Procedurální programování

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 01

B0B36PRP – Procedurální programování

Přehled témat

- Část 1 – Organizace předmětu
  - Cíle předmětu
  - Prostředky dosažení cílů PRP
  - Hodnocení předmětu a zkouška
- Část 2 – Zadání 0. domácího úkolu (HW00)
- Část 3 – Programování v C
  - První program (v C)
  - Základních koncepty programování

S. G. Kochan: kapitoly 2, 3

- Část 4 – Zadání 1. domácího úkolu (HW01)

Část I

Organizace předmětu

Předmět a přednášející

B0B36PRP – Procedurální programování

- Webové stránky předmětu
  `https://cw.fel.cvut.cz/wiki/courses/b0b36prp`
- Odevzdávání domácích úkolů
  `https://cw.felk.cvut.cz/brute`
- Přednášející:

  - prof. Ing. **Jan Faigl**, Ph.D.
    - Katedra počítačů – `https://cs.fel.cvut.cz`
    - Centrum umělé inteligence – **Artificial Intelligence Center** (**AIC**)
      `https://aic.fel.cvut.cz`
    - Centrum robotiky a autonomních systému
      Center for Robotics and Autonomous Systems – CRAS
      `https://robotics.fel.cvut.cz`
    - **Laboratoř výpočetní robotiky** (**Computational Robotics Laboratory**)
      `https://comrob.fel.cvut.cz`

Cíle předmětu

- Osvojit si pohled na výpočetní prostředky jako „*počítačový vědec*“ a naučit se je efektivně používat.

  Computer scientist
  - Formulovat problém a jeho řešení počítačovým programem.
  - Získat povědomí jaké problémy lze výpočetně řešit.
- Získat zkušenost s programováním

  získání vlastní zkušenosti
  - Programování v C

  cvičení, domácí úkoly, zkouška
- Osvojit si schopnost číst, psát a porozumět malým programům
- Získat programovací návyky jak psát
  - Srozumitelné a přehledné zdrojové kódy;
  - Opakovaně použitelné programy.

Výuka programování

„Separating Programming Sheep from Non-Programming Goats“

http://blog.codinghorror.com/separating-programming-sheep-from-non-programming-goats

http://www.eis.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf

- Efektivní metody výuky programování se hledají již od dob prvních počítačů

  Více než 50 let
- Přesto se zdá, že je každý základní kurz programování obtížný a 30 % až 60 % studentů jej na poprvé nezvládne.

  Průchodnost PRP je výrazně vyšší.

  2021/2022: 79 % (92 % z udělených zápočtů)

  2020/2021: 73 % (93 % z udělených zápočtů)

  2019/2020: 64 % (96 % z udělených zápočtů)
- **Základní koncept je pochopení principu přiřazení hodnoty proměnné!**

  Je to především práce s pamětí, která je v Cčku velmi transparentní.

Princip přiřazení

- Zápis programu pro přiřazení hodnot do proměnných *a* a *b* a následné přiřazení proměnné *b* do *a*.

Přiřazení hodnoty proměnné

```
1 int a = 10;
2 int b = 20;
3
4 a = b;
```

- Jaké jsou hodnoty proměnných *a* a *b*?

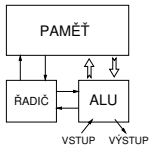
a. a = 20, b = 0
b. a = 20, b = 20
c. a = 0, b = 10
d. a = 10, b = 10
e. a = 30, b = 20
f. a = 30, b = 0
g. a = 10, b = 30
h. a = 0, b = 30
i. a = 10, b = 20
j. a = 20, b = 10

Program vlastně „jen“ přesouvá a upravuje číselné hodnoty v paměti na základě definovaných podmínek!

Počítač „počítá“, tedy pracuje s čísly

- Výpočet je realizován *aritmeticko-logickou jednotkou* (ALU).
- Číselné hodnoty jsou uloženy v paměti počítače.
- Předpis jak a co počítat je zapsán programem.

  Opět jako posloupnost číselných hodnot se specifickým významem.



- Základní jednotkou uložení informace v paměti počítače je bit (binární 0 nebo 1)

  Historicky vychází z děrného štítku - zápis a strojové zpracování informací
- ALU pracuje s vyhrazenou pamětí, např. součet dvou hodnot 10 + 4 může být realizován registry nebo akumulátorem.

registry

```
mov $10, %r1
mov $04, %r2
add r1, r2, r3
```

akumulátorem

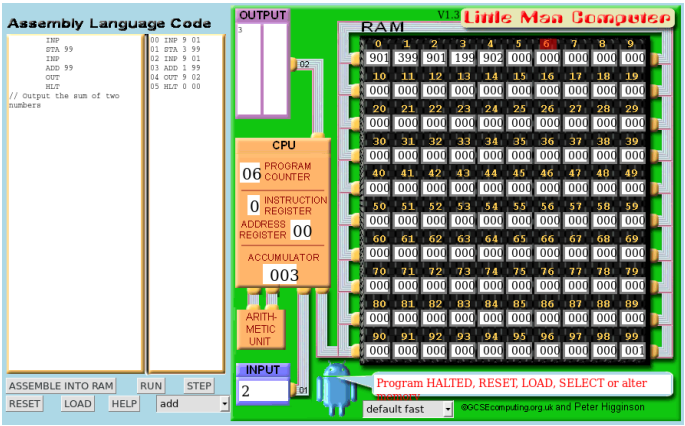
```
lda $10
add $04
sto r3
```

Každá instrukce má svůj příslušný zápis jako číselná hodnota (opcode), program je tak posloupnost číselných hodnot.

