosti, keď bola pamäť malá**
  - **double (8 byte)** presnosť 15 desatinných miest
    hodnoty 2.3E-308 až 1.7E+308
    **štandardne sa používa tento dátový typ**

# 2 Dátové typy (4)

- Dátový typ reťazec znakov (**string**) typicky píšeme v úvodzovkách „“
- „Peter“
- **Reťazec je v pamäti reprezentovaný po písmenkách** (jedno písmenko je typicky typu **char**)
- Jeden znak (char) sa typicky uvádza v apostrofoch ‘a’, ‘b’, … ‘7’
- Reťazce:
  “Bratislava”, “47”
  keď je úvodzovka (alebo iný špeciálny znak) súčasťou reťazca, tak ho escapujeme: “Uvodzovka je \” znak”.

# 2 Umiestnenie v pamäti




- Adresa premennej **k** ? 0 … Adresa premennej **c** ? 4
- Adresa premennej **x** ? 40 … Adresa premennej **p** ? 24
- Ako si do pamäte uložím znak? (napr. ‘x’)
- Ako si do pamäte uložím meno? (napr. „Bratislava“)

# 2 Typový systém (type system)

- Pravidlá, podľa ktorých sa určujú v programe typy (pre premenné, pri výpočtoch vo výrazoch, ...)
- **Staticky typované** (napr. Java, C), typ je jasne definovaný v zdrojovom kóde programu (resp. v čase kompilácie)
  - **Premenné majú typ**
  - Prípadné chyby pri typovaní sa odhalia už pri vytváraní spustiteľného programu zo zdrojového kódu
- **Dynamicky typované** (napr. Javascript), typy nemusia byť presne určené v čase kompilácie, ale až pri vykonávaní programu.
  - Premenne nie sú viazané k typu, ale: **hodnoty majú typ**, teda „aktuálny dátový typ“ premennej zodpovedá typu jej aktuálnej hodnoty

# 2 Dátové typy pri vyhodnocovaní výrazu

- Pri vyhodnocovaní výrazu sa po každej operácii môže zmeniť typ
- „Peter“ + „Cibulka“ je „PeterCibulka“ (typu **string**)
- „Peter“ + 3 (**int**) je „Peter3“ (typu **string**)
- 30/3 = 10 je typu (**int**)
- 30/4 = ?
- 30/4 = 7 je typu (**int**)
- (double)30/4 = 7.5 je typu **double**
- Premenné môžeme pretypovať (**type-cast**)

# 2 Rozsah platnosti (scope)

- Premenná je previazanie pamäti s nejakým menom
- Deklarácia: int i, j, k;  znamená tri (rôzne) premenné typu int
- Program môže byť dlhý
- Rozsah platnosti nám hovorí, kde v programe môžeme meno použiť, a s ktorou pamäťou je meno previazané
- **Blok** – časť kódu, ktorá je zoskupená spolu, typicky **ohraničená zloženými zátvorkami { }**
- Príklad (rozsah platnosti premenných vyznačený stĺpcami):

```
int i = 3;
if (...)
{
  int k = 4;
  i = 20;
}
k = k+1;
```

i

k

CHYBA: Premenná **k** na tomto mieste neexistuje!

# 3 Cyklus (loop)

- Zatiaľ poznáme len inštrukcie, ktoré sa vykonajú raz
- Čo ak chceme opakovať inštrukcie viac krát?
- Cyklus
  - Vopred daný rozsah (napr. chceme opakovať 20 krát, alebo x krát, kde x je premenná), zvyčajne použijeme **for**

```
for (premenné; podmienka; krok)
{ blok; }
```

  - Vopred nie je daný rozsah (napr. chceme opakovať, pokým platí podmienka), zvyčajne použijeme **while**, dve varianty

a)
```
while (podmienka) { blok; }
```

b)
```
do { blok; } while (podmienka);
```

# 3 Cyklus (loop) – for

- **for** (premenné; podmienka; krok)
  { blok; }
- Cyklus for pridáva (riadiace) **premenné** do rozsahu platnosti
  Napr. opakovanie x krát – vypíš čísla 1 2 3 ... x :

i

```
for (int i = 1; i <= x; i++)
{
   PRINT i
}
```

- **Podmienka** značí podmienku, ktorá musí platiť;
  ak podmienka platí, tak sa vykoná telo cyklu (blok príkazov).
- Po vykonaní tela sa vykoná **krok,** a opäť sa skontroluje podmienka, ak platí opakuje sa vykonanie tela cyklu, inak cyklus končí, a program pokračuje za cyklom.
- **Aká bude hodnota i po vykonaní cyklu?**

# 3 Cyklus (loop) – for (2)

- Napr. opakovanie x krát – vypíš čísla 1 2 3 ... x :

i

```
for (int i = 1; i <= x; i++)
{
   PRINT i
}
```

- **Aká bude hodnota i po vykonaní cyklu?**
- Ak je i je premenná z rozsahom platnosti v cykle, tak po skončení zanikne.

