

# **Programování a modelování**

## **(část Programování)**

Přednáška 5

# Opakování

- Co je vyšší programovací jazyk?
  - kompilační vs. interpretační, příklady
- Co je překlad, překladač a jak funguje?
  - syntaxe, sémantika jazyka
  - lexikální symboly, lexikální a syntaktická analýza
  - jazyk relativních instrukcí (soubory .obj)
  - sestavení programu (linkování, linker), knihovny

(cvičení 2 předmětu Programování)

# Jazyk C

- 70. léta - AT&T Bell Laboratories
- Kernighan a Ritchie - verze K&R C
- silná vazba na UNIX
- 1988 - standardizace ANSI normou -  
ANSI C
  - ANSI 99
  - ANSI C1X (2011)

# Jak probíhá překlad v jazyce C ?

- hlavičkové soubory .h (.hpp)
  - obsahují prototypy funkcí (hlavičky)
- preprocesor
  - provádí textové úpravy zdrojového kódu (náhrady, vložení hlavičkových souborů)
  - direktivy preprocesoru – začínají znakem #, neukončují se středníkem
  - příklad:

```
#define MAX 10
#include <stdio.h>
```

# Jak probíhá překlad v jazyce C ?

## Poznámka:

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

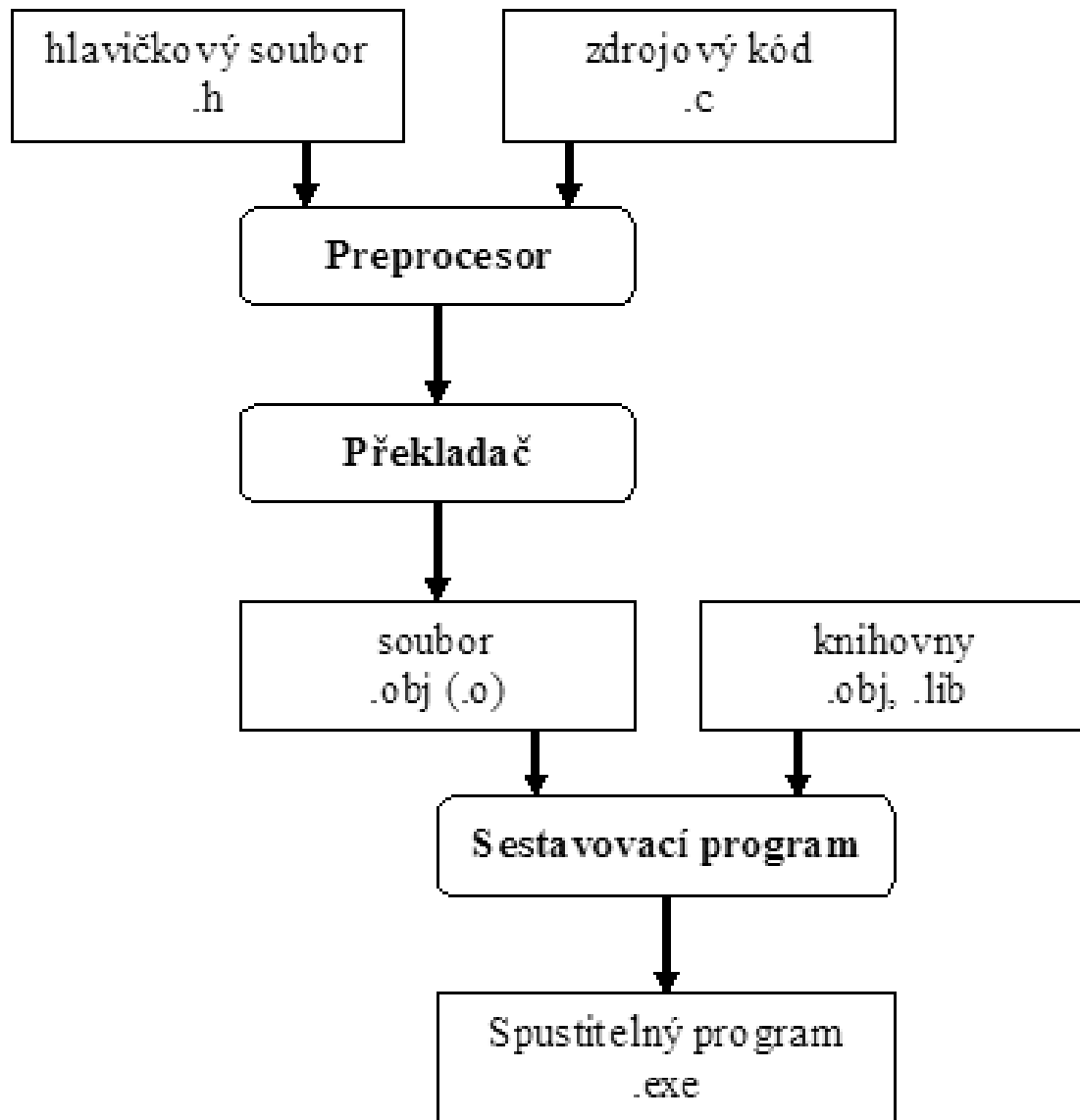
- preprocesor hledá soubor `stdio.h` ve standardních adresářích překladače (zpravidla podadresář `include`)

```
#include "soubor.h"
```