Princip výpočtu

- Pochopení principu výpočtu na simulátoru procesoru, např. Little Man Computer.

https://peterhigginson.co.uk/LMC/, http://www.vivaxsolutions.com/web/lmc.aspx



Základní instrukce:
LDA – Load to the acc.;
STA – Store the acc. to address;
ADD – Add to the acc.;
INP – Input to the acc.;
OUT – Output of the acc.;
BRP – Set PC on zero or possitive acc.;
HLT – Stop executing program.

https://www.youtube.com/watch?v=6cbJWV4AGmk

Způsob reprezentace znalostí

- Z hlediska výpočtu můžeme rozlišit dva základní typy znalostí.

Způsoby popisu problému.

Deklarativní

- Tvrzení popisující stav.
- Axiomatické.
- Umožňuje jednoduše ověřovat (testovat) pravdivost tvrzení.
- Neposkytuje návod jak hodnotu vypočítat.

$$\sqrt{x} = y, y^2 = x, x \geq 0, y \geq 0.$$

Imperativní

- Popis jak něco vypočítat.
- Posloupnost výpočtu.
- Test jak ovlivnit průběh výpočtu.

1. If $y^2 \approx x$
2. Then
   return y
3. Else
   $y \leftarrow \frac{y+\frac{x}{y}}{2}$
   Go to Step 1

Program je „recept“

- Program je „recept“ – posloupnost kroků (výpočtů) popisující průběh řešení problému.
- Programování je schopnost **samostatně**
  - **tvořit programy**;
  - **dekomponovat** úlohy na menší celky;
  - sestavovat z **dílčích částí větší programy** řešící komplexní úlohu.

B0B36PRP – je příležitostí, jak se těmto schopnostem naučit!

Organizace a hodnocení **B0B36PRP** – Procedurální programování

- Rozsah: 2p+2c; Zakončení: Z,ZK; Kredity: 6; 1 ECTS kredit je 25–30 hodin za semestr, cca 180 h.
  - Kontaktní část (přednáška a cvičení): 3 hodiny týdně, tj. 42 hodin celkem.
  - Zkouška včetně přípravy: *10 hodin*.
  - Domácí příprava (úkoly) cca **9 hodin týdně**. *Medián zátěže*
- **Průběžná práce v semestru** – domácí úkoly a test.
- **Zkouškový test a implementační zkouška.** *Schopnost samostatné práce na počítačích v učebnách.*
- Docházka na **cvičení** a odevzdání domácích úloh. *Samostatná práce (kontrola plagiátů).*
  - Postupujte systematicky, budete tak postupně rozvíjete své schopnosti.
  - Využijte čas v prvních úlohách a naučte se psát programy správně.

  Program musí být nejen správný a funkční, ale také čitelný a udržovatelný!
- Konzultace – Pokud nevíte, tápete nebo řešíte domácí úkol příliš dlouho, **konzultujete** s cvičícím/přednášejícím. *Čtěte (učebnici), pochopte principy (nejen hledat řešení), hlídejte si čas a včas konzultujete!*
  - **Maximálně využijte kontaktní čas na cvičení/přednášce, ptejte se, diskutujete!**
- *„Alternativní“* absolvování předmětu pro velmi zkušené – **Seminář ACM** - předmět A4B36ACM!

Zdroje a literatura

- **Knihy (učebnice)** *Základní učební text „Programming in C" (Kochan, 2014)*

Programming in C, 4th Edition, *Stephen G. Kochan*, Addison-Wesley, 2014, ISBN 978-0321776419

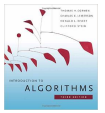
C Programming: A Modern Approach, 2nd Edition, *K. N. King*, W. W. Norton & Company, 2008, ISBN 860-1406428577

The C Programming Language, 2nd Edition (ANSI C), *Brian W. Kernighan, Dennis M. Ritchie*, Prentice Hall, 1988 *1st edition 1978*

- Přednášky – podpora učebního textu, slidy, videa, poznámky a **vlastní poznámky**.
  Součástí přednášek jsou také zdrojové kódy s příklady!
- Cvičení – získání praktických dovedností řešením domácích úkolů a dalších úloh.
  programovat, programovat, programovat

Další zdroje

Introduction to Algorithms, 3rd Edition, *Cormen, Leiserson, Rivest, and Stein*, The MIT Press, 2009, ISBN 978-0262033848



Algorithms, 4th Edition, *Robert Sedgewick, Kevin Wayne*, Addison-Wesley, 2011, ISBN 978-0321573513

The C++ Programming Language, 4th Edition (C++11), *Bjarne Stroustrup*, Addison-Wesley, 2013, ISBN 978-0321563842

Způsob výuky programování v B0B36PRP

- Naší snahou je vybudovat zkušenost a rozvinout dovednost programování.
  - Programování vs. algoritmizace;
  - Programování je „řemeslo", jak správně implementovat nějaký algoritmus.
  - **Jen funkční nestačí - program musí být i správně!** *Očekávaný vstup vs. co všechno může uživatel na vstup zadat.*
- Studijní zátěž je proto rozložena do výukové části semestru.
  - Úkoly na cvičení a termíny domácích úkolů.
- Systematické rozvíjení dovednosti programování v průběhu semestru považujeme za zásadní.
  Typicky je na začátku semestru čas pro pochopní principů (čtení učebnice)!
- Bez znalosti konstruktů a základní příkazů nelze efektivně programovat nelze.
  Neávislost na našeptávači!
  - Začínáme relativně jednoduchými úlohami k osvojení programovacích konstruktů a způsobu organizace zdrojového kódu. *Přehledný kód a schopnost se efektivně orientovat v kódu je zásadní!*
  - *Úkoly jdou vždy realizovat s tím, co si řekneme na přednášce/cvičení.*
    Řešení s pokročilejšími konstrukty může být elegantnější(kratší), nemusí však dodat potřebný vhled.
  - V prvních přednáškách pokrýváme nezbytné znalosti, které jsou dále prohlubovány.
    - Cvičení doplňují přednášky a dávají více prostoru pro praktické osvojení problematiky.
- Můžete volit praktický způsob vstřebávání znalosti programování z příkladů, který je vhodný doplnit teoretickou přípravou z učebnic(e).