# 3 Cyklus (loop) – for (3)

- Napr. opakovanie x krát – vypíš čísla 1 2 3 ... x :

i

```
int i;
for (i = 1; i <= x; i++)
{
   PRINT i
}
PRINT i   ...Vypíše x+1
```

- **Aká bude hodnota i po vykonaní cyklu?**
- Ak i platí aj mimo cyklu, tak hodnota zostáva. **i bude mať hodnotu x+1**

# 3 Cyklus (loop) – while; do while

- Alternatívy, zvyčajne sa používajú pre vopred neurčený počet iterácií (opakovaní cyklu).
- Napr. program vypíš čísla 1 2 … x:

**while varianta:**

i

```
int i = 1;
while (i <= x)
{
  PRINT i
  i++;
}
```

**do-while varianta:**

i

```
int i = 1;
do
{
  PRINT i
  i++;
} while (i <= x)
```

# 3 Cyklus (loop) – iterátor, in

- Ak máme nejaký zoznam prvkov, niekedy píšeme aj formou iterátora zoznamu (operátor in):
- zoznam = {1, 2, …, x}

i

```
for (i  in  zoznam)
{
   PRINT i
}
```

# 3 Cyklus (loop) – vnorené cykly

i

```
for (int i = 0; i < 10; i++)
{
  for (int j = 0; j < 10; j++)
  { PRINT (10*i + j) }
  PRINT newline
}
```

j

- Vypíše tabuľku

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
...
90 91 92 93 94 95 96 97 98 99
```

# 3 Cyklus (loop) – prerušenie cyklu

- **break** – prerušiť vykonávanie cyklu
- **continue** – ukončiť vykonávanie aktuálnej iterácie, a pokračovať na krok, podmienku (a možno ďalšiu iteráciu)

```
for (int i = 0; i < 10; i++)
{
  if (i % 2 == 0)
      continue;      ... ak je i párne nepokračuj, ale choď na ďalšie
  for (int j = 0; j < 10; j++)
  {
    if (i == j)
      break;         ... ak je i rovné j skonči vykonávanie tohto cyklu
    PRINT (10*i + j)
  }
  PRINT newline
}
```

- Čo to vypíše?

~~0 1 2 3 4 5 6 7 8 9~~
10 ~~11~~ ~~12 13 14 15 16 17 18 19~~
~~20 21 22 23 24 25 26 27 28 29~~
30 31 32 ~~33~~ ~~34 35 36 37 38 39~~
~~40 41 42 43 44 45 46 47 48 49~~
…

# 4 Funkcie

- Funkcia je pomenovaná časť programu, ktorá vykonáva určitú úlohu
  (funkcia je tiež uložená v pamäti podobne ako premenné, pričom jej dáta sú samotné inštrukcie, ktoré funkcia vykonáva)
- Funkcia je definovaná ako:

```
typ nazovFunkcie(argumenty)
{
  blok;    (telo funkcie)
}
```

**argumenty** je zoznam pomenovaných dátových typov, ktoré nadobudnú platnosť v rozsahu bežiacej funkcie ako (nové) premenné.

# 4 Funkcie (2)

- Napr. výpočet obvodu obdĺžnika:

```
int obvod_obdlznika (int a, int b)
{
  return 2*a + 2*b;
}
```

a,b

**return** je príkaz, ktorý ukončí funkciu, a ako návratovú hodnotu vráti príslušnú hodnotu.

# 4 Funkcie (2)

- Návratovú hodnotu funkcie potom môžeme použiť tam, kde funkciu voláme (tzv. **volanie funkcie**), napr.:

