Úvod do programování v C

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 01

BAB36PRGA – Programování v C

Přehled témat

- Část 1 – Organizace předmětu
  - Cíle předmětu a prostředky jejich dosažení
  - Úvod do programování
- Část 2 – Zadání 0. domácího úkolu (HW0)
- Část 3 – Programování (v C)
  - První program (v C)
  - Základních koncepty programování

S. G. Kochan: kapitoly 2, 3

- Část 4 – Zadání 1. domácího úkolu (HW1)

Část I

Organizace předmětu

Předmět a přednášející

BAB36PRGA – Programování v C

- Webové stránky předmětu https://cw.fel.cvut.cz/wiki/courses/bab36prga
- Odevzdávání domácích úkolů https://cw.felk.cvut.cz/brute
- Přednášející:
  - prof. Ing. **Jan Faigl**, Ph.D.

- Detailní informace o předmětu, organizaci a hodnocení na 1. cvičení a na https://cw.fel.cvut.cz/wiki/courses/bab36prga.

**lec*00*.pdf*

Organizace a hodnocení **BAB36PRGA** – Programování v C

- 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.
- **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!**

Zdroje a literatura

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

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

Recommended textbook.

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

More like a reference manual, still comprehensive textbook.

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

Zdroje a literatura

- **Knihy (učebnice)**

Základní učební text „Programming in C“ (Kochan, 2014)

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

Recommended textbook.

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

More like a reference manual, still comprehensive textbook.

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 učebnice jazyka C

Practical C Programming, *Steve Oualline*, O'Reilly Media, Inc., 3rd edition, 1997)

Briefer than Kochan's textbook, still comprehensive.

Effective C: An Introduction to Professional C Programming, *Robert C. Seacord*, William Pollock, 2020.

Great if you already known some of C syntax and like to improve your skill further.

Fluent C, Principles, Practices, and Patterns, *Christopher Preschern*, O'Reilly Media, Inc., 2022.

Suitable if you like to know more about coding practices.

21st Century C: C Tips from the New School, *Ben Klemens*, O'Reilly Media, 2012.

Obsah

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.

Způsob výuky programování v BAB36PRGA

- 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 je zásadní.

  Na začátku semestru čas pro čtení učebnice!
- Vědět a umět použít (nikoliv “slepovat”). *Nezávislost na našeptávači!*
  - Začínáme relativně jednoduchými úlohami k osvojení programovacích konstruktů a způsobu organizace zdrojového kódu. *Přehledný kód a schopnost se efektivně orientovat v kódu!*
  - *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 dopňují 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).

Přehled přednášek

Státní svátek

- 01 - Úvod do programování v C — *S. G. Kochan: kapitoly 1–3*
- 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 - Paralení programování, paralelní výpočty a synchronizační primitiva (semafory, zprávy a sdílená paměť
- 11 - Vícevláknové programování, modely aplikací, POSIX vlákna C11 vlákna
- 12 - ANSI C, C99, C11 a rozdíly mezi C a C++. Úvod do C++ v příkladech
- 13 - *Rezerva* - Rektorský den
- 14 - Stručný úvod do C++ (v příkladech)

Přednáška není jen prezentace slidů – interakce, řešení dotazů a diskuse.

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

Přehled domácích úkolů

- Domácí úkoly s povinným, volitelným, případně bonusovým zadáním. *52 h, bonus +28 h*
  `https://cw.fel.cvut.cz/wiki/courses/b3b36prga/hw/start`
  0. HW 0 - První program *1 h*
  1. HW 1 - Načítání vstupu, výpočet a výstup
     (**Kontrola přehlednosti kódu** – až -100% z dosažených bodů) *4 h*
     Seznámení se s prostředím, psaním programů, jejich laděním, testováním a odevzdáváním. ∼ 20–40 h
  2. HW 2 - Kreslení (ASCII art) (**Kontrola kódu** – až -100%) *5 h*
  3. HW 3 - Prvočíselný rozklad (**Kontrola kódu** – až -100%) *5 h, bonus +8 h*
  4. HW 4 - Caesarova šifra (**Kontrola kódu** – až -100%) *6 h, bonus +6 h*
  5. HW 5 - Hledání textu v souborech *6 h*
  6. HW 6 - Maticové počty (**Kontrola kódu** – až -100%) *6 h bonus +6 h*
  7. HW 7 - Kruhová fronta v poli - *Dynamicky linkovaná knihovna* *6 h*
  8. HW 8 - Fronta spojovým seznamem s řazením *7 h*
  9. HW 9 - Vícevláknová aplikace s meziprocesovou komunikací. *6 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 **30b**, volitelné zadání **15b**, bonusové **20b**.