Přednášky – zimní semestr (ZS) akademického roku 2022/2023

- Harmonogram akademického roku 2022/2023.
  `https://www.fel.cvut.cz/cz/education/harmonogram2223.html`
- Přednášky
  - Dejvice, místnost T2:D3-309, středa, 16:15–17:45.
- 14 výukových týdnů. *12+1 přednášek*
- Státní svátek – 28.09.2022 (středa).
- Konzultační cvičení: pátek 10:00-11:30 na kampusu na Karlově náměstí (KN).

Cvičící

- RNDr. **Ingrid Nagyová**, Ph.D.

- Ing. **Jindřiška Deckerová**

- Bc. **Jakub Dupák**

- Bc. **Petra Fridrichová**

- Ing. **Jakub Sláma**
  BRUTE

- Ing. **Rudolf Jakub Szadkowski**
  Detekce plagiátů

Komunikace související s PRP

- Obracejte se na svého cvičícího dle cvičení, na které chodíte (jste přihlášeni).
  Případně na **prp-teachers@fel.cvut.cz**
- Komunikovat můžete elektronickou poštou (e-mail).
  - Pište ze své fakultní adresy (odesílatel).
  - Do předmětu zprávy uvádějte zkratku předmětu PRP.
  - Kopii zprávy (Cc) posílejte též příslušnému vedoucímu cvičení (dle studijního programu).
  - V případě zásadních problému (např. týkajících se zápočtu) uvádějte do Cc též přednášejícího.
- Případně můžete využít discord kanálu. Time management - nečekejte okamžitou odpověď.

Pracujte soustředěně a užijte si tvůrčí zápal.

Vývojové prostředí

- Počítačové laboratoře - Ubuntu se síťovým bootováním a domovskými adresáři (NFS v4).
  Přenos a synchronizace souborů – ownCloud, SSH, gdrive, sharepoint
- Doporučený operační systém - Ubuntu-based, Pop OS!, Win s WSL(2) ideální s nativní VS Code.
  Přímočará instalace potřebných programů
- Překladače **gcc** a **clang**, sestavení **make** (GNU make) `https://gcc.gnu.org` a `http://clang.llvm.org`
- **Visual Studio Code** (VSC) - `https://code.visualstudio.com/`
- Editor – **gedit**, `atom`, **sublime**, **vim** – `https://atom.io/`, `http://www.sublimetext.com/`
  `http://www.root.cz/clanky/textovy-editor-vim-jako-ide`
  Pokud programovat umíte, investuje čas do efektivního ovládání editoru, např. **vim**
- C/C++ vývojová prostředí – **WARNING: Do Not Rely on an IDE**
  `http://c.learncodethehardway.org/book/ex0.html`
  - **CLion**, `NetBeans 8.0 (C/C++)`, `Eclipse-CDT` – `https://www.jetbrains.com/clion`
  - `Geany`, `Code::Blocks`, `CodeLite` `https://www.geany.org/`, `http://www.codeblocks.org`, `http://codelite.org`
  - **Nejdříve porozumějte principům**, nakonfigurujte nástroje a programování zefektivněte.
- Odevzdávání domácích úkolů BRUTE `https://cw.felk.cvut.cz/brute`.
  BRUTE – Bundle for Reservation, Uploading, Testing and Evaluation

Služby akademické sítě – FEL, ČVUT

- `http://www.fel.cvut.cz/cz/user-info/index.html`
- Diskové úložiště ownCloud – `https://owncloud.cesnet.cz`
- Zasílání velkých souborů – `https://filesender.cesnet.cz`
- Rozvrh a termíny – FEL Portal – `https://portal.fel.cvut.cz`
- FEL Google Account – autentizovaný přístup do `Google Apps for Education`
  Více viz `http://google-apps.fel.cvut.cz/`
- Gitlab FEL – `https://gitlab.fel.cvut.cz/`
- Přístup k informačním zdrojům (IEEE Xplore, ACM, Science Direct, Springer Link)
  `https://dialog.cvut.cz`
- Akademické a kampusové licence `https://download.cvut.cz`
- Národní Gridová Infrastruktura MetaCentrum `http://www.metacentrum.cz/cs/index.html`
- RCI Cluster `https://login.rci.cvut.cz`

Domácí úkoly a další úlohy

- Samostatná práce s cílem osvojit si praktické zkušenosti.
- Jednotné zadání na přednášce a jednotný termín odevzdání.
- Odevzdání domácích úkolů prostřednictvím BRUTE.

  `https://cw.felk.cvut.cz/brute`

  - Nahrání (upload) archivu s nezbytnými zdrojovými soubory.
  - Ověření správnosti implementace automatickými testy.
  - Penalizace za překročení počtu uploadů.

    Odevzdávejte funkční kódy, nikoliv „pouze“ kódy, které projdou testy!
  - Detekce plagiátů

    *Cílem řešení úkolů je **získat vlastní zkušenost***
- Úkoly jsou navrhovány tak, aby byly stihnutelné. Plánujte a hlídejte si čas, včas konzultujte.
- Klíčem k úspěšnému dokončení předmětu je samostatná práce a osvojení si technik a znalostí

  Průběžná práce a řešení úkolů
- Pokud něčemu nerozumíte, **ptejte se**!

  Pokud chybujete, tak se učíte, pokud nechybujete, tak už to umíte!