- preprocesor hledá soubor `soubor.h` v aktuálním adresáři (např. kde je uložen projekt)

```
#include "incl/soubor.h"
```

```
#include <ole/access.h>
```



- pomocí příkazů preprocesoru lze realizovat *podmíněný překlad*:

```
#define PRACOVNI
```

```
#ifndef PRACOVNI
```

```
    printf("Kontrolni tisk");
```

```
    printf("x = %X", x);
```

```
#endif
```

- pomocí příkazů preprocesoru lze realizovat *podmíněný překlad*:

```
#define PRACOVNI
```

```
#ifdef PRACOVNI
```

```
    printf("Kontrolni tisk");
```

```
    printf("x = %X", x);
```

```
#else
```

```
    printf("x = %d", x);
```

```
#endif
```



# Programovací jazyk C++

- 1983 - AT&T Bell Laboratories  
Bjarne Stroustrup - „Jazyk C s objekty“
- 1986 Bjarne Stroustrup  
The C++ Programming Language
- 1990 Bjarne Stroustrup:  
The Annotated C++ Reference Manual

- 1995 M.Ellis, B.Stroustrup  
The Annotated C++ Reference Manual
- 1997 - ANSI/ISO standardizace C++
  - vydána ISO/IEC 14882:1998
  - standard označován jako C++98
- 2003        C++03
- 2011        C++11
- 2014        C++14
- 2017        C++17
- 2020        C++20
- 2023\*       C++23

\*resp. 12/2022

- jazyk C++ je rozšířené C-čko o objektové rysy, ale *to není zcela přesné:*
  - C není podmnožinou C++, některé neobjektové rysy C++ jsou jiné oproti standardnímu jazyku C

- **přesto** lze většinu programů zapsaných v jazyce C přeložit pod překladačem C++, zejména pokud jde o zápis podle ANSI C

# Opakování základů jazyka C

- jazyk se slabou typovou kontrolou (v C++ je zesílená)
- case sensitive (rozdíl mezi malými a velkými písmeny)
- rysy funkcionálního jazyka
- označení bloku (těla funkce) - { }
- hlavní program: funkce `main`
  - `void main()`
  - `int main(int argc, char* argv[])`

# Literály:

- *konstanty*

- desítkové konstanty: 13, -5, 15L
- osmičkové konstanty: 056
- šestnáctkové konstanty: 0x20, 0X1F
- desetinná čísla: 5.3, -4E-3
- znakové: 'a', '\n', '\t', '0x0A'
- řetězcové: "Ahoj, světe!"