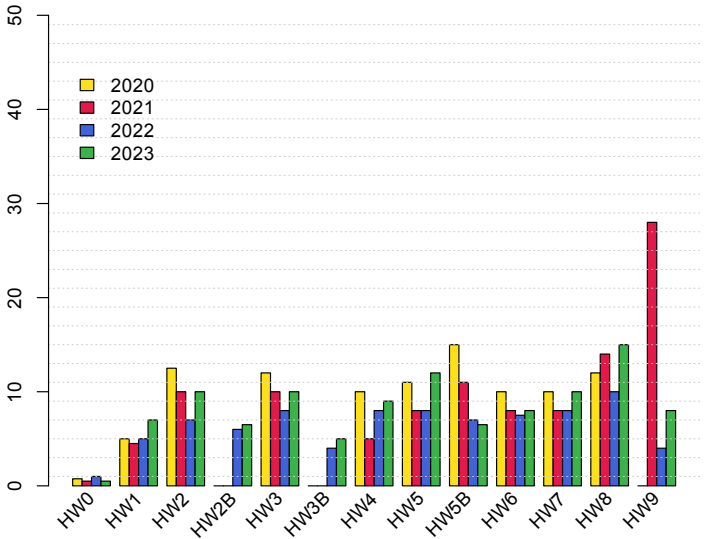
Medián reportované časové náročnosti vypracování úloh

- Očekáváná náročnost cca 80 h.
  - Součet mediánů (bez bonus úloh) očekáván v rozsahu cca 52 h.
  - Bonusy cca 28 h.
  - Celkem cca 72 h.
- Od HW3 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, 16 h zkouška, 80 h úlohy, *"rezerva" cca 42 h*.
- Změna úloha v roce 2023 a 2024.

  HW1 a HW9 (HW8 z 2023 přesun do lab12 a lab13.



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.
  - Cíl není odevzdat úkoly v BRUTE, je to prostředek ověření funkčnosti programu.
  - BRUTE je nástroj průběžné kontroly postupu a získávání znalostí.
  - Cíl je naučit se **samostatně programovat** funkční programy správně.
- 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 (discord), cvičících, na přednášce nebo **konzultaci**.
- Úkoly HW1–HW4 a HW6 budou kontrolovány na správnost a přehlednost kódu.
  - Zaměřeno na konzistenci, čitelnost, a **modularitu** (rozdělní do funkcí).

    Z hlediska tréninku a učení, i zdánlivě triviální program se snažte rozumně rozdělit na více funkcí.
  - *Motivace je netrávit příliš mnoho času implementací bez výrazného postupu.*

Obsah

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` $y^2 \approx x$ *Využití deklarativní znalosti.*
2. `Then`
   return y
3. `Else`
   $y \leftarrow \frac{y + \frac{x}{y}}{2}$
   `Go to Step 1`

- Program je „recept“ – posloupnost kroků (výpočtů) popisující průběh řešení problému.

Program a programování

Základní koncept je princip přiřazení hodnoty proměnné!