Přehled domácích úkolů

- Domácí úkoly s povinným, volitelným, případně bonusovým zadáním. *47 h, bonus +23 h*

  `https://cw.fel.cvut.cz/wiki/courses/b0b36prp/hw/start`

  0. HW 00 - První program *1 h*
  1. HW 01 - Načítání vstupu, výpočet a výstup *1 h*

     Seznámení se s prostředím, psaním programů, jejich laděním, testováním a odevzdáváním. ∼ 20–40 h
  2. HW 02 - První cyklus (**Kontrola přehlednosti kódu** – až -100% z dosažených bodů) *2 h*
  3. HW 03 - Kreslení (ASCII art) (**Kontrola kódu** – až -100%) *3 h*
  4. HW 04 - Prvočíselný rozklad (**Kontrola kódu** – až -100%) *5 h, bonus +8 h*
  5. HW 05 - Caesarova šifra (**Kontrola kódu** – až -100%) *6 h*
  6. HW 06 - Maticové počty (**Kontrola kódu** – až -100%) *7 h bonus +7 h*
  7. HW 07 - Hledání textu v souborech *5 h*
  8. HW 08 - Kruhová fronta v poli - *Dynamicky linkovaná knihovna* *5 h*
  9. HW 09 - Načítání a ukládání grafu *5 h*
  10. HW 10 - **Integrace** načítání grafu a prioritní fronta v úloze hledání nejkratších cest

      HW 09 + 12. přednáška, soutěž na extra body *8 h bonus +8 h*
- Podmínkou zápočtu je úspěšné odevzdání všech domácích úkolů.
- Odevzdání volitelného zadání je doporučeno (není částečné odevzdání).

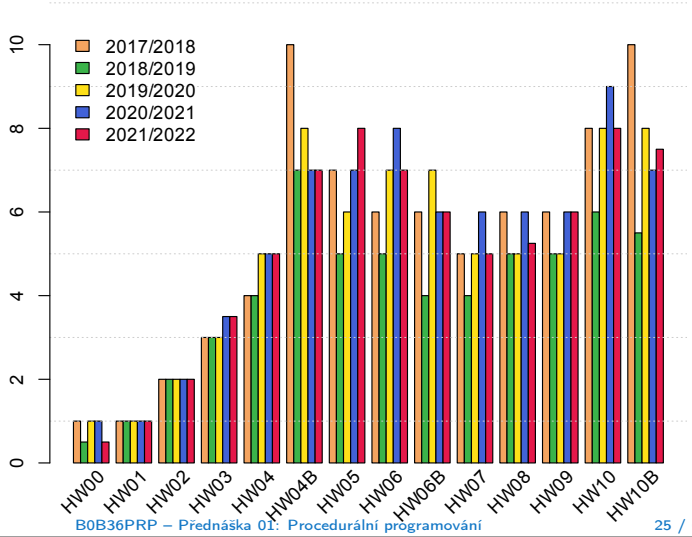
  Celkové body za povinné zadání **25b**, volitelné zadání **20b**, bonusové **10b**$^+$

Medián reportované časové náročnosti vypracování úloh

- Součet mediánů cca 40–54 h celkem (bez bonus úloh).
- Bonusové úlohy cca 16–26 h.
- Celkem cca 56–76 h.
- Očekávaná náročnost cca 70 h.
- Od HW04 jsou úlohy přibližně stejně časově náročné.
- V prvních týdnech věnujte čas seznámení se s prostředím trénování kódování (přepisujte a zpřehledňujte) a čtení učebnice.

  Řešení pozdějších úloh pak může být rychlejší.
- **Časová dotace cca 180 h**.

  42 h kontaktní část, 10 h zkouška, 70 h úlohy, *"rezerva" cca 58 h*.



Odevzdávání domácích úkolů – BRUTE

- **BRUTE** – Bundle for Reservation, Uploading, Testing and Evaluation
  - Formální kontrola – kompilace programu.
  - Testování funkčnost a správnosti – **kontrola výstupu pro daný vstup**.
    - Veřejné vstupy a odpovídající výstupy / neveřejné vstupy.
  - Před uploadem programu si program otestujete sami.
    - S využitím dostupných vstupů a výstupů.
    - Vytvoření vlastních vstupů a laděním programu.
    - Vytvoření vstupů **přiloženým generátorem vstupů**.
    - Ověření výstupu **přiloženým testovacím nebo referenčním programem**.
- Porozumění kódu a kontrola možných stavů.
  - **Pro každý řádek byste měli být schopni odpovědět proč tam je a co dělá!**
  - Pro **každou funkci nebo načtení vstupu** od uživatele analyzujte možné vstupní hodnoty nebo **návratové hodnoty funkcí**!
    - Pokud je z hlediska funkčnosti vstup nebo návratová hodnota zásadní, **proveďte kontrolu vstupu a/nebo příslušnou akci**, např. vypsání hlášení a ukončení programu.

      Např. očekávaný vstup je číslo a uživatel zadá něco jiného.

Úkoly a **BRUTE**

- Úkoly nejsou jen o odevzdání implementace, která projde BRUTE testy.
- Úkoly jsou především o **postupném získání zkušeností** s konkrétními konstrukty.
- Úkoly mají relativní obtížnost velmi podobnou.
  - Je důležité postupně samostatně řešit jednotlivé úkoly a osvojovat si dílčích dovednosti.

Absolutně jsou úlohy postupně náročnější a náročnější!