```
int x=10, y=20, obvod;
obvod = obvod_obdlznika(x, y);
PRINT obvod
```

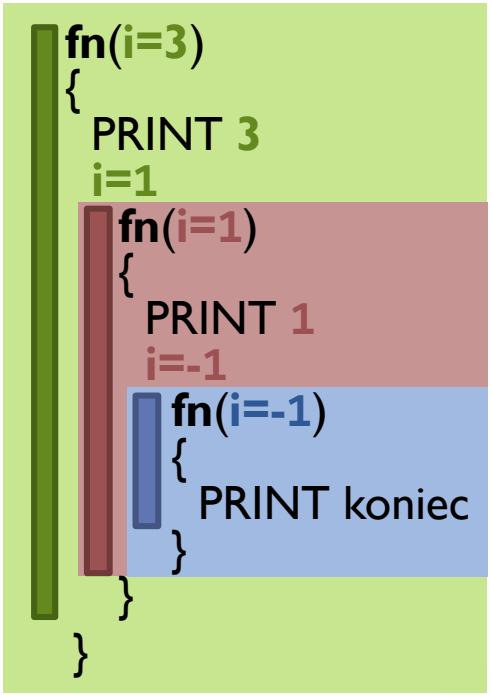
- Premenná **obvod** bude mať hodnotu 60

- Volanie funkcie vytvorí nový rozsah platnosti pre parametre funkcie
- Hlavná funkcia programu od ktorej sa začnú vykonávať príkazy tesne po spustení programu sa (niekedy) označuje **main**
  - niekedy sa explicitne neoznačuje, a hneď od začiatku zdrojového kódu sa príkazy začnú vykonávať

# 4 Funkcie – rekurzia – rozsah platnosti

- Čo keď funkcia bude vo svojom tele volať funkciu s rovnakým názvom? Nazývame to **rekurzia**
- Uplatnia sa rozsahy platnosti
- 
```
fn(int i)
{
  if (i <= 0)
    PRINT koniec
   else
  {
    PRINT i
    i = i-2;
    fn(i);
  }
}
```

- Volanie fn(3)




# 5 Programovacie jazyky

- Doteraz sme preberali všeobecne aplikovateľné princípy programovania, teraz sa pozrieme na konkrétne programovacie jazyky
- Sprostredkujú inštrukcie počítaču, ktorý ich vykonáva
- **Syntax** jazyka (**forma**) – jasne definovaná pravidlá ako zapisovať symboly jazyka
- Syntax jazyka opisuje ako by sa mali písať programy, aby boli správne zapísané v programovacom jazyku
- **Sémantika** – **význam** symbolov programu

# 5 Programovacie jazyky (2)

- Zápis v programovacom jazyku (syntax) neposkytuje ešte potrebné informácie k celkovému významu
- Čo vypíše nasledujúci program?

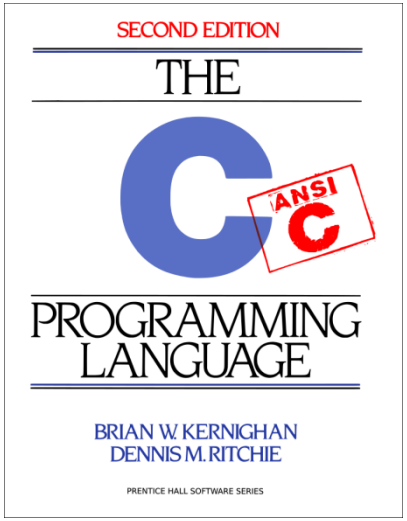
```
int x
print x
```

- Je dobre napísaný (syntax je platná), ale jeho význam (sémantika) nie je plne definovaný
  - Premenná x nie je inicializovaná
- Kompilátor dokáže skontrolovať syntax, a vytvoriť inštrukcie pre počítač podľa pravidiel jazyka, ale
- Kompilátor nedokáže nájsť všetky chyby, ktoré vzniknú pri vykonávaní programu

# 5 Programovacie jazyky (3)

- Na tomto predmete sa budeme učiť najmä

- **Programovací jazyk C**
  - Dennis Ritchie 1969
  - UNIX operačný systém
  - Efektívne mapovanie príkazov na inštrukcie počítača




  - Nadviazalo na neho veľa ďalších jazykov: C++, Java, JavaScript, C#, Objective-C, PHP, Python, Go, …
  - Najnovšia verzia C11 štandard (ratifikovaný 8.12.2011)

# 5 Programovacie jazyky (4)

- **Programovací jazyk Python**
  - Guido van Rossum 1991
  - Všeobecne použiteľný, dobre čitateľný
  - Syntax umožňuje vyjadriť koncepty jadrnejšie ako C, C++ a Java
  - Ľahko naučiteľný
  - Najnovšia verzia 3.5.0 (vydaná 13.9.2015)

# 5 C vs. Python

- C je kompilovaný jazyk
  **Python je interpretovaný**
- C je staticky typovaný
  **Python je dynamicky typovaný**
- C je procedurálny
- **Python podporuje viaceré paradigmy**
  - Procedurálne programovanie
  - Objektovo-orientované programovanie
  - Funkcionálne programovanie

# 6 Prvý (nudný) program

- C (verzia ANSI C)