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

- *Co je* `int`*?*
- *Co je* `a` *a* `b`*?*
- *Co je* `=` *na řádku 1–2 a na řádku 4?*

- Program (posloupnost instrukcí v paměti programu) je předpis práce s pamětí dat.
- Instrukce přesouvají hodnoty v paměti, počítají a testují (porovnávají) hodnoty.
  - **Posloupnost** instrukcí (příkazů).
  - **Větvení** - podmíněný skok na posloupnost instrukcí v závislosti na hodnotě proměnné.
  - **Cykly** - opakování stejné poslopnosti instrukcí s jinými daty.
- Programování je schopnost **samostatně**
  - **tvořit programy**;
  - **dekomponovat** úlohy na menší celky;
  - sestavovat z **dílčích částí větší programy** řešící komplexní úlohu.
- Programování je dovednost (řemeslo) zapsat řešení výpočetního problému programem.

Čitelný, srozumitelný a udržovatelný zápis!

Část II

Část 2 – Zadání 0. domácího úkolu (HW0)

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/bab36prga/hw/hw0`
  - Napište program, který vytiskne na obrazovku text `Hello PRG!` zakončený znakem nového řádku `\n`.
- Termín odevzdání: 10.03.2024, 23:59:59 PST.

PST – Pacific Standard Time

Část III

Část 3 – Programování (v C)

Obsah

Program a definice programovacího jazyka

- Program je posloupnost kroků (výpočtů) popisující průběh výpočtu řešení problému.
- Programovací jazyk je způsob zápisu programu, který definuje sadu konstrukcí programu.

Zapisovaný v textové („lidsky čitelné“) podobě.

- **Syntax** – definice povolených výrazů a konstrukcí programu.

Plná kontrola a podpora vývojových prostředí.

  - Příklad popisu výrazu gramatikou v Backus-Naurově formě
    <exp> ::= <exp> + <exp> | <exp> * <exp> | <exp> | <number> <number> ::= <number> <digit> | <digit> <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.
- **Statická sémantika** – definuje jak jsou konstrukty používány.

Částečná kontrola a podpora prostředí.

  - Příklad axiomatické specifikace: $\{P\}\ S\ \{Q\}$,
    - P - precondition,
    - Q - postcondition,
    - S - konstrukce jazyka.
- **Plná sémantika** – co program znamená a dělá, jeho smysluplnost.

Kontrola a ověření správnosti je v režii programátora/ky.

Správnost programu

- Syntakticky i staticky sémanticky správný program neznamená, že dělá to co od něj požadujeme.
- Správnost a smysluplnost programu je dána očekávaným chováním při řešení daného problému.
- V zásadě při spuštění programu mohou nastat následující události.
  - Program havaruje a dojde k chybovému výpisu.

    Mrzuté, ale výpis (report) je dobrý start řešení chyby (bug).
  - Program běží, ale nezastaví se, počítá v nekonečné smyčce.

    Zpravidla velmi obtížné detekovat a program ukončujeme po nějaké době, proto je vhodné mít představu o výpočetní náročnosti řešené úlohy a použitém přístup řešení (algoritmu).
  - Program včas dává odpověď.

    Je dobré vědět, že odpověď je korektní.

Správnost programu je plně v režii programátorky nebo programátora, proto je důležité pro snadnější ověření správnosti, ladění a hledání chyby používat **dobrý programovací styl**.

Program a jeho zápis

- V předmětu BAB36PRGA používáme programovací jazyk **C**.

  Programování není 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.

- Klíčové pro správné fungování programu je zacházení s pamětí.

  Cílem kurzu PRGA je naučit se základním principům, které lze následně generalizovat též pro jiné programovací jazyky. Pochopení těchto principů je klíčem k efektivnímu psaní efektivních programů.

- Program se skládá z
  - **klíčových slov**, **výrazů** (operátorů/volání funkcí), **literálů** (hodnot);
  - **identifikátorů** proměnných a funkcí.

  Identifikátor proměnné je pojmenování datové oblasti v paměti.

  Identifikátor funkce je pojmenování posloupnosti instrukcí (příkazů).

Platné znaky pro zápis zdrojových souborů

- Malá a velká písmena, číselné znaky, symboly a oddělovače.

  ASCII – American Standard Code for Information Interchange.

