# Základy jazyka C

Úvod do programování 1

Tomáš Kühr

# Organizační záležitosti

- Konzultace
  - Pracovna 5.043
  - Úterý 12.30 – 14.00
  - Čtvrtek 14.00 – 16.00
  - Pátek 9.30 – 11.30 a od 14.45 dle potřeby (neoficiální)
  - Emailem tomas.kuhr@upol.cz
- Web předmětu http://www.inf.upol.cz/kombinovane-studium/uvod-do-programovani-1
- Sbírka úloh http://jazykc.inf.upol.cz/

# Literatura


- Pavel Herout: Učebnice Jazyka C. Kopp, 2007.
- Brian W. Kernighan, Dennis M. Ritchie: Programovací jazyk C, Computer Press, 2006.
- Reek Kenneth: Pointers on C. Addison Wesley, 1997.
- Robert Sedgewick: Algorithms in C. Addison-Wesley Professional, 2001.
- Jeri R. Hanly, Elliot B. Koffman: Problem Solving and Program Design in C. Addison Wesley, 2006.
- Eric S. Roberts: Programming Abstractions in C. Addison Wesley, 1997.
- Eric S. Roberts: The Art and Science of C. Addison Wesley, 1994.
- a další…

# Zápočet

- Bude udělen za vzorné vyřešení zadaných úloh
- Zadání naleznete ve Sbírce úloh z jazyka C (http://jazykc.inf.upol.cz/)
- Minimální požadovaný počet úspěšně vyřešených úloh k jednotlivým tématům bude uveden na webu předmětu
- Úlohy řešte v maximální možné míře samostatně
- Části vašich řešení, které jste převzali z literatury / webových tutoriálů, náležitě označte (komentářem)
- Podobně ve zdrojovém souboru označte části, na kterých jste spolupracovali s kolegy
- Řešení odevzdávejte emailem nejpozději do příštího semináře
- Za neúplnou nebo pozdě odevzdanou sadu řešení bude potřeba vyřešit 1 úlohu navíc

# Jazyk C

- vytvořen 1972
- autor Dennis Ritchie
- poslední verze C11 (prosinec 2011)
- my si ale vystačíme s ANSI C (1990), příp. C99 (1999)
- inspirace pro mnoho soudobých jazyků
- vlastnosti:
  - nízkoúrovňový
  - překládaný (kompilovaný)
  - platformově nezávislý
  - staticky typovaný
  - procedurální (imperativní) paradigma

# Ukázka zdrojového souboru

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

/*
 * Tento program nedělá nic jiného,
 * než že pozdraví celý svět.
 */

int main() { // hlavní funkce programu

    /* sem budeme zatím psát všechny příkazy */

    printf("Ahoj, světe!\n"); // napíše text do konzole

    return 0; // úspěšný konce programu
}
```

# Vývojová prostředí

- MS Visual Studio / MS Visual C++ (MS Windows)
- Code::Blocks (multiplatformní)
- GNU Emacs (GNU/Linux)
- Xcode (OS X)
- Libovolný textový editor

# Několik poznámek k syntaxi

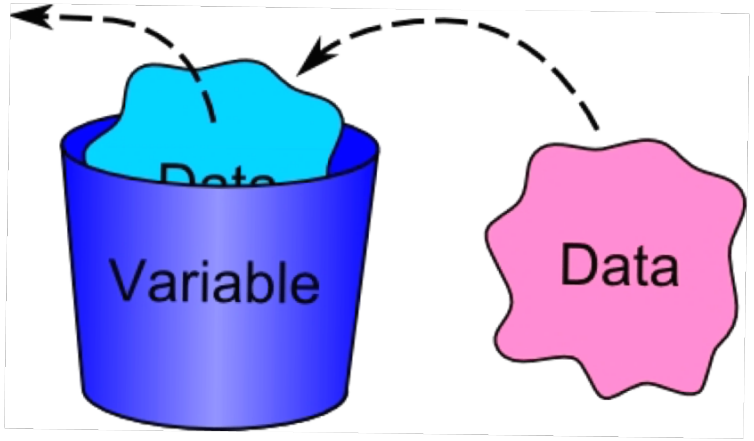
- C rozlišuje velká a malá písmena (case sensitivity)
  `system ≠ System ≠ SYSTEM`
- C ignoruje většinu bílých znaků (mezer, tabulátorů, odřádkování)
  `(2*pi*r*(r+v))` je stejné jako `(2 * pi * r * (r+v))`
- ale naopak
  `"Ahoj světe!" ≠ "Ahoj    světe!"`
  `int cislo; ≠ intcislo;`
- Příkazy jsou ukončeny středníkem