```c
#include <stdio.h>

int main(void)
{
    printf("Ahoj!");
    return 0;
}
```

- Python (verzia 3)

```python
print('Ahoj!')
```

- stdio.h je knižnica, ktorá obsahuje funkcie pre vstup a výstup (scanf, printf)

# 6 Prvý (zaujímavejší) program

- C:

```c
#include <stdio.h>

int main(void)
{
    int i, x = 1, n;
    printf("Zadaj N: ");
    scanf("%d", &n);
    for (i = 1; i < n+1; i++)
        x *= i;
    printf("%d", x);
    return 0;
}
```

- Python:

```python
x = 1
n = int(input('Zadaj N: '))
for i in range(1,n+1):
    x *= i
print(x)
```

# 6 Rekurzívny výpočet

- C:

```c
#include <stdio.h>

int f(int n)
{
    if (n <= 1) // 0! = 1
        return 1;
    return n * f(n-1);
}

int main(void)
{
    int n;
    printf("Zadaj N: ");
    scanf("%d", &n);
    printf("%d", f(n));
    return 0;
}
```

- Python:

```python
def f(n):
    if n <= 1:
        return 1
    return n * f(n-1)

n = int(input('Zadaj N: '))
print(f(n))
```

# 7 Opakovanie – riešenie úlohy 1-1

- Sledujte priradenia a zistite hodnotu premennej po vykonaní všetkých priradení.

$s \leftarrow 5, j \leftarrow 6, k \leftarrow 7$

$k \leftarrow j - k$
$k \leftarrow j + s$
$k \leftarrow s + k$
$s \leftarrow s - j$
$j \leftarrow j + s$

$s = ?$

| s | j | k |
|---|---|---|
| 5 | 6 | 7 |
| | | 6 - 7 = -1 |
| | | 6 + 5 = 11 |
| | | 5 + 11 = 16 |
| 5 – 6 = -1 | | |
| | 6 + (-1) = 5 | |
| -1 | | |

# 7 Opakovanie – riešenie úlohy 1-2

- Zisti pravdivostnú hodnotu logického výrazu.

$a \leftarrow 2, s \leftarrow 0, k \leftarrow 0$

```
!a AND !a AND s AND k OR s
(!a AND !a AND s AND k) OR s
( 0 AND ..............) OR 0 == 0
```

$m \leftarrow 0, h \leftarrow 0, k \leftarrow 0$

```
!m OR !k OR k OR m AND k
!m OR !k OR k OR (m AND k)
 1 OR ..................... == 1
```

$u \leftarrow 5, l \leftarrow 2, k \leftarrow 0,$

```
!u AND !u OR !k OR !u OR k
(!u AND !u) OR !k OR !u OR k
( 0 AND ...)OR  1 .......... == 1
```

# 7 Opakovanie – riešenie úlohy 1-3

- Zisti čo vypíše vetvenie.

```
x ← 0, a ← 1, k ← 1
if (a OR k) ... true
{
  if (!a OR !a) ... false
  { ... }
  else
  {
    if (a AND !a) ... false
    { ... }
    else
    { PRINT -12 }
  }
}
else
{ ... }
```

# 8 Riešenie úlohy 2-1

- Sledujte priradenia a zistite hodnotu premennej po vykonaní všetkých priradení.

```
unsigned ch
int v = 800
v += k
k += v
v = a + k
k = v + x
x = a + k
k = x + v
v = a + k

v = ?
```

| k (uchar) | a (uchar) | v (int) | x (int) |
|---|---|---|---|
| 220 | 190 | 800 | 700 |
| | | +=220 = 1020 | |
| 1240-256-256-…=216 | | | |
| | | 190+216=406 | |
| 1106-256-256-…=82 | | | |
| | | | 190+82=272 |
| 678-256-256-…=166 | | | |
| | | 190+166 = 356 | |

# 8 Riešenie úlohy 2-2

- Sledujte priradenia a zistite hodnotu premennej po vykonaní všetkých priradení.