  - a–z A–Z 0—9
  - ! " # % & ' ( ) * + , - . / : ; < = > ? [\ ] ^ _ { | } ~
  - mezera, tab, nový řádek.
- Escape sekvence pro symboly
  - \' – ', \" – ", \? – ?, \\– \.
- Escape sekvence pro tisk číselných hodnot v textovém řetězci
  - \o, \oo, kde o je osmičková číslice;
  - \xh, \xhh, kde h je šestnáctková číslice.

```
1  int i = 'a';
2  int h = 0x61;
3  int o = 0141;
4
5  printf("i: %i h: %i o: %i c: %c\n", i, h, o, i);
6  printf("oct: \141 hex: \x61\n");
```

Např. \141, \x61 lec01/esqdho.c

- \0 – znak pro konec textového řetězce (null character).

Klíčová slova

- Klíčová (rezervovaná) slova (**keywords**)$_{32}$:

  auto break case char const continue default do double else enum extern float for goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while. C98

- Mezi klíčová slova patří označení typu hodnoty, které definuje potřebné paměťové místo reprezentace čísla: `int`, `double`, `float`, `char`, `short`, `long`.
  - Hodnoty jsou literály, výsledek výrazů, proměnné, návratová hodnota funkce, operandy.

  Každá hodnota (číslo) má svou reprezentaci (typ) a potřebnou velikost paměti.

Zápis programu

```c
/*
 * Funkce main je hlavní funkce programu v c, je spuštěna
 * po spuštění programu.
 */
int main(void)     // Funkce main vrací hodnotu typu int.
{
   int a = 10;     // Definice proměnné a typu int.
                   // inicializace literálem 10 (typ int).

   int b = 20;     // Definice proměnné je přirazení symbolu
                   // alokovanému paměťovému místu.

   int c;          // Hodnota proměnné c není definována.

   c = a + b / 2; // Výraz má hodnotu typu int.
                   // Dělení je celočíselné (oba operandy typu int).

   return c;       // Návratová hodnota funkce (v případě main(), návratová hodnota programu).
}
```

- Použitá klíčová slova `int`, `void` a `return`.
- Literály jsou `10`, `20` a `2`.
- Použité operace ve výrazu jsou sčítání, dělení a přirazení.
- Identifikátory jsou jména proměnných `a`, `b`, `c` a jméno funkce `main()`.

Jméno hlavní funkce `main()` *je předepsané.*

Literály – Zápis hodnot

- Hodnoty datových typů označujeme jako **literály**.
- Zápis celých čísel (celočíselné literály).
  - dekadický 123 450932
  - šestnáctkový (hexadecimální) 0x12 0xFAFF (začíná `0x` nebo `0X`)
  - osmičkový (oktalový) 0123 0567 (začíná `0`)
  - `unsigned` 12345U (přípona `U` nebo `u`)
  - `long` 12345L (přípona `L` nebo `l`)
  - `unsigned long` 12345ul (přípona `UL` nebo `ul`)
- Není-li přípona uvedena, jde o literál typu `int`. Výchozí typ Cčka.
- Neceločíselné datové typy `float` a `double` jsou dané implementací, většinou se řídí standardem IEEE-754-1985. `float`, `double`
  - `1.0` nebo `1.` výchozí typ necelého čísla je `double`.
  - `1.0f` nebo `1.f` literál typu `float`.

Identifikátory

- **Identifikátory** jsou jména proměnných, funkcí, a vlastních typů.
- Pravidla pro volbu identifikátorů. *Názvy proměnných, typů a funkcí.*
  - Znaky a–z, A–Z, 0–9 a _.
  - První znak není číslice.
  - Rozlišují se velká a malá písmena (case sensitive).
  - Délka identifikátoru není omezena.

  Prvních 31 znaků je významných – může se lišit podle implementace.
- **Proměnné** jsou data, typicky volíme **podstatná jména**.
- **Funkce** něco provádí, typicky volíme **slovesa**.

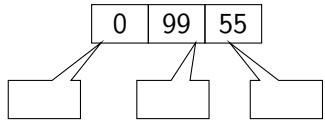
Cvičte se v používání angličtiny!