- Netrapte se s řešením příliš dlouho sami, ptejte se na fóru, cvičících, přednášce, **konzultaci**.
- Úlohy HW02–HW06 budou manuálně kontrolovány na správnost, přehlednost a čistotu kódu.
  - Zaměřeno na konzistenci, čitelnost, a **modularitu** (rozdělní do funkcí).
  - *Také proto, že studenti tráví mnoho času implementací aniž by měli v implementaci nějaký výrazný postup.*

Hodnocení předmětu

Zdroj bodů	Maximum bodů	Přípustné minimum bodů
Domácí úkoly	45	-
Bonusové úkoly	10^+	-
Test v semestru	5	-
Písemný zkouškový test	20	10
Implementační zkouška	20	**10**
Součet	100^+ bodů	

- **Zápočet**: nejméně 30 bodů ze semestru a odevzdáné všechny domácí úkoly a to **nejpozději do 14.1.2023 ve 23:59 CET!**
- Předmět lze úspěšné ukončit **zápočtem** a **zkouškou**.
- Test a písemná zkouška – krátké stručné odpovědi prokazující porozumění problematice.

`https://cw.fel.cvut.cz/wiki/courses/b0b36prp/resources/test`

- Implementační zkouška – prokázaní samostatně porozumět a napsat krátký program.

`https://cw.fel.cvut.cz/wiki/courses/b0b36prp/resources/exam`

Klasifikace předmětu

Klasifikace	Bodové rozmezí	Hodnocení	Slovní hodnocení
A	≥ 90	1	výborně
B	80–89	1,5	velmi dobře
C	70–79	2	dobře
D	60–69	2,5	uspokojivě
E	50–59	3	dostatečně
F	<50	4	nedostatečně

- Včasné odevzdáním všech domácích úkolů s povinným a **volitelným** zadáním (45 bodů).
- Bonusová úloha a bonusové odevzdání HW10 (10^+ bodů).
- Test v semestru (5 bodů).
- Písemná zkouška (20 bodů) .
- Implementační zkouška (20 bodů).
- **95 bodů** a více (A – výborně), **76 bodů** (C – dobře) – (20% ztráta).
- Body jsou indikátorem průběžných výsledků.

15 a více bodů je velmi slušný výsledek

Zkouška může známku zlepšit, ale také v případě zásadní neznalosti zhoršit!

Přehled přednášek

- 01 - Informace o předmětu, Úvod do programování — *S. G. Kochan: kapitoly 1–3*
  Státní svátek
- 02 - Základy programování v C — *S. G. Kochan: kapitoly 2–5 a část 6*
- 03 - Řídicí struktury, výrazy a funkce — *S. G. Kochan: kapitoly 4–6 a 12*
- 04 - Pole, ukazatel, textový řetězec, vstup a výstup programu — *S. G. Kochan: kapitoly 7, 10 a 11*
- 05 - Ukazatele, paměťové třídy a volání funkcí — *S. G. Kochan: kapitoly 8 a 11*
- 06 - Struktury a uniony, přesnost výpočtů a vnitřní reprezentace číselných typů — *S. G. Kochan: kapitoly 9, 14, 17 a Appendix B*
- 07 - Standardní knihovny C. Rekurze. (Základní vlastnosti jazyka C probrány.)
- 08 - Spojové struktury — *S. G. Kochan: kapitola 16 a Appendix B*
- 09 - Abstraktní datový typ (ADT) - zásobník, fronta, prioritní fronta
- 10 - Stromy
- 11 - Prioritní fronta, halda. Příklad použití při hledání nejkratší cesty v grafu
- 12 - Stručný úvod do C++
- 13 - C++ konstrukty v příklade
- 14 - Rezerva

Přednáška nemusí být prezentace slidů – je očekávána interakce, řešení dotazů a diskuse problematických a náročnějších částí.

Podklady k přednášce jsou k dispozici před přednáškou podobně jako **učebnice**.

Část II

Část 2 – Zadání 0. domácího úkolu (HW00)

Zadání 0. domácího úkolu HW00

Téma: **První program**

Povinné zadání: **1b**; Volitelné zadání: není; Bonusové zadání: není

- **Motivace**: Seznámení se s odevzdávacím systémem BRUTE.
- **Cíl**: Osvojit si kompilaci a odevzdávání domácích úkolů .
- **Zadání**: `https://cw.fel.cvut.cz/wiki/courses/b0b36prp/hw/hw00`
  - Napište program, který vytiskne na obrazovku text `Hello PRP!` zakončený znakem nového řádku `\n`.
- Termín odevzdání: 01.10.2022, 23:59:59 PDT.

PDT – Pacific Daylight Time

Část III

Část 3 – Programování v C

Program je „recept“

- Program je posloupnost kroků (výpočtů) popisující průběh výpočtu pro řešení problému.

  (je to „recept“ řešení problému)
- Pro zápis receptu potřebujeme jazyk.

  Je to způsob zápisu programu.
- Jazyk definuje základní sadu primitiv (operací/příkazů), které můžeme použít pro zápis receptu.
- V předmětu B0B36PRP používáme programovací jazyk **C**.

  Programování není jen o znalosti konkrétního programovacího jazyka, je to o způsobu uvažování a řešení problému. Jazyk C nám dává příležitost osvojit si základní koncepty, které lze využít i v jiných jazycích.

Programu v Cčku

- Pamětová místa s daty jsou „odkazována“ proměnnými.
  - Typ proměnné definuje kolik paměti je použito pro uložení dat (číselné) hodnoty.
  - Např. zavedení proměnných pro uložení celých čísel typu `int`.

```
int a;
int b;
int c;
```

- Dále používáme obvyklý zápis operací.

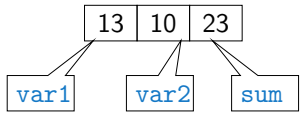
```
a = 10;
b = 4;
c = a + b;
```

Zápis uloží hodnotu 10 na paměťové místo odkazované proměnnou `a`, hodnotu 4 na paměťové místo odkazované proměnnou `b` a následně provede součet hodnot, který uloží na pamětové místo odkazované proměnnou `c`.