- *identifikátory proměnných*

- musí začínat písmenem, uvažovaná délka závisí na překladači (zpravidla 31), C++ vše

# Datové typy - jednoduché

- *celočíselné*
  - char (8 bitů)
  - short, int, long (unsigned, signed), long long
  - $\text{sizeof}(\text{short}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long}) \leq \text{sizeof}(\text{long long})$
- *plovoucí řádová čárka (desetinná čísla)*
  - float, double, long double (ANSI)
  - v některých překladačích i long long double

# Typické velikostí typů

Typ	Velikost [bit]	Rozsah
char	8	-128 až +127
short	16	-32 768 až +32 767
int	32	-2 147 483 648 až +2 147 48 3647
long	32 (64)	-2 147 483 648 až +2 147 48 3647
long long	64	-9 223 372 036 854 775 808 až +9 223 372 036 854 775 807
float	32	$-3,402823 \cdot 10^{+38}$ až $+3,402823 \cdot 10^{+38}$ nejnižší kladná hodnota $1,175494 \cdot 10^{-38}$ počet platných cifer 7 až 8
double	64	$-1,7976931348623157 \cdot 10^{+308}$ až $+1,7976931348623157 \cdot 10^{+308}$ nejnižší kladná hodnota $2,225073858507202 \cdot 10^{-308}$ počet platných cifer 15 až 16
long double	96 (80) 128 v 64-bit. překl	$-3,4 \cdot 10^{+4932}$ až $+3,4 \cdot 10^{+4932}$ nejnižší kladná hodnota $1,1 \cdot 10^{-4932}$ počet platných cifer 19



# Typické velikostí typů

- **standard jazyka C nedefinuje velikost typů, záleží to na překladači (platformě)**
- **musí pouze platit nerovnost:**

`sizeof(short) ≤ sizeof(int) ≤ sizeof(long) ≤ sizeof(long long)`

- rozsahy typů jsou definovány jako konstanty v hlavičkových souborech `limits.h`, resp. `climits` a `float.h`, resp. `cfloat`
  - `INT_MAX`, `INT_MIN`, `LONG_MAX`, ...
  - `FLT_MAX`, `DBL_MAX`, `LDBL_MIN`, ...
- rozšířené definice celočíselných typů jsou definovány v hlavičkovém souboru `stdint.h`

- **typ boolean**

- v jazyce C není, nahrazuje se typem **int**
  - jakákoliv nenulová hodnota je považována za „pravdu“,
- v C++ je definován datový typ **bool** s hodnotami **true**, **false**

```
bool je_otevren = true;
```

```
if (je_otevren) ...
```

```
if (je_otevren == false) ...
```

- **řetězec**

- není, řetězce jsou pole znaků (char\*)
- v C++ je již definován v knihovně STL typ `string`

# Komentáře

- víceřádkové komentáře v klasickém C

```
/* Toto je víceřádkový  
   komentář */
```

- jednořádkové komentáře v C ++

```
// Toto je jednořádkový komentář
```

# Proměnné:

- globální
- lokální (v bloku)
- deklarace proměnné:

```
typ proměnná;
```

```
int x=10; // počát. inicializace
```

```
char c;
```

```
float polomer, obsah;
```

```
/* Proměnná volba je globální, viditelná ve
   funkci obsah i v hlavním programu */
int volba;
int obsah()
{ /* Proměnné x a y jsou lokální ve funkci
   obsah */
   float x, y;
}
int main(int argc, char **argv)
{
/* Proměnná a je lokální viditelná pouze v
   hlavním programu */
   int a;
   int x; /* Lokální x v hl. programu, jiné x
   než ve funkci obsah */
}
```

# Kde je možné v jazyce C deklarovat proměnnou?

- klasické C-čko
  - pouze na počátku bloku, před kódem
- C++
  - kdekoliv, až před použitím

# Platnosti dekrarací

```
int main()  
{ int i;  
  ...  
  for (i=0; i<10; i++)  
  { int x;  
    ...  
  }  
  /* zde je platné i, x již ne */  
}
```