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.

Paměť není incializována.



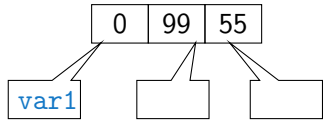
Přiřazení, proměnné a paměť – Vizualizace unsigned char

```
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.

Paměť není incializována.



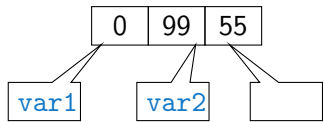
Přiřazení, proměnné a paměť – Vizualizace unsigned char

```
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.

Paměť není incializována.

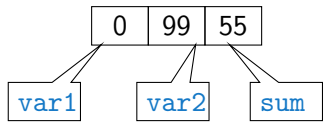


Přiřazení, proměnné a paměť – Vizualizace unsigned char

```
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.

Paměť není incializována.

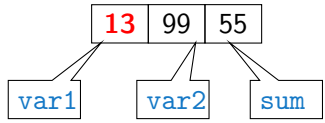


Přiřazení, proměnné a paměť – Vizualizace unsigned char

```
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.

Paměť není incializována.

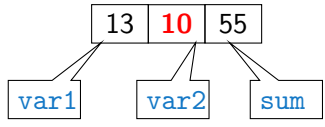


Přiřazení, proměnné a paměť – Vizualizace unsigned char

```
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.

Paměť není incializována.

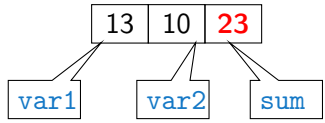


Přiřazení, proměnné a paměť – Vizualizace unsigned char

```
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.

Paměť není incializována.

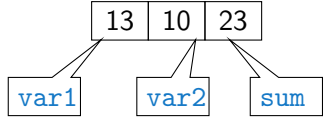


Přiřazení, proměnné a paměť – Vizualizace unsigned char

```
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.

Paměť není incializována.



Program v C

- Program v C je organizován do funkcí.
- Spustitelný program vyžaduje funkci, která se spustí jako první – `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 }
```

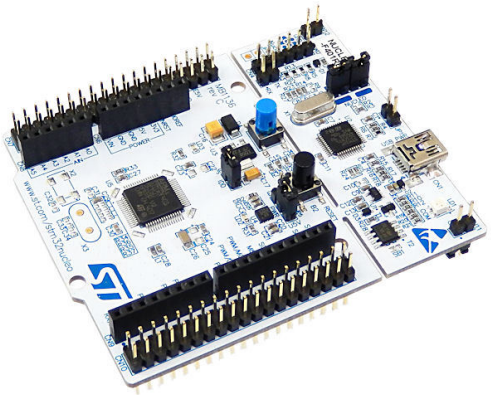
lec01/add.c

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

- Základním způsobem interakce s uživatelem je textový výstup a vstup.
- V případě programu běžícího v rámci operačního systému (OS) využíváme služby OS, 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
                   // Deklarace je hlavička funkce.
int main(void)
{
   puts("I like BAB36PRGA!"); // Volání funkce puts(), viz man puts.
   return 0; // 0 indikuje úspěšné provedení programu - EXIT_SUCCESS
}
```

lec01/program0.c

- Program vrací návratovou hodnotu OS a tím komunikuje s uživatelem nebo nadřazeným programem, který tak může identifikovat jakým způsobem byl program ukončen. *Viz cvičení, shell a hodnota* `$?`.
- Funkce `puts()` vrací návratovou hodnotu indikující úspěšné vykonání funkce, viz `man puts`.
- Náš první program tak v podstatě **obsahuje chybu**, protože volání `puts()` se nemusí povést.

Ošetření návratových hodnot volání funkcí

- Funkce `puts()` vrací `EOF` při chybě, jinak nezáporné celé číslo.
- K zpřehlednění využijeme symbolické konstanty `EXIT_SUCCESS` a `EXIT_FAILURE` z knihovny `stdlib.h`.

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

int main(void)
{
    if (puts("I like BAB36PRGA!") != EOF) {
        return EXIT_SUCCESS;
    } else {
        return EXIT_FAILURE;
    }
}
```

lec01/program.c

- Typicky k chybě na standardní (textový) výstup (`stdout`) nedochází, přesto je kontrola návratových hodnot funkcí zásadní, zejména při práci se soubory nebo načítání vstupu.