Přiřazení, proměnné a paměť – Vizualizace `unsigned char`

```
1 unsigned char var1;
2 unsigned char var2;
3 unsigned char sum;
4
5 var1 = 13;
6 var2 = 10;
7
8 sum = var1 + var2;
```

- Každá z proměnných alokuje právě 1 byte.
- Obsah paměti není po alokaci definován.
  Undefined behavior
- Jméno proměnné „odkazuje“ na paměťové místo.
- Hodnota proměnné je obsah paměťového místa.



Program v C

- Program v C je organizován do funkcí.
- Spustitelný program vyžaduje funkci, která se spustí jako první.
- V Cčku je to funkce `main()`.

```
1  void main(void)
2  {
3      int a;
4      int b;
5      int c;
6
7      a = 10;
8      b = 4;
9      c = a + b;
10 }
```

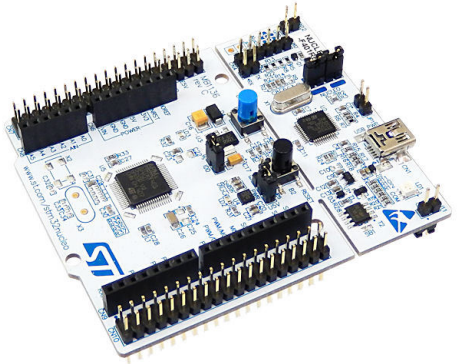
lec01/add.c

Takový program můžeme zkompilovat a spustit, ale úplně nemáme přehled co dělá a jaký je výsledek. Program přímo neinteraguje s uživatelem.

Interaktivní program

- Proměnné (paměťová místa) mohou přímo reprezentovat periferie - např. tlačítko (0 - stisknuto) a LED (1 - svítí).
- Ovládání LED tlačítkem tak můžeme realizovat jako nekonečnou smyčku, ve které nastavujeme hodnotu LED podle stisknutého nebo nestiknutého tlačítka

```
1  #include "mbed.h"
2
3  DigitalOut myled(LED1);
4  DigitalIn mybutton(USER_BUTTON);
5
6  int main()
7  {
8     while (1) {
9        if (mybutton == 0) {
10          myled = 1;
11       } else {
12          myled = 0;
13       }
14    }
15 }
```



Textově orientovaná interakce s uživatelem

- Dalším ze způsobů interakce s uživatelem je textový výstup a vstup.
- V případě programu běžícího v rámci operačního systému je nutné využívat služby operačního systému, který realizuje interakci s uživatelem s využitím hardwarových prostředků. *OS realizuje hardwarovou abstrakci.*
- V Cčkovém programu proto přidáme podporu pro vstup a výstup, knihovnu `stdio.h`.

```c
#include <stdio.h> // Vložení deklarací funkcí pro vstup/výstup

int main(void)
{
   printf("I like B0B36PRP!\n"); // Volání funkce printf()

   return 0;
}
```

lec01/program.c

Program vrací návratovou hodnotu OS a tím komunikuje s uživatelem nebo nadřazeným programem, který tak může identifikovat jakým způsobem byl program ukončen. V tomto případě neřešíme.

Zápis a kompilace programu

- Zdrojový kód programu v jazyce C se zapisuje do textových souborů (**.c** a také **.h**).

Jméno nebo koncovka není pro typ souboru úplně zásadní!

- Kompilací zdrojových souborů překladačem do binární podoby vznikají objektové soubory (s koncovkou **.o**) nebo spustitelný program.

Zdrojový soubor `program.c` přeložíme do spustitelné podoby kompilátorem např. `clang` nebo `gcc`.

```
clang program.c
```

Vznikne soubor `a.out`, který můžeme spustit např.

```
./a.out
```

Nebo kompilujeme do souboru **program**.

```
clang program.c -o program
```

```c
#include <stdio.h> // Vložení
    deklarací funkcí pro
    vstup/výstup

int main(void)
{
   printf("I like B0B36PRP!\n");
    // Volání funkce printf()

   return 0;
}
```

lec01/program.c

Příklad součtu dvou hodnot

```c
#include <stdio.h>

int main(void)
{
  int sum; /* definice lokalni promenne typu int */

  sum = 100 + 43; /* hodnota vyrazu se ulozi do sum */
  printf("The sum of 100 and 43 is %i\n", sum);
  /* %i formatovaci prikaz pro tisk celeho cisla */
  return 0;
}
```

- Proměnná `sum` typu `int` reprezentuje celé číslo, jehož hodnota je uložena v paměti
- `sum` je námi zvolené symbolické jméno (**identifikátor**) místa v paměti, kde je uložena celočíselná hodnota (typu `int`).

Standardní vstup a výstup

- Spuštěný program v prostředí operačního systému má přiřazený znakově orientovaný standardní vstup (`stdin`) a výstup (`stdout`), případně standardní chybový výstup (`stderr`).

Výjimkou jsou programy pro MCU bez OS.

Grafické programy mají grafický výstup a vstup z dalších periférií. Princip je obdobný jen komplexnější.

- Program může prostřednictvím `stdout` a `stdin` komunikovat s uživatelem.
- Základní funkce pro znakový výstup je `putchar()` a pro vstup `getchar()` definované ve standardní knihovně `<stdio.h>`.
- Formátovaný výstup je možné tisknout funkcí `printf()`, např. číselné hodnoty.
- Formátované načítání číselných hodnot lze realizovat funkcí `scanf()`, uživatel ale nemusí na vstup zadat číslo. Proto je vhodná kontrola úspěšného načtení čísla!

Jedná se o knihovní funkce, ze standardní knihovny. Jména funkcí nejsou klíčová slova jazyka C.