# Překrývání

```
int main()  
{ int x;  
  ...  
  for (i=0; i<10; i++)  
  { int x; /* toto x překryje předchozí  
    ...  
  }  
  /* zde je platné „původní“ x */  
}
```



# Definice nového (uživatelského) typu

- klíčové slovo `typedef`
- jako příklad si ukážeme definici tzv. výčtového typu
  - v příští hodině definici strukturovaného typu

## Problém:

- v programu chceme uchovávat informaci o barvě, např. barvě auta

# Řešení

## 1. prostými číselnými hodnotami

```
int barva;
```

```
barva = 0;
```

```
if (barva == 0)
```

– někde na papír si poznamenanám: 0 znamená černá, 1 znamená červená, ..

- **nejhorší možný způsob**

– o něco lepší řešení: tuto informaci zapíšeš do programu jako komentář

```
/* 0 - černá, 1 - červená */
```

# Řešení

## 2. definice konstant v programu

```
#define CERNA 0
#define CERVENA 1
int barva;
barva = CERVENA;
if (barva == CERVENA)
```

- přehledné řešení

### 3. definice výčtového typu

- na začátku programu nebo do samostatného hlavičkového souboru napíšeš definici deklaraci

```
typedef enum
```

```
{CERNA, CERVENA, BILA} Barvy;
```

- v programu definuješ proměnou typu

```
Barvy
```

```
Barvy barva;
```

```
barva = BILA;
```

```
if (barva == CERVENA)
```

- jak je výčtový typ implementován vnitřně?

```
typedef enum
```

```
{CERNA, CERVENA, BILA} Barvy;
```

– celočíselně, CERNA hodnotou 0, CERVENA hodnotou 1, BILA hodnotou 2

– hodnoty může předepsat programátor

```
typedef enum {CERNA, CERVENA,  
BILA=6} Barvy;
```

- POZOR

```
typedef enum {CERNA, CERVENA,  
BILA, MODRA=1} Barvy;
```

- CERNA hodnotou 0, CERVENA hodnotou 1,  
BILA hodnotou 2, MODRA hodnotou 1

- díky slabé typové kontrole překladač nezastaví překlad chybou

```
barva = 4;
```

- v C přeložitelné
- v C++ dojde k chybě (neplatná konverze z typu int na typ Barva)

# Konstanty

Jak je to v klasickém C s konstantami?

- původně jen symbolické

```
#define MAX 10
```

- v ANSI C a C++ klíčové slovo **const**
  - typované a netypované
  - nelze využít všude (meze polí v C)

```
const int MAX = 10;
```

# Přiřazení:

- přiřazovací výraz:

```
y = 3*a + 12    i=j=1
```

- přiřazovací příkaz

– výraz ukončený ; je příkaz

```
y = 3*a + 12; a = 10; c = 'a';  
barva_auta = BILIA;
```



# Operace:

- aritmetické: +, -, \*, /
  - % - zbytek po celočíselném dělení
  - pozor: dělení celočíselné, neceločíselné
    - typ operace záleží na typu operandů
    - $5/3$ ,  $5/3.0$
    - `int a,b,n; a/b, (double) a/b`
    - $1/n$                        $1.0/n$
  - C/C++ nemá operátor po mocninu, pouze funkci  
`double pow(double b, double exp);`
- bitové: &, |, ~, ^

# Bitové operátory

Operátor	Operace
&	Bitový součin (AND)
	Bitový součet (OR)
^	Exklusivní bitový součet - výlučné NEBO (XOR)
~	Bitová negace
<<	Bitový posuv vlevo
>>	Bitový posuv vpravo

```
unsigned char a = 0x85;
/* 133 desitkove, 10000101 dvojkove */
unsigned char b = 0x46;
/* 70 desitkove, 01000110 dvojkove */
unsigned char c,d,e,f,g,h;
c = a & b;
d = a | b;
e = a ^ b;
f = ~ a;
g = a << 2; // bitový posuv o 2 vlevo
h = b >> 3; // bitový posuv o 3 vpravo
```