Zejména z důvodu přehlednosti, není v BAB36PRGA vyžadováno kontrolování návratových hodnot při výstupu na `stdout` *s využitím funkcí typu* `puts()` *nebo* `printf()`.

Hodnocení programů a programátorské styly

- Hodnocení programů je neosobní, program je správně nebo není. *Není to hodnocení Vás, ale programu.*
- Pokud program není správně, je to z hlediska učení se v pořádku, protože se učíte.

  Pokud je správně, už to umíte, už se neučíte. Tedy když se učíte, děláte chyby.
- Programování není o memorování fakt. Nezaměňujte memorování s programováním správně.
  - Naučit se fakta aplikovat v různých situacích, dokud nebudete vědět, jak je použít.
- Naučit se, znamená zkoušet, uvědomit si, vědět.

  Zkopírováním nebo vygenerováním funkčního programu se to zpravidla nenaučíte—programování vs. prompt engineerring.
- Všechny programy v PRGA už někdo někdy implementoval.

  Naučit se programovat znamená umět řešit problémy, které ještě nikdo neřešil.
- Kreativní programování je rychlé prototypování. *Nedělám chyby, program je funkční a dokonalý, jako jeho autor.*
- Defenzivní programování předpokládá, že programy mají chyby, které nemůžeme plně odstranit, ale pouze redukovat jejich četnost. Nejste váš program, ale jste zoodpovědní za chyby.
- Draftujte program kreativně. Pak se přepněte do defenzivního stylu a řešte možné chyby.
  - **Najděte odvahu program kompletně přepsat.** Nestyďte se programovat na načisto (klidně i na papír).
- Programování není o bušení do klávesnice, ale o návrhu, přemýšlení a strategii implementace.

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>
#include <stdlib.h>

int main(void)
{
    int ret = EXIT_SUCCESS;
    int r = puts("I like BAB36PRGA!");
    if (r == EOF) {
        ret = EXIT_FAILURE;
    }
    return ret;
}
```

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 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 kontrolujeme úspěšné 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(stdout, "You entered %02i\n", i);
7 } else {
8    fprintf(stderr, "ERROR: Input does not contain valid integer value!\n");
9 }
```

Příklad formátovaného vstupu

```c
#include <stdio.h>
#include <stdlib.h> // for EXIT_SUCCESS

int main(void)
{
   int ret = EXIT_SUCCESS;
   double d;

   printf("Enter a double value: ");
   int r = scanf("%lf", &d);
   if (r == 1) {
      printf("You entered %0.1f\n", d);
   } else if (r == 0) {
      fprintf(stderr, "ERROR: Input does not match double value!\n");
      ret = 101; // Indicate error on input, number has not been given.
   } else {
      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)
      ret = 102; // Indicate no input has been given.
   }
   return ret;
}
```

lec01/scanf.c

Obsah

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 počitáme 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× opakováním příkazu tisku.

```c
#include <stdio.h>

int main(void)
{
    printf("I like BAB36PRGA!\n");
    printf("I like BAB36PRGA!\n");
    printf("I like BAB36PRGA!\n");
    printf("I like BAB36PRGA!\n");
    return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
    const int N = 4;
    for (int i = 0; i < N; ++i) {
        printf("I like BAB36PRGA!\n");
    }
    return 0;
}
```

- Použití cyklu a řídicí proměnné je programátorský přístup.
- Příklad zoobecníme o zadání počtu opakování uživatelem ze standardního vstupu.