Formátovaný výstup – printf()

- Číselné hodnoty lze tisknout (vypsat) na standardní výstup prostřednictvím funkce `printf()`.

  `man printf`, *resp.* `man 3 printf`.
- Argumentem funkce je textový řídicí řetězec formátování výstupu.
- Řídicí řetězec formátu je uvozen znakem `'%'`.
- Znakové posloupností (nezačínající %) se vypíší tak jak jsou uvedeny.
- Základní řídicí řetězce pro výpis hodnot jednotlivých typů.

`char`	`%c`
`_Bool`	`%i`, `%u`
`int`	`%i`, `%x`, `%o`
`float`	`%f`, `%e`, `%g`, `%a`
`double`	`%f`, `%e`, `%g`, `%a`

- Dále je možné specifikovat počet vypsaných míst, zarovnání vlevo (vpravo), atd.

  Více na cvičení a v domácích úkolech.

Formátovaný vstup – scanf()

- Číselné hodnoty ze standardního vstupu lze načíst funkcí `scanf()`.

  `man scanf`, *resp.* `man 3 scanf`.
- Argumentem je textový řídicí řetězec s podobným syntax jako `printf()`.
- Funkce uloží načtenou hodnotu na konkrétní paměťové místo.

  Paměť musí být alokovaná a dostatečně velká, proto předáváme adresu proměnné konkrétního typu.
- Příklad načtení hodnoty celého čísla s kontrolou a výpisem na standardní chybový výstup.

```
1 int i;
2
3 printf("Enter int value: ");
4 int r = scanf("%i", &i); /* operator & vraci adresu promenne i */
5 if (r == 1) {
6    fprintf(fstdout, "You entered %02i\n", i);
7 } else {
8    fprintf(fstderr, "ERROR: Input does not contain valid integer value!\n");
9 }
```


Příklad formátovaného vstupu

```
1  #include <stdio.h>
2  #include <stdlib.h> // for EXIT_SUCCESS
3
4  int main(void)
5  {
6     int ret = EXIT_SUCCESS;
7     double d;
8
9     printf("Enter a double value: ");
10    int r = scanf("%lf", &d);
11    if (r == 1) {
12       printf("You entered %0.1f\n", d);
13    } else if (r == 0) {
14       fprintf(stderr, "ERROR: Input does not match double value!\n");
15       ret = 101;
16    } else {
17       fprintf(stderr, "WARN: No input provided!\n"); //press Ctrl+D to terminate the
        input EOT - End-of-Transmission character (or Ctrl+Z on Disk Operating System /
        Windows like systems)
18       ret = 102;
19    }
20    return ret;
21 }
```

lec01/scanf.c

Základní koncepty programování

V programování jsou využívány tři klíčové koncepty, kterou jsou vzájemně kombinovány a umožňují vytvářet komplexní programy.

- **Přiřazení** - uložení hodnoty na definované místo v paměti
- **Větvení** - volba posloupnosti instrukcí na základě hodnoty nějaké proměnné (místa v paměti)
- **Cyklus** - Opakování nějaké posloupnosti instrukcí s novými daty

Abychom mohli lépe a snadněji organizovat posloupnosti instrukcí do složitější celků, je vhodné program strukturovat do znovupoužitelných částí: **procedur** a **funkcí**

- Procedura představuje předpis co se má s jednolivými paměťovými místy provádět
- Výsledek procedury závisí na hodnotách uložených v paměti
- Procedura/funkce/algoritmus řeší obecnou úlohu nějakého výpočtu

Neméně důležitým konceptem je zobecňování výpočtu, které „zjednodušuje“ řešení problémů.

Příklad opakovaného tisku na základě uživatelského vstupu

- Úkol: Uživatel zadá počet opakování tisku zprávy a pokud je počet větší než 0 a zároveň menší než 10 vypíše zprávu tolikrát kolik bylo zadáno. V opačném případě upozorní uživatele na omezený rozsah.
  - **Přiřazení** - uložení hodnoty počtu opakování od uživatele (proměnná `n`).
  - **Větvení** - kontrola mezí vstupní hodnoty.
  - **Cyklus** - opakování vypisu *n* krát.
    - Při opakovaném průchodu cyklem počítáme kolikrát byla zpráva vytištěna (řídicí proměnná `i`).

Detalní popis a vysvětlení syntaxe v učebnici a další přednášce.

Opakovaný tisk textové zprávy uživateli

- Zprávu lze například vytisknout 4× opakování příkazu tisku.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5     printf("I like B0B36PRP!\n");
6     printf("I like B0B36PRP!\n");
7     printf("I like B0B36PRP!\n");
8     printf("I like B0B36PRP!\n");
9     return 0;
10 }
```

- Použití cyklu a řídicí proměnné je správnější programátorský přístup.

Příklad řešení

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5     int ret = 0; //0 is EXIT_SUCCESS
6     int n;
7     printf("Enter a positive integer number from 1 to 9: ");
8     int r = scanf("%d", &n);
9     if (r == 1 && n > 0 && n < 10) {
10       int i = 0;
11       while (i < n) {
12          printf("I like B0B36PRP!\n");
13          i = i + 1;
14       }
15    } else {
16       printf("Input value must be in the range (0,10)\n");
17       ret = -1;
18    }
19    return ret;
20 }
```

lec01/print.c

Výpočetní problém, algoritmus a program jako jeho řešení

Příklad: Najít největšího společného dělitele čísel 6 a 15.