# Výsledky bitových operací

10000101    10000101    10000101    10000101

&

|

^

~

01000110

01000110

01000110

---

00000100

11000111

11000011

01111010

# Výsledky bitových operací

10000101    01000110

<< 2       >> 3

---

00010100    00001000

# K čemu je to dobré ?

- určitá funkce vrátí např. 8 bitovou hodnotu a každý bit znamená nějakou chybu; číslo je uloženo v proměnné chyba

00010100



- potřebuji zjistit, zda je tento bit nastaven na 1
- k testu použiji číslo, kterému se říká **maska**:

00000100

00000100 bin = 04 hex

- **if** (chyba & 0x04 != 0)

- unární inkrementace, dekrementace (post, pre): ++, --

`a++; ++a; a--; --a;`

`y = 3*a++; z = 5/--a;`

– detaily viz Cvičení 5 předmětu Programování

- přiřazovací operátor `op=`

`y *= 3; stejné jako y = y*3;`

`y += 3; stejné jako y = y+3;`

`y >>= 2;`

# Příkazy

- každý příkaz ukončen středníkem

- podmínka:

```
if (a < 5) chyba();
```

```
if (x != 0) vypocet(); else err();
```

- relační operátory:

<, <=, >, >=, !=, ==

- logické operátory:

&&, ||, ! - menší priorita než aritmetické



## Pozor:

**if** (a == 5) ... porovnání

**if** (a = 5) ... přiřazení

```
int c;
```

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

```
if (c == '\n') ...
```

```
if ( (c=getchar()) == '\n' ) ...
```

**if** (a) ... porovnání s 0

je to samé jako **if** (a != 0) ...

- **cykly:**

```
while (výraz) příkaz;
```

```
do příkaz; while (výraz);
```

```
do { příkaz1; příkaz2; }  
  while (výraz);
```

```
for (výraz1; výraz2; výraz3) příkaz;
```

ekvivalentní s:

```
výraz1;
```

```
while (výraz2) { příkaz; výraz3; }
```

– nejčastější použití:

```
for (i=1; i<n; i++)
```

– lze i:

```
for (i=1 ; i<n ; i=i+2)
```

- pozor na středník!
- co kdybych napsal čárku?
  - čárka v jazyce C je speciální operátor
- **continue** a **break** v cyklu
  - `break` předčasně ukončí cyklus
  - `continue` ukončí provádění těla a skočí na testovací podmínku
  - mohou být použity ve všech cyklech

```
i = 0;
```

```
while (i < n)
```

```
{
```

```
...
```

```
if (x) continue;
```