Příklad řešení 1/3

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

int main(void)
{
   int ret = EXIT_SUCCESS;
   int n; // hodnota není incializována
   printf("Enter a positive integer number from 1 to 9: ");
   int r = scanf("%d", &n);  // kontrola úspěšnosti načtení celého čísla %d
   if (r == 1 && n > 0 && n < 10) {
      int i = 0;
      while (i < n) {
         puts("I like BAB36PRGA!");
         i = i + 1;
      }
   } else {
      printf("ERROR: Input value must be in the range (0,10)\n");
      ret = EXIT_FAILURE;
   }
   return ret;
}
```

lec01/print.c

- Naivní, funkční řešení, v zásadě postačující, nicméně i takový program můžeme dekomponovat.

Příklad řešení 2/3

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

void print(int n);

int main(void)
{
   int ret = EXIT_SUCCESS;
   int n;
   printf("Enter a positive integer number from 1 to 9: ");
   int r = scanf("%d", &n); // předáváme adresu proměnné n, hodnotu vyplní funkce
   if (r == 1 && n > 0 && n < 10) {
      print(n);
   } else {
      fprintf("ERROR: Input value must be in the range (0,10)\n");
      ret = EXIT_FAILURE;
   }
   return ret;
}
```

lec01/print2.c

- Tisk v samostatné funkce `print()`.
- Lepší, ale stále relativně složité – můžeme oddělit načítání, ale také zoobecnit hodnoty a vyhnout se „magic numbers“ v definici funkcí.

Příklad řešení 3/3

```c
#include <stdio.h>
#include <stdlib.h> // Because of EXIT_SUCCESS

int read(int min, int max, int *n);
void print(int n);

#define MIN 1
#define MAX 9

int main(void)
{
   int ret = EXIT_SUCCESS;
   int n; // memory allocation for the read value
   if (read(MIN, MAX, &n)) {
      print(n);
   } else {
      printf("ERROR: Input value must be in the range (%d,%d)\n", MIN - 1, MAX + 1);
      ret = EXIT_FAILURE;
   }
   return ret;
}

int read(int min, int max, int *n)
{
   printf("Enter a positive integer number from %d to %d: ", min, max);
   return scanf("%d", n) == 1 && *n >= min && *n <= max;  // logical true is a value != 0, shortcut evaluation
}

void print(int n)
{
   int i = 0;
   while (i < n) {
      puts("I like BAB36PRGA!");
      i = i + 1;
   }
}
```

lec01/print3.c

- Funkci `read()` předáváme ukazatel na platnou adresu paměti, to zajišťujeme programově.
- Program vrací návratovou hodnotu a upozorňuje uživatele na chybný vstup. *Můžeme dále použít* `fprintf(stderr,)`.
- Hodnoty **MIN** a **MAX** můžeme dále rozšířit o možnost definování při překladu (`#ifndef`).

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.

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.

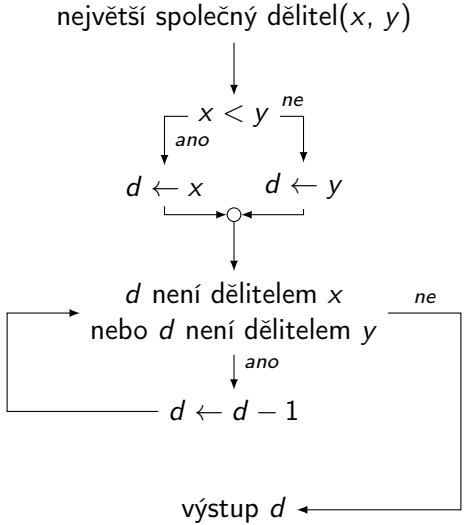
Slovní popis činnosti algoritmu

- 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**
  $d \leftarrow x$;
else
  $d \leftarrow y$;
while d *není dělitelem* x *nebo* d *není dělitelem* y **do**
  $d \leftarrow d - 1$;
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

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 (HW1)