```
int v = 60, r
double k = 400
r = 750 + k/s
k = 700 + v/r
v = 700 + s/k
s = 300 + v/r
v = 500 + s/k
k += v
r = 500 + k/s

r = ?
```

| v (int) | r (int) | k (double) | s (double) |
|---|---|---|---|
| 60 | 120 | 400 | 600 |
| | 750+4/6=750 | | |
| | | 700+0=700 | |
| 700+6/7=700 | | | |
| | | 300+700/750=300+0=300 | |
| 500+0=500 | | | |
| | | +=500=1200 | |
| | 500+1200/300=504 | | |

# 8 Riešenie úlohy 2-3

- Sledujte priradenia a zistite hodnotu premennej po vykonaní všetkých inštrukcií.

```
int r, v=0, k=0
for (r = 2; r <= 5; r += 2)
{
  if ((r + v) % 2 == 1)
    continue
  else
    r++
  for (v = r + 2; v <= r + 4; v++)
  {
    if (r + v > 8)
      break
    k += r + v
  }
} k = ?
```

**r+=2 … r=7 … r<5 false (endloop)**

**r=5, v=6, r+v=11 … true (continue)**

**r++ … r=3**

**v++ … v=6**

**3+6 > 8 … true (break)**

**k+=3+5 … k=8**

**v=6**

**k=8**

# 9 Píšeme prvé programy

- Zadanie úlohy
  - Voľný text, treba mu porozumieť a zistiť čo je vlastne vstup, a čo by mal byť výstup
    (niečo ako slovná úloha v matematike)
  - Napr. **Napíš funkciu, ktorá určí menší z prvkov.**
- Vstup
  - Musím jednoznačne určiť čo (v zmysle programovania) je vstupom – teda, aké dátové typy, koľko, a ukážkový vstup
  - Napr. **vstupom sú dve celé čísla (int) x a y**
  - Určím **formát vstupu**

# 9 Píšeme prvé programy

- Vstup
  - Musím jednoznačne určiť čo (v zmysle programovania) je vstupom – teda, aké dátové typy, koľko, a ukážkový vstup
  - Napr. **vstupom sú dve celé čísla (int) x a y**
  - **Ukážka vstupu: x=5, y=3**
  - Určím **formát vstupu**
- Výstup
  - Musím jednoznačne určiť, čo (v zmysle programovania) sa očakáva ako výstup (riešenie úlohy) – aké dátové typy a koľko a ukážkový výstup
  - Napr. **výstupom je menšie z čísel x a y, označme z**
  - **Výstup pre ukážkový vstup(x=5, y=3): z=3**
  - Určím **formát výstupu (výpis, návratová hodnota, ...)**

# 9 Píšeme prvé programy

- Výstup
  - Musím jednoznačne určiť, čo (v zmysle programovania) sa očakáva ako výstup (riešenie úlohy) – aké dátové typy a koľko a ukážkový výstup
  - Napr. **výstupom je menšie z čísel x a y, označme z**
  - **Výstup pre ukážkový vstup(x=5, y=3): z=3**
  - Určím **formát výstupu (výpis, návratová hodnota, ...)**
- Postup
  - Vymyslím postup (algoritmus) – postupnosť príkazov programu, ktorý pre každý platný (spĺňajúci formát vstupu) vstup jednoznačným spôsobom určí výstup
  - Napr.

```
z ← x
if y<x:
  z ← y
```

Iná možnosť:

```
if x<y
  z ← x
else z ← y
```

# 9 Píšeme prvé programy

- Postup
  - Vymyslím postup (algoritmus) – postupnosť príkazov programu, ktorý pre každý platný (spĺňajúci formát vstupu) vstup jednoznačným spôsobom určí výstup
  - Napr.

    **z ← x**
    **if y<x:**
    **z ← y**

    Iná možnosť:

    **if x<y:**
    **z ← x**
    **else z ← y**

- Program

```
int min(int x, int y)
{
    int z = x;
    if (y < x)
        z = y;
    return z;
}
```

```
int min(int x, int y)
{
    if (x < y)
        return x;
    return y;
}
```

# 9 Píšeme prvé programy

- Zadanie úlohy
  - Voľný text, treba mu porozumieť a zistiť čo je vlastne vstup, a čo by mal byť výstup
    (niečo ako slovná úloha v matematike)
  - Napr. **Napíš funkciu, ktorá určí menší z prvkov.**
- Čo keď by sme mali iný (formát) vstupu?
  - Rímske čísla
  - Čísla zapísané v dvojkovej sústave
  - Desatinné čísla
  - Usporiadaná postupnosť N čísel
  - Množina N čísel
  - Dve (usporiadané) postupnosti N a M čísel

# 9 Píšeme prvé programy

- Základná schéma každej programátorskej úlohy
  1. Zadanie
  2. Vstup
  3. Výstup
  4. Postup
  5. Program