```
povrch = (2 * pi * r * (r+v));
printf("Ahoj světe!");
```

# Proměnná

- Pojmenované místo v paměti
- Lze v ní uchovávat 1 hodnotu
- Předem daného datového typu
- Vytvoření proměnné:

```
typ identifikátor;
typ identifikátor = hodnota;
```




- Příklady:

```
int cislo;
double desetinne_cislo = 3.14;
char pismeno = 'A';
int logickaHodnota = 0;
```

# Identifikátor proměnné

- Jedinečný v rámci daného bloku / programu
- Může obsahovat písmena, číslice a podtržítko
- Musí začínat písmenem (nebo podtržítkem)
- Identifikátor by měl být smysluplný
  „Program častěji čteme než píšeme!“
  `obsah = pi * polomer * polomer;` vs. `xyz = ahoj * abc * abc;`
- Nedoporučuji používat háčky a čárky
- Doporučuje se underscore_case, případně camelCase

```
double polomer_kruznice;
double pi = 3.14;
int idHlavnihoUzivateleVAplikaci;
```

# Základní datové typy

- char - obvykle pro znaky (1 byte, rozsah typicky 0 až 255)
- int - celé číslo (velikost přirozená pro danou platformu)
- float / double - desetinná čísla (s pohyblivou řadovou čárkou) (double má dvojnásobnou přesnost)
- Na typ int lze aplikovat kvalifikátory long a short:

```
short int prumerna_mzda = 25000;
int poslanecky_plat = 56000;
long int majetek_billa_gatese = 1180000000000;

short prumerna_mzda = 25000;
long majetek_billa_gatese = 1180000000000;
```

- Kvalifikátor long lze aplikovat na double:

```
long double velke_desetinne_cislo = 0.12345678902345678 93;
```

# Znaménkové a neznaménkové typy

- Na celočíselné typy (int, char, ale i short, long) lze aplikovat modifikátory signed a unsigned
- unsigned - reprezentace pouze nezáporných čísel
- signed - čísla mohou být kladná i záporná
- Stejná velikost paměti
  - Rozsah signed char je od -128 do 127.
  - Rozsah unsigned char je od 0 do 255.

# Logické hodnoty v C

- Logické hodnoty pravda a nepravda reprezentujeme v C čísly
- Není zde speciální datový typ
- Nula = nepravda
- Nenulové číslo (typicky 1) = pravda

# Číselné konstanty

- Celočíselné konstanty zapisujeme číslicemi
- Záporní čísla odlišujeme znaménkem mínus
- Implicitně mají typ int (velké hodnoty long int)
- Desítkové: 14, 16U, -12, 5145L, 58UL
- Osmičkové: 07, 0245, 0123
- Šestnáctkové: 0xABC, 0x0, 0x12B
- U desetinných čísel používáme tečku (ne čárku)
- Implicitně jsou tyto konstanty typu double
- Desetinné: 12.34, 15.47F, 57.478L, 1e+6, 3.14e-3

# Konstantní znaky a řetězce

- Znakové konstanty zapisujeme do apostrofů
- Znakové: 'a', '9', '\'', '\n'
- Řetězcové konstanty zapisujeme do úvozovek
- Řetězce: "Ahoj", "124", "\"Ahoj\"", "Prvni\nDruhy\nTreti\n"
- Textové řetězce jsou posloupnosti znaků ukončené '\0'

# Výstup na obrazovku

- Pomocí funkce printf z knihovny stdio
- Obecně:
  `printf(řídící_řetězec, hodnota1, hodnota2, ...);`
- Příklady:

```
int x = 5, y = 3;
int sum = 8;
int cislo = 23;

printf("Součet je 8");
printf("Součet je %d", 8);
printf("Součet je %d", sum);
printf("Součet je %d", x + y);
printf("Součet je %d\t Součin je %d\n", x + y, x * y);
printf("Plán jsme splnili na 100 %%.");
printf("Dekadicky %d je oktalově %o a hexadecimálně %x.\n",
       cislo, cislo, cislo);
```

# Vstup z klávesnice

- Pomocí funkce scanf z knihovny stdio
- Obecně:
  `scanf(řídící_řetězec, &promenna1, &promenna2, ...);`
- Příklady:

```
int cislo, cislo2;
unsigned int hexa_cislo;
unsigned int den, mesic, rok;

scanf("%d", &cislo);
scanf("%x", &hexa_cislo);
scanf("%d %d", &cislo, &cislo2);
scanf("%d.%d.%d", &den, &mesic, &rok);
```

# Možnosti formátovacího řetězce

- %c – výpis nebo načtení znaku
- %d (%i) – celé číslo desítkově znaménkově
- %u – celé číslo desítkově neznaménkově
- %o – celé číslo osmičkově
- %x (%X) – celé číslo šestnáctkově (malá / velká písmena)
- %f – desetinné číslo (float a double při výpisu)
- %lf – načtení desetinného čísla typu double
- %e (%E) – desetinné číslo semilogaritmicky
- %g (%G) – jako %f nebo %e (%E) podle hodnoty čísla
- %s – textový řetězec

# Operátory a jejich vlastnosti

- Základní konstrukce (skoro) každého jazyka
- Z daných operandů vytvoří výsledek, který je možné dále využívat
- Arita – udává počet operandů (vstupů)
- Některé operátory mají i tzv. vedlejší efekt
- Příklady výrazů s operátory:
  - `a+b` (binární sčítání)
  - `a-b` (binární odčítání)
  - `-a` (unární mínus)
  - `a<=b` (binární menší nebo rovno)
  - `a--` (unární dekrementace)
  - `prumer = (a+b)/2;` (přiřazení, sčítání a dělení v 1 výrazu)

# Priorita a asociativita operátorů

- **Priorita** určuje pořadí, ve kterém se operátory vyhodnocují
- Znáte ji vlastně už z matematiky
- `1 + 2 / 3 + 4` = ?
- Vyhodnocování výrazu lze ovlivnit použitím (kulatých) závorek
  `(1 + 2) / (3 + 4)`  `1 + (2 / 3 + 4)`  `(1 + 2) / 3 + 4`
- **Asociativita** udává směr, ve kterém se vyhodnocují binární operátory se stejnou prioritou
- Zleva nebo zprava
- `1 - 2 - 3 - 4` = ?
- Opět lze ovlivnit závorkami
  `((1 - 2) -3) - 4`  `(1 - 2) - (3 - 4)`  `1 - (2 - (3 - 4))`

# Aritmetické operátory

- Unární operátory `+` a `-`
- Binární operátory `+`, `-`, `*` a `/` s obvyklým významem
- Asociativita zleva
- Příklady:

```
stejne_cislo = +cislo;
opacne_cislo = -cislo;
delka = vetsi - mensi;
prumer = (prvni + druhy + treti) / 3;
obsah = 2 * pi *polomer * polomer;
```

- Binární operátor `%` pro určení zbytku po celočíselném dělení (modulo)
- Příklad:

```
int delenec = 13, delitel=5, podil, zbytek;
podil = delenec / delitel;
zbytek = delenec % delitel;
```

# Aritmetické operátory

- Pozor na typy operandů
- Pokud jsou všechny operandy celočíselné, je i výsledek celočíselný
- Jinak je výsledek reálné číslo
- Například při dělení toto může ovlivnit i hodnotu výsledku

```
int cislo = 15;
double polovina;
polovina = cislo / 2;
```

- Lze obejít přetypováním operandu nebo desetinným zápisem konstanty

```
polovina = (double)cislo / 2;
polovina = cislo / 2.0;
```

# Přiřazení

- Pomocí operátoru =
- Zápis ve tvaru *identifikátor_proměnné = jakýkoli_výraz*
- Příklady použití

```
cislo = 15;
druhe_cislo = 2*cislo;
```

- Vedlejší efekt – do proměnné uvedené vlevo uloží výsledek výrazu vpravo
- Asociativní zprava

```
prvni = druhy = treti = 42;
(prvni = (druhy = (treti = 42)));
```

- Další přiřazovací operátory

```
+=  -= *= /= %=
```

atd.

- Význam

```
cislo += 5;     cislo = cislo+5;
```

# Inkrementace a dekrementace

- Aritmetické operátory, které v matematice nemáme
- Mají vedlejší efekt
- Inkrementace (++) zvyšuje hodnotu operandu o 1

```
cislo++;      cislo+=1;      cislo = cislo+1;
```

- Dekrementace (--) snižuje hodnotu operandu o 1

```
cislo--;      cislo-=1;      cislo = cislo-1;
```

- Mohou být v tzv. prefixovém nebo postfixovém tvaru
- Prefixové a postfixové použití se odlišuje výsledkem

```
int vysledek1, vysledek2;
int cislo = 5;
vysledek1 = cislo++;
vysledek2 = ++cislo;
```

- Nepoužívejte přiřazení, inkrementaci a dekrementaci ve složitějších výrazech

```
y = (x++ - 5 + (z = y - 2));
```

# Podmínkové operátory

- Operátory pro porovnávání
  `<` `>` `<=` `>=` `==` (rovná se) `!=` (nerovná se)
- Binární operátory
- Výsledkem je logická hodnota (nula vs. nenulové číslo)
- Příklady:

```
prvni >= druhe + treti
cislo != - cislo
dalsi_cislo == cislo + 10
```

- Typické chibyčki:

```
prvni >= druhe > treti
10 < cislo < 20
```

# Logické operátory

- Slouží pro konstrukci složitějších podmínek
- K dispozici máme operátory
  - konjunkce (a zároveň) &&
  - disjunkce (nebo) ||
  - negace !
- Asociativita zleva
- Příklady:

```
(10 < cislo) && (cislo < 20)
(cislo <= 10) || (20 <= cislo)
!((10 < cislo) && (cislo < 20))
```

| a | b | a && b | a \|\| b | !a |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 |

# Líné vyhodnocování

- Při vyhodnocování logických operátorů (konjunkce a disjunkce)
- Vyhodnocuje se pouze část výrazu nezbytná pro určení výsledku
- Příklady:

```
(2 < 1) && (10 < cislo/0)
(10 <= cislo) && (cislo <= 20)
(10 < 12) || cokoli
(10 < cislo) || (cislo <= 0)
```

- Nepoužívejte v podmínkách operátory s vedlejším efektem (přiřazení, inkrementace, dekrementace)

# Přetypování

- Změna typu výrazu na jiný typ
- Obecně:
  *(nový_typ)výraz*
- Příklady:

```
int delenec = 5, delitel = 2;
double podil;
podil = (double)delenec / delitel;
```

- Existuje i implicitní přetypování:

```
double cislo = 3.123;
int cela_cast;
cela_cast = cislo;
```

# Podmínkový operátor

- Slouží pro větvení uvnitř výrazu
- Při složitějším použití nepřehledný
- Jediný ternární operátor
- Obecně:
  *podmínka* ? *výraz_splněno* : *výraz_nesplněno*
- Pokud platí *podmínka*, pak se operátor vyhodnotí na *výraz_splněno*, jinak na *výraz_nesplněno*.
- Příklady:

```
int cislo1, cislo2;
int min;
...
min = (cislo1 < cislo2) ? cislo1 : cislo2;
```

# Čárka

- Slouží pro vytváření sekvencí
- Smysluplný téměř jen v řídících řádcích cyklů
- Obecně:
  *výraz_1, výraz_2*
- Vyhodnotí se *výraz_1*, výsledek se zapomene, vyhodnotí se *výraz_2* a jeho výsledek udává výsledek celé sekvence
- Výrazy obvykle mívají nějaký vedlejší efekt
- Příklady:

```
int cis1 = 5, cis2 = 2;
int min, max, por;
...
por = (cis1 < cis2) ? (min = cis1, max = cis2, 1) :
                      (min = cis2, max = cis1, -1);
```

# Přehled operátorů

| Priorita | Operátory | Asociativita | Arita |
|---|---|---|---|
| 1 | `() [] -> .` | | |
| 2 | `! ~ ++ -- + - (typ) * & sizeof` | | unární |
| 3 | `* / %` | zleva doprava | binární |
| 4 | `+ -` | zleva doprava | binární |
| 5 | `<< >>` | zleva doprava | binární |
| 6 | `< > <= >=` | zleva doprava | binární |
| 7 | `== !=` | zleva doprava | binární |
| 8 | `&` | zleva doprava | binární |
| 9 | `^` | zleva doprava | binární |
| 10 | `\|` | zleva doprava | binární |
| 11 | `&&` | zleva doprava | binární |
| 12 | `\|\|` | zleva doprava | binární |
| 13 | `? :` | zprava doleva | ternární |
| 14 | `= += -= *= /= %= >>= <<= &= \|= ^=` | zprava doleva | binární |
| 15 | `,` | zleva doprava | binární |

# Operátory – cvičení 5

Vytvořte program, který převede velikost úhlu ve stupních na velikost úhlu v radiánech. Program pořádně otestujte…

# Operátory – cvičení 6

Vytvořte program, který převede finanční částku v amerických dolarech na odpovídající částku v českých korunách. Pro převod použijte aktuální kurz České národní banky…

# Operátory – cvičení 7

Vytvořte program, který načte velké písmeno a vypíše odpovídající malé písmeno.

Příklad použití:

```
Zadejte velke pismeno: C
Male pismeno je: c
```

# Operátory – cvičení 8

Vytvořte jednoduchou textovou kalkulačku. Program se uživatele zeptá na první operand a poté na druhý operand. Následně provede výpočet operací +, -, *, /  a vypíše výsledky do konzole.

Příklad použití:

```
Zadejte prvni cislo: 14
Zadejte druhe cislo: 3

Soucet je: 17
Rozdil je: 11
Soucin je: 42
Podil je: 4.6667
```

# Operátory – cvičení 9

Vytvořte textovou kalkulačku. Program se uživatele zeptá na první operand, poté na znaménko operace a nakonec na druhý operand. Následně provede výpočet a vypíše výsledek do konzole.

Příklad použití:

```
Zadejte prvni cislo: 14
Zadejte operaci: +
Zadejte druhe cislo: 3

Vysledek je: 17
```