Zadání 1. domácího úkolu HW1

Téma: Načítání vstupu

Povinné zadání: **3b**; Volitelné zadání: není; Bonusové zadání: není

- **Motivace**: „Automatizovat“ a zobecnit výpočet pro „libovolně“ dlouhý vstup.
- **Cíl**: Osvojit si využití cyklů jako základní programového konstrukce pro hromadné zpracování dat.
- **Zadání**: `https://cw.fel.cvut.cz/wiki/courses/bab36prga/hw/hw1`
  - Zpracování **libovolně dlouhé** posloupnosti celých čísel.
  - Výpis načtených čísel.
  - Výpis statistiky vstupních čísel.
    - Počet načtených čísel; Počet kladný a záporných čísel a jejich procentuální zastoupení na vstupu.
    - Četnosti výskytu sudých a lichých čísel a jejich procentuální zastoupení na vstupu.
    - Průměrná, maximální a minimální hodnota načtených čísel.
- Termín odevzdání: 16.03.2024, 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

Diskutovaná témata

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

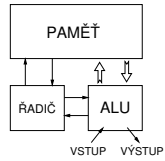
- Příště: Základy programování v C

Část VI

Appendix

Počítač „počítá“, pracuje s daty (čísly)

- Výpočet realizuje *aritmeticko-logická jednotka* (ALU).
- Číselné hodnoty jsou uloženy v paměti počítače – registry (ALU), paměť (RAM).
- Předpis jak a co počítat je zapsán programem – posloupností instrukcí.
- Instrukce jsou také číselné hodnoty (opcode), uložené v paměti (programu).



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

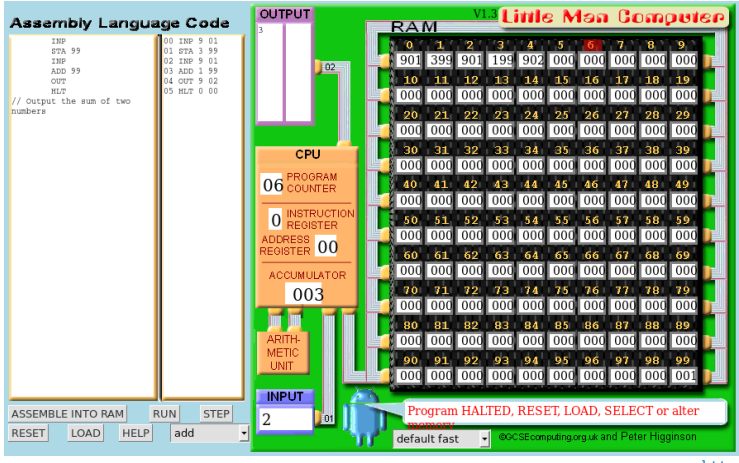
```
lda $10
add $04
sto r3
```

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

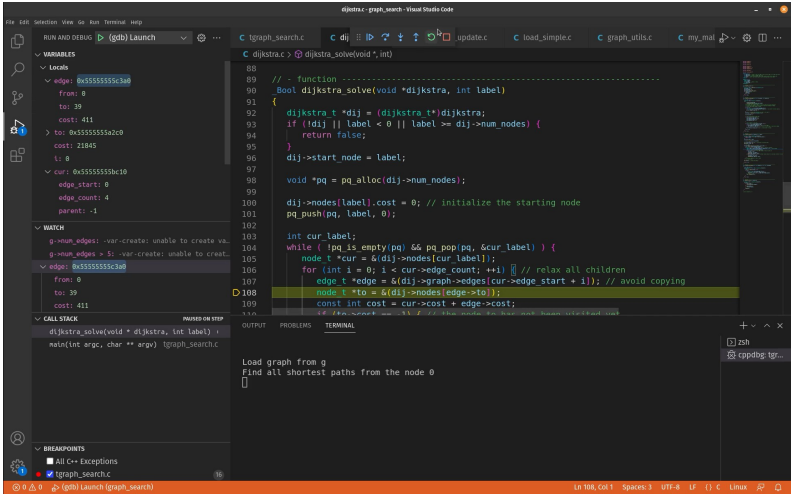


Základní instrukce:

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

Příklad ladění krokováním



https://youtu.be/rTv_ypcm9XI (∼ 25 min)