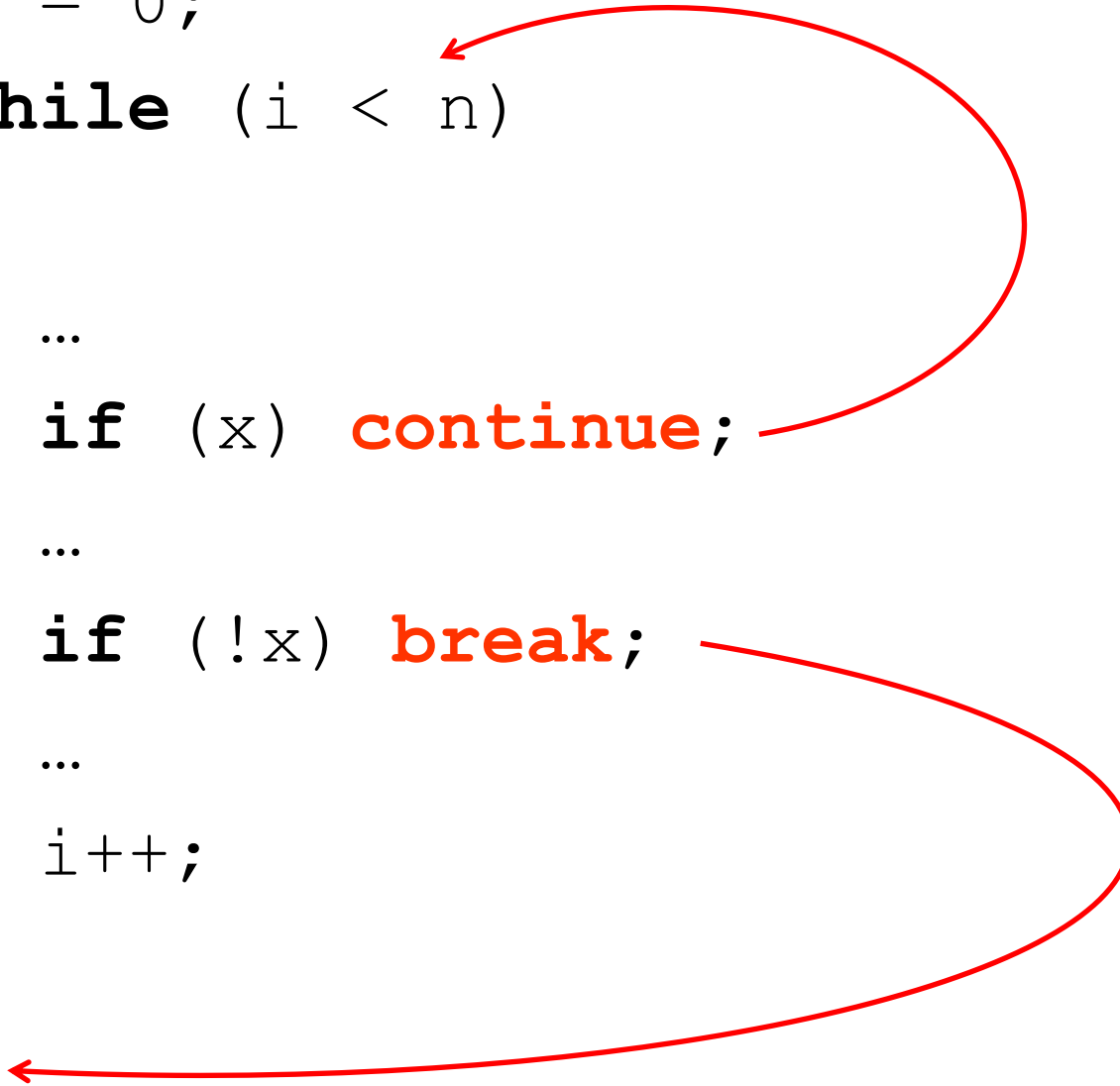
```
...
```

```
if (!x) break;
```

```
...
```

```
i++;
```

```
}
```



Napíšeme program, který přečte ze standardního vstupu (z klávesnice) posloupnost anglických znaků zakončených klávesou Enter. Všechna velká písmena převede na malá a text vytiskne.

```
int main(int argc, char **argv)
{
    int c;
    c = getchar();
    while (c != '\n')
    {
        if (c >= 'A' && c <= 'Z') c += ('a' - 'A');
        putchar(c);
        c = getchar();
    }
    return 0;
}
```

- efektivnější varianta

```
int main(int argc, char **argv)
{
    int c;
    while((c = getchar()) != '\n')
    {
        if (c >='A' && c <= 'Z') c+=('a'-'A');
        putchar(c);
    }
    return 0;
}
```

# Podmíněný výraz

- podmíněný výraz: *ternární operátor*, jehož operandy jsou tři výrazy
- syntaxe podmíněného výrazu:  
výraz1 ? výraz2 : výraz3
  - hodnota podmíněného výrazu je výraz2, pokud má výraz1 nenulovou hodnotu (neboli je pravdivý, true); v opačném případě má hodnotu výraz3
- (a==3) ? 1 : 0  
// je-li hodnota proměnné a rovna 3, podmíněný výraz má hodnotu 1, jinak 0

# Podmíněný výraz

- podmíněný výraz lze s výhodou využít např. při přiřazení:
- `x = (a==3) ? 1 : 0`  
// je-li hodnota proměnné a rovna 3, do proměnné x se přiřadí hodnota 1

## *Poznámky:*

- pomocí podmínky bychom zapsali přiřazení takto:  
`if (a==3) x = 1; else x = 0;`
- první výraz není nutné uzavírat do závorky, ale často se uzavírá kvůli čitelnosti kódu
- podmíněným výrazem nelze nahradit jakoukoliv podmínku!



- varianta s podmíněným výrazem

```
int main(int argc, char **argv)
{
    int c;

    while ( (c = getchar()) != '\n' )
    {
        putchar ( (c >= 'A' && c <= 'Z') ? c + ('a' - 'A') : c );
    }
    return 0;
}
```

- takto program vypadat nemá, ale budeme si demonstrovat názorně použití **break** a **continue**

```
int main(int argc, char **argv)
{
    int c;

    while(1)
    {
        c = getchar();
        if (c=='\n') break;
        if (c >='A' && c <= 'Z')
        {
            putchar(c+('a'-'A')); continue;
        }
        putchar(c);
    }
    return 0;
}
```

# Operátor „čárka“

- operátor čárka představuje jednu ze specialit jazyka C
- oddělují se jím jednotlivé výrazy v "multivýrazu" (dovoluje napsat jakýsi vícenásobný výraz) a použití může nalézt tam, kde syntaxe jazyka dovoluje zapsat pouze jeden výraz

`výraz1, výraz2, výraz3`

- zapsaná trojice výrazů je považována za jeden výraz
- význam:
  - nejprve se vyhodnotí výraz1, pak výraz2 a naposledy výraz3.
  - hodnotou celého výrazu je hodnota posledního vyhodnoceného výrazu

# Operátor „čárka“

*Příklad:*

$$x = a=1, 3, a+b;$$

- nejprve se vyhodnotí přiřazovací výraz, tedy do proměnné  $a$  se přiřadí hodnota 1; hodnota výrazu  $a=1$  je 1, ale tato hodnota se „zahodí“.
- vyhodnotí se další výraz, tedy 3, tato hodnota se opět „zapomene“.
- nakonec se sečtou hodnoty proměnných  $a$  a  $b$ ; součet je hodnotou celého výrazu (protože  $a+b$  je poslední) je přiřazen do proměnné  $x$

# Operátor „čárka“

Prakticky se využívá pouze v cyklu `for`:

```
s=0;  
for (i=1;i<=n;i++) s += i;
```

```
for (s=0, i=1; i<=n; i++) s += i;
```

# Příklad – operátor čárka

Napíšeme program, který do proměnné x načte z klávesnice celé číslo. Pokud je číslo záporné, vytiskne na obrazovku upozornění a do proměnné y přiřadí absolutní hodnotu čísla. Použijte podmíněný výraz a operátor čárky a funkci printf.

*Řešení:*

- uvědomíme se, že printf je funkce a že ji lze použít ve výrazu. Řešení může vypadat např. následovně:

# Příklad – operátor čárka

```
int main(int argc, char **argv)
{
    int x,y;
    scanf("%d",&x);
    y = (x<0) ? printf("Zadal jste zaporne cislo"),-x : x;
    system("PAUSE");
    return 0;
}
```

- **větvení:**

```
switch (x)
```

```
{
```

```
    case 1: příkaz1; break;
```

```
    case 2:
```

```
    case 3: příkaz3; break;
```

```
    default: příkaz4;
```

```
}
```

- pozor na **break**

- **continue** do příkazu **switch** nepatří!



# Konzolový vstup a výstup

- knihovna `stdio.h`
- `printf(const char*format, ...)`
- `scanf(const char*format, ...)`
- pozor u `scanf`: operátor `&`  
`scanf("%d", &x)`

# Používané překladače

- Bloodshed Dev C++
  - <http://www.bloodshed.net/devcpp.html>
- CodeBlocks
  - <http://www.codeblocks.org>
    - pokud nemáte již nainstalovaný překladač, stáhněte si instalátor s MINGW v názvu
      - jinak se Vám nainstaluje pouze vývojové prostředí
- event. MS Visual C++