- Víme co musí platit pro číslo d, aby bylo největším společným dělitele čísel x a y.
- Známou deklarativní znalost o problému můžeme využít pro návrh výpočetního postupu jak takové číslo najít, např.
  1. Nechť máme nějaký odhad čísla d;
  2. Potom můžeme ověřit, zdali d splňuje požadované vlastnosti ;
  3. Pokud ano, jsme u cíle;
  4. Pokud ne, musíme d vhodně modifikovat a znovu testovat.
- Výpočetní problém chceme vyřešit využitím konečné množiny primitivních operací počítače.
- Konkrétní úlohu pro čísla 6 a 15 zobecňujeme pro „libovolná“ čísla x a y, pro který **navrhneme algoritmus**.
- Algoritmus následně přepíšeme do programu využitím konkrétního programovacího jazyka.

Příklad největší společný dělitel

- Úloha

  Najděte největší společný dělitel čísel 6 a 15.

  Co platí pro společného dělitele čísel?

- Řešení

  Návrh postupu řešení pro dvě libovolná přirozená čísla.

  *Definice **vstupu** a **výstupu** algoritmu.*

  - Označme čísla x a y.
  - Vyberme menší z nich a označme jej d.
  - Je-li d společným dělitelem x a y končíme.
  - Není-li d společným dělitelem pak zmenšíme d o 1 a opakujeme test až d bude společným dělitelem x a y.

- Symboly x, y a d reprezentují **proměnné** (paměťové místo), ve kterých jsou uloženy hodnoty, které se v průběhu výpočtu mohou měnit.

Slovní popis činnosti algoritmu

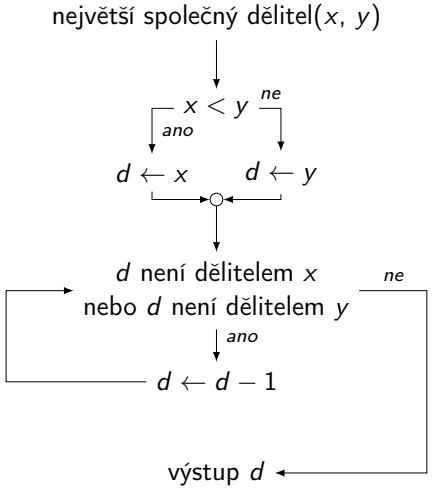
- Úloha:

  Najít největší společný dělitel přirozených čísel x a y.

- Popis řešení
  - **Vstup**: dvě přirozená čísla x a y.
  - **Výstup**: přirozené číslo d – největší společný dělitel x a y.
  - **Postup**
    1. Je-li $x < y$, pak d má hodnotu x, jinak má d hodnotu y.
    2. Pokud d není dělitelem x nebo d není dělitelem y opakuj krok 3, jinak proveď krok 4.
    3. Zmenši d o 1.
    4. Výsledkem je hodnota d.

Algoritmus = výpočetní postup jak zpracovat vstupní data a určit (vypočítat) požadované výstupní hodnoty (data) s využitím elementárních výpočetních instrukcí a pomocných dat.

Postup výpočtu algoritmu vyjádřený formou vývojového diagramu



Zápis algoritmu v pseudojazyku

- Zápis algoritmu využitím klíčových a dobře pochopitelných slov

Algoritmus 1: Nalezení největšího společného dělitele

Vstup: x, y – kladná přirozená čísla

Výstup: d – největší společný dělitel x a y

```
if x < y then
    d ← x;
else
    d ← y;
while d není dělitelem x nebo d není dělitelem y do
    d ← d − 1;
return d
```

Neodpovídá přesně zápisu programu v konkrétním programovacím jazyku, ale je čitelný a lze velmi snadno přepsat.

Zápis algoritmu v C – motivační ukázka

```
1  int getGreatestCommonDivisor(int x, int y)
2  {
3     int d;
4     if (x < y) {
5        d = x;
6     } else {
7        d = y;
8     }
9     while ( (x % d != 0) || (y % d ! = 0)) {
10       d = d - 1;
11    }
12    return d;
13 }
```

- Úlohu největšího společného dělitele čísel 6 a 15 jsme zobecnili.
- A dekomponovali na funkci `getGreatestCommonDivisor()`.
- Funkci můžeme opakovaně použít.
- Případně definovat v samostatném souboru.

- Nebo také s využitím ternárního operátoru.

podmínka ? výraz : výraz

Více učebnice, cvičení nebo příští přednáška.

```
1  int getGreatestCommonDivisor(int x, int y)
2  {
3     int d = x < y ? x : y;
4     while ( (x % d != 0) || (y % d ! = 0)) {
5        d = d - 1;
6     }
7     return d;
8  }
```

lec01/demo-gcd.c

Část IV

Část 4 – Zadání 1. domácího úkolu (HW01)

Zadání 1. domácího úkolu HW01

Téma: **Načítání vstupu, výpočet a výstup**

Povinné zadání: **1b**; Volitelné zadání: není; Bonusové zadání: není

- **Motivace**: Získat představu o interakci uživatele s programem.
- **Cíl**: Osvojit si načítání vstupu, formátovaného výstupu a základní posloupnosti příkazů
- **Zadání**: `https://cw.fel.cvut.cz/wiki/courses/b0b36prp/hw/hw01`
  - Načítání celých čísel ze standardního vstupu.

    (čísla v rozsahu [-10 000; 10 000])
  - Výpis čísel v dekadické a šestnáckové soustavě.
  - Provedení základní aritmetických operací s načtenými čísly.
  - Výpočet podílu a průměrné hodnoty čísel.
  - Dodržení správného formátování výstupu.

    Použijte hex zobrazení výstupu – `hexdump -C`.
- Termín odevzdání: 08.10.2022, 23:59:59 PDT.

PDT – Pacific Daylight Time

Shrnutí přednášky

Diskutovaná témata

- Informace o předmětu
- Procedurální programování (v C)
- Standardní vstup a výstup programu
- Formátovaný vstup a výstup

- Příště: Základy